ON CRYPTOGRAPHICAL PROTOCOLS AND PROCEDURES: NTRU AND THE SHORTEST VECTOR PROTECTION

Master Thesis in Applied Computer Science
submitted to the Faculty of Mathematics and Computer Science
of Georg-August-Universität Göttingen
on October 16, 2015

by
Fabian Bürger

Supervisor
Prof. Dr. Preda Mihăilescu

Second Supervisor
Prof. Dr. Carsten Damm
Contents

1 Prologue 2

2 Introduction 3
   2.1 Modern Cryptography 3
   2.2 Secret Key Encryption 4
   2.3 Problems of Secret Key 7
   2.4 Public Key Encryption 8

3 The LLL Algorithm 12
   3.1 Lattices 12
   3.2 The Shortest Vector Problem 14
   3.3 Gram-Schmidt Process 16
   3.4 LLL-reduced basis 17
   3.5 The LLL-algorithm 20
   3.6 Factoring polynomials 22

4 The NTRU cryptosystem 28
   4.1 Notation of NTRU 28
   4.2 Key Generation 30
   4.3 Encryption 30
   4.4 Decryption 31
   4.5 Lattice attack on NTRU 32
      4.5.1 Setup 32
      4.5.2 Alternative decryption 36
   4.6 Practical results 39

A Source Code 42
1 Prologue

Cryptography is a field of mathematics and computer science which became incredibly important in the last 20 years due to the vastly grown usage of digital communication as on the internet. Without it, very many actions of our everyday life would not be possible, or at least insecure which often means unusable. Especially public key cryptography provides a whole new concept of encryption of data with solutions to classic cryptographical problems such as the key distribution problem. Therefore I very much liked the idea to investigate a public key cryptosystem which is actually used in practice nowadays. Besides the very well known and widely used RSA cryptosystem there is at least one other important scheme, the NTRU cryptosystem.

In chapter 4 I will introduce the NTRU cryptoscheme and show how en- and decryption works. Afterwards I will present one possible attack on NTRU that was developed by Coppersmith and Shamir and makes usage of lattice space mathematics. Since the Shortest Vector Problem is the underlying hard problem of NTRU it is very natural to also investigate the problem of finding a shortest vector in a given lattice and algorithms trying to achieve exactly that.

The algorithm used in this thesis to retrieve a rather short vector, is the LLL algorithm which is described in chapter 3. It will guarantee that its output vector is not much longer than an actual shortest vector and that will surprisingly be enough for many practical applications. It will indeed turn out that its output will often be good enough for the attack that is presented at the end of chapter 4.

This thesis will be concluded by the presentation of the practical results. Therefore the theoretical attack of Coppersmith and Shamir was implemented in sage, an open source mathematical software. It will show that the attack does indeed work very often, at least when the key size is not very long.
2 Introduction

2.1 Modern Cryptography

If one is thinking about a suitable definition for cryptography, it is essential to distinguish between the classical era of cryptography which can be traced back all the way to the ancient Roman Empire and its modern successor which we call modern cryptography. Significant changes happened in the early 20th century when the invention of computer systems lead to completely new possibilities for en- and decryption of messages.

But there are not only technical differences that separate these two types of cryptography. Before the 20th century it relied almost completely on the creativity of involving parties. There existed nearly no scientific research on this topic and also no indicators to decide whether a given cryptosystem was easy to break or not. Due to this fact one describes cryptography before the 20th century as art.

Also by investigation of the purposes of cryptosystems we discover obvious differences. The classic use only focused on preserving confidentiality, meaning that it should prevent unauthorized persons from reading messages and thus getting information they should not access. In modern times there are many other uses of cryptography, like authentication, message integrity or non repudiation.

With the invention of the internet also the institutions who use cryptographic codes drastically changed. For hundreds of years cryptoschemes were used almost exclusively by military or intelligence organisations. Nowadays everyone who uses the internet, mobile phones or any other digital device for communication is, often unknowingly, entrusted to them. For example just think about websites with login for authorization or public wireless lan connections, where, without cryptography, everyone could eavesdrop or even change the whole communication flow. Due to these facts cryptography plays an important role in our everyday life which is also the reason why the research of it is becoming a more and more popular part in computer science.

This thesis will only deal with modern cryptography but it is nevertheless recommended taking a look at the classic approach as it helps to understand the development and also to get a brief idea what different kinds of schemes exist to encrypt messages.
2.2 Secret Key Encryption

The classical construction of ciphers, or nowadays called encryption schemes, did only focus on preventing unauthorized persons from eavesdropping the communication. In this setup receiver and sender of the message agreed beforehand on a mutual secret, called key, that both parties know. The key is used to encrypt the message by the sender and also by the recipient to decrypt it back into the original message. So both parties share the same secret, which explains the term private cryptography. This kind of encryption scheme is also called symmetric cryptography due to the fact that both, sender and recipient, use the same key for en- and decryption. The original message is called plaintext while its encrypted counterpart on the receiver’s end is named ciphertext.

In order to use a private key encryption scheme it is necessary to arrange a meeting of both parties to choose the secret key they both have to know about. Obviously it is not a good idea to transfer it over the same channel as the ciphertext, for an attacker could also eavesdrop the secret key and can then decrypt the ciphertext on its own. This does not seem to be a very hard restriction at first glance. Indeed, if we think about the mainly military use of cryptography more than one century ago, it was possible to arrange a meeting for agreeing on a mutual secret key. But most of modern users of cryptography want to establish ad-hoc communication with other parties all over the world and therefore physical appointments are just not realizable. Due to this fact, many modern encryption protocols do not only rely on private key encryption alone but combine it with other types.

Definition 2.1. A private key encryption scheme consists of three different algorithms. They have mainly the following purposes:

1. **Gen** is the key generation algorithm. It outputs a key $k$ that is chosen by some probability distribution which is defined by the scheme.

2. **Enc** is the encryption algorithm. Given its input, the private key $k$ and the plaintext $m$, it outputs a ciphertext $c$. The encryption of the plaintext $m$ with the key $k$ will be written as $Enc_k(m)$.

3. **Dec** is the decryption algorithm. Given its input, the private key $k$ and the ciphertext $c$, it outputs a plaintext $m$. The decryption of the ciphertext $c$ with the key $k$ will be written as $Dec_k(c)$.

The image of the key generation algorithm **Gen** is called the key space and will be denoted by $K$. In nearly every modern encryption scheme **Gen** simply randomly chooses
a key $k$ out of $\mathcal{K}$. All messages that are a valid input for the encryption algorithm form the plaintext space $\mathcal{M}$. If $\mathcal{K}$ and $\mathcal{M}$ are completely described, they define the ciphertext space $\mathcal{C}$ by computing every encryption of all plaintexts with every possible key. Since the rightful recipient of the message must be able to reconstruct the plaintext $m$, it must hold that for every $m \in \mathcal{M}$ we have

$$Dec_k(Enc_k(m)) = m.$$ 

A usual communication with private key encryption may look like the following: At first the parties must use $\text{Gen}$ to create a key $k$ that they both know. Then the sender takes the plaintext $m$ and uses $\text{Enc}$ to compute $c := Enc_k(m)$ and sends $c$ through the channel. After the message is transmitted the receiver takes the private key $k$ and $\text{Dec}$ for reconstructing the plaintext $m$ by computing $m := Dec_k(c)$.

It is immediately clear that any person who possesses the key $k$, knows $\text{Dec}$ and somehow manages to capture the ciphertext $c$ will be able to decrypt it and obtain the plaintext $m$. Hence it is essential to hide at least either the key $k$ or the decryption algorithm. In 19th century Aguste Kerckhoffs suggested the following which is nowadays known as Kerkhoffs principle:

A cipher must be secure even if everything except for the key is known to a potential attacker.

So Kerkhoffs only relies on the private key $k$ to ensure secure communication. There are several reasons to justify this opinion. At first it is obviously easier to keep a relatively short key hidden, compared to a whole encryption scheme which is often fairly complicated and much larger.

Secondly if security only relies on the key, it is easy to restore security once the key gets compromised. Just generate another key and use this one for further communication. If otherwise security lies within the hidden decryption algorithm one has to change the whole encryption scheme once it gets compromised. This is by far a harder task to do.

At last there is the problem of communication between more than two parties, for instance in a big company. It is much easier to use the same algorithm but different keys for all parties than using different algorithms.

Practice showed that it is actually a very good idea to publish a new invented cryptosystem. It will hopefully lead to various testing by experts which helps to reveal possible
weaknesses in the algorithms. It seems natural to trust an algorithm that is publicly known for a long period of time without any known attacks in comparison to an algorithm which is unknown to most people and hence could not be tested by any other person than the inventors.

At the end of this chapter we will take a brief look at common attack scenarios. There are four types of basic attacks:

- **Ciphertext-only attack**: This is the first and easiest attack that one can think of. The adversary just observes one or multiple ciphertexts and tries to recover their corresponding plaintexts.

- **Known-plaintext attack**: In this setup the adversary knows about several plaintext-ciphertext pairs encrypted with the same key. It will use this information to recover the plaintext of another ciphertext it observed.

- **Chosen-plaintext attack**: Here the adversary also possesses arbitrarily many plain- and ciphertext pairs. The difference to a Known-plaintext attack is that it can freely choose the plaintexts as it wishes. Again it will use this information to recover the plaintext of another ciphertext.

- **Chosen-ciphertext attack**: This is the final type of attack. The adversary can choose any ciphertext he wishes and will obtain its decryption. It then proceeds to recover the plaintext of another ciphertext it somehow could not manage to decrypt.

These attacks can be portioned in two different categories. The first one describes a passive adversary who will only eavesdrop messages sent through the channel of communication. This is a fairly easy way to attack a cryptosystem and every encryption scheme must be secure against any attack of this kind. The second attack is also a passive one but now the attacker may also know some pairs of plain- and ciphertext. This is reasonable to assume since history taught us that there are often some parts of messages that will eventually be discovered by an attacker. The other reason is that often messages are only secret for a given period of time. After it expired, the information will be publicly available and the adversary who stored the transmitted ciphertexts will know to which plaintext they corresponded. Hence every cryptosystem that is designed nowadays must at least be secure against these two types of attacks.

The last two types are active. While it might feel unnatural to assume apriori that an adversary can en- or decrypt arbitrary messages, these are attacks that occur in practice.
However, this does not mean that every cryptosystem must be secure against every of these four attack types. It is in fact essential to estimate both the value of the hidden information and the resources an attacker may use to discover it. There are practical situations where it is absolutely fine to be only secure against passive attacks in order to keep the cryptosystem simple and fast.

2.3 Problems of Secret Key

Secret Key Encryption was for a very long time the only encryption scheme that was used in practice. There is an understandable temptation to believe that, if it is possible to realize a secure communication over an insecure channel by means of cryptosystems based on secret keys, there is absolutely no need to think about other schemes that might take a completely different approach in order to establish a cryptographically secure communication. While it is true in theory that we can handle our whole communication only by methods of Secret Key Encryption there are several problems that occur in practical use.

The most obvious problem is the one of key distribution. Every party that takes part in a conversation over an insecure channel must have access to the same secret key in order to decrypt the messages properly. But how can they share a common secret key in the first place? It is obviously not a smart idea to just send it over the insecure channel because any adversary who eavesdrops the secret key will be able to listen to their communication and can decrypt all messages. For large companies or governments it might not be a large problem to agree on a physical meeting and send a trusted employee to exchange the shared secret key. But for personal use, let us say an ad-hoc conversation over the internet, this is absolutely impractical.

Another obstacle is the question of how many keys does one need and how should they be stored. Imagine a company of \(N\) employees where everyone should be able to communicate with everyone else. This would lead to \(\binom{N}{2}\) keys meaning each employee will have to store roughly \(N^2\) number of keys. Since attacks on personal computers, for example over the internet, occur more and more often it is not advisable to store all those keys directly on one’s own hard disk. Storing the keys on an external media on the other hand makes the whole process more insecure.

So as conclusion one must accept that the setting of Secret Key Encryption alone does not satisfy the requirements for modern communication where many ad-hoc conversations take place with parties of different countries which have no possibility to arrange a personal meeting beforehand. Due to these large problems people came up with at least one other
type of cryptosystem that is widely used nowadays.

2.4 Public Key Encryption

As seen above especially for modern types of communication, say over telephone or internet, cryptosystems which only rely on secret key encryption are not applicable. Public key encryption addressed this problem by designing a whole new type of cryptosystems. In the classic setting both parties shared the same secret key that was used for encryption and decryption at the same time. However, this is not the case in public cryptography. Here the key consists of two parts called the secret key $k_s$ and the public key $k_p$. Let us say Bob wants to send a message to Alice over an insecure channel. In the setup of public cryptography Alice then computes the key pair $(k_s, k_p)$ and transmits the public key $k_p$ over the channel. It is important to understand that it is absolutely no problem if this public key is overheard by a possible adversary. In fact Alice can publish her public key anywhere she wishes, say on her website or in the newspaper. She does not even have to know how many parties would like to communicate with her or when the communication will take place. It is only important that everyone who wishes to send secure messages to Alice has access to her public key $k_p$. In this case one would use $k_p$ to encrypt the message and send it to Alice. Upon receiving she would use her own private key, which she must not reveal to anyone else, to decrypt the message. Due to the fact that there is one key used for encryption and another one used for decryption, public key encryption is sometimes also called an asymmetric cryptosystem. Since $k_p$ is publicly available for anyone it is clear that the security of the conversation can only rely on $k_s$ which is only known by Alice. As both keys $k_s$ and $k_p$ clearly cannot be chosen independently from each other, it is not possible to choose the keys at random.

One might ask oneself what are the advantages of these cryptosystems because they are used so much in practice nowadays. So let us take a look at how the problems of secret key encryption mentioned above are addressed by public key systems.

- The issue of key distribution is solved in a very admirable way. There is absolutely no need for a physical meeting of party members because the public key can be distributed over a publicly accessible channel. In fact every part of the communication between the parties may be overheard by an adversary but their messages are still secure.

- The issue of key storage is also dealt with. Alice only needs to compute one key pair
(k_p, k_s) which will be used by everyone who wishes to communicate with her. There is no need to generate and store different keys for every different communication partner which reduces the number of keys to store drastically.

So did we find the perfect cryptosystem and there is no need anymore for private key cryptography? Unfortunately this is not true since the main disadvantage of public key cryptography is its slowness. It is significantly slower than comparable private key cryptosystems. This means that performing en- or decryption on small devices with small computational power like mobile phones or smart cards is not always an option. Even with high processing power at hand there might still be problems if there are many messages that need to be en- or decrypted, for example a server that processes credit card transactions. In fact it is even recommended to use secret key cryptosystems whenever it is possible. In practice public key encryption is often used in some combination with private key settings.

It is necessary to mention that so far we did only think about a passive adversary that has the possibility to record the whole conversation between Alice and Bob. However if we want to also consider an active attacker, new problems arise. Let us think about a classical man in the middle attack. Alice computes her key pair (k_s, k_p) and sends k_p to Bob. The adversary may then intercept the public key of Alice and exchange it with its own k_p' which it will send Bob. But Bob has no idea that this is not Alice’ key and will hence encrypt his message with k_p' and send it back to Alice. Again the adversary can intercept this message and use its own public key to decrypt it. Afterwards it will encrypt the message with Alice’ public key k_p and send it to Alice. Not only did it manage to read the message that Bob sent to Alice but it even did so completely unnoticed by Alice and Bob. This little example (which is in fact a practical one that really happened) shows that there are still problems with the key distribution even in public key cryptography that have to be considered.

The following definition will give us a precise formal understanding of public key cryptosystems which is very similar to Definition 2.1 but treats the fact that now there are two different kinds of keys.

**Definition 2.2.** A public key encryption scheme consists of three different algorithms. They have mainly the following purposes:

1. **Gen** is the key generation algorithm. It outputs a key pair (k_s, k_p) which consists of the secret key k_s and the public key k_p.
2. **Enc** is the encryption algorithm. Given its input, the public key $k_p$ and the plaintext $m$, it outputs a ciphertext $c$. The encryption of the plaintext $m$ with the key $k$ will be written as $Enc_{k_p}(m)$.

3. **Dec** is the decryption algorithm. Given its input, the secret key $k_s$ and the ciphertext $c$, it outputs a plaintext $m$. The decryption of the ciphertext $c$ with the key $k$ will be written as $Dec_{k_s}(c)$.

Considering possible attacks on a public key cryptosystem, one must deal with the fact that the public key $k_p$ is known to the attacker at any time. This has great effects because the attacker can generate every message it wishes and encrypt it with $k_p$. It can then try to determine any information about the original message or the secret key. This is exactly the setup of a chosen-plaintext attack. Of course it is necessary to demand that a public key scheme must be secure against any attack of this type leading to the fact that any public key encryption must be secure against a chosen-plaintext attack by default.

Apart from setting up a cryptographically secure communication, public key cryptosystems play a huge role in digital authentication. One problem arising from the increasing use of digital media for communication is verification, in that sense that it is very hard and without certain mechanics near to impossible to ensure that the sender of a message is really the person one assumes and not an impostor. Think about a simple email from Alice to Bob. How can Bob really be sure that Alice is the sender and no one else claims to be her? The answer is given by means of public key cryptography. The sender, Alice for instance, just encrypts, respectively digitally signs the message with her private key $k_s$ and sends the message. Bob and everyone else simply needs to take Alice' public key $k_p$ to decrypt the message. If decryption worked correctly Bob can be sure that only Alice could have sent the message, of course assuming that no other person has access to her private key. To ensure that no one else except for Alice can claim to be her, certification authorities were introduced in the process. This authority is a trusted third party, where trusted means that both parties, here Alice and Bob, have to trust it. With this setup Bob can verify Alice' public key and therefore be nearly completely sure that it is indeed her key and not the one of an impostor.

After having seen now both, secret and public key encryption, one natural question that will arise, is which one should be preferred over the other. As discussed earlier the answer to this question depends completely on the environment wherein the cryptosystem should operate. Nowadays many commonly used encryption schemes try to combine the easy key distribution of public key encryption with the speed advantage of secret key systems.
This is achieved by generating a secret key $k$ and encrypt it with a public key scheme. Afterwards it can be sent over the insecure channel leading to a shared secret $k$ of both communication parties. Then secret key encryption is applied to achieve an effective and fast way of communication.

In chapter 4 we will see an important example for a public key cryptosystem, namely the NTRU cryptosystem. Beforehand we will take a look at the mathematical background which is needed to understand NTRU and possible attacks on it.
3 The LLL Algorithm

3.1 Lattices

In this section we will introduce lattices in order to understand the Shortest Vector Problem. That is, for a given lattice \( L \) and a norm \( N \) on \( L \) find the shortest nonzero vector in \( L \) with respect to \( N \). Nowadays it is still unknown if there exists an algorithm which can compute a shortest vector in polynomial time. In this chapter we will focus on the approach of Lenstra, Lenstra and Lovász that will not necessarily output a shortest vector of the lattice but rather a fairly close one which turns out to be still quite useful in practice.

Usually lattices are introduced as discrete subspaces of an Euclidean vectorspace. Discrete means, that there is some positive constant \( \epsilon \) such that every two vectors in the lattice have a distance at least as big as \( \epsilon \). This leads to the following definition.

**Definition 3.1.** A subset \( L \) of a Euclidean vectorspace \( E \) is called lattice if and only if there are \( \mathbb{R} \)-linearly independent vectors \( b_1, b_2, \ldots, b_n \in E \) such that

\[
L = \sum_{i=1}^{n} \mathbb{Z} b_i = \left\{ \sum_{i=1}^{n} c_i b_i : c_i \in \mathbb{Z} \text{ for } i = 1, \ldots, n \right\}.
\]

The rank of \( L \) is \( n \). If the rank of \( L \) is equal to the dimension of \( E \) one says that the lattice is of full dimension.

In this case the matrix \( B \) which is formed out of the vectors \( b_1, b_2, \ldots, b_n \) is called a basis of \( L \).

It is also possible and in many cases useful to give another definition of lattices. This one does not need Euclidean vector spaces but a quadratic form \( q \).

**Definition 3.2.** Let \( K \) be a field of characteristic different from 2, and let \( V \) be a \( K \)-vector space. Then a quadratic form \( q \) is a map from \( V \) to \( K \) that satisfies both of the following conditions:

1. For every \( \lambda \in K \) and \( x \in V \) it is

\[
q(\lambda \cdot x) = \lambda^2 q(x).
\]

2. If we set \( b(x, y) = 1/2(q(x + y) - q(x) - q(y)) \) then \( b \) is a symmetric bilinear form which means \( b(x + x', y) = b(x, y) + b(x', y) \) and \( b(\lambda \cdot x, y) = \lambda b(x, y) \) for every \( \lambda \in K \), \( x, x' \) and \( y \) in \( V \).
With \( b(x, x) = q(x) \) we easily obtain \( q \) from \( b \).

If \( K = \mathbb{R} \) then \( q \) is positive definite if for all \( x \in V \) it holds that \( q(x) > 0 \). Together with \( q \) we can define the lattice:

**Definition 3.3.** A lattice \( L \) is a free \( \mathbb{Z} \)-module with a finite rank combined with a positive definite quadratic form \( q \).

If we choose a \( \mathbb{Z} \)-basis of \( L \) and call it \((b_i)_{1 \leq i \leq n}\), we get for \( x = \sum_{1 \leq i \leq n} x_i b_i \in L \) with \( x_i \in \mathbb{Z} \) that our quadratic function \( q \) is described by

\[
q(x) = \sum_{1 \leq i \leq n} q_{i,j} x_i x_j \quad \text{with} \quad q_{i,j} = b(b_i, b_j).
\]

As usual \( b \) is the symmetric bilinear form we associate with \( q \).

Now we want to take a look at the matrix \( Q = (q_{i,j})_{1 \leq i,j \leq n} \). It is easily seen that \( Q \) is a symmetric matrix and is also positive definite if \( q \) is positive definite. With the help of \( Q \) we can denote our inner product \( b \) by \( b(x, y) = Y^t Q X \) which means especially that \( q(x) = X^t Q X \) with \( X \) and \( Y \) are the column vectors of \( x \) and \( y \) respectively in the basis \((b_i)\).

There are several possibilities to describe a lattice \( L \) but in this chapter we will always use a \( \mathbb{Z} \)-basis \((b_i)_{1 \leq i \leq n}\) of the lattice to describe it. In that case we know that an element \( x \in L \) is considered as a column vector \( X \) with the integral coordinates of \( x \) with respect to the basis. Also the quadratic form \( q \) can then be described by the matrix \( Q \) as seen above.

If we change the \( \mathbb{Z} \)-basis of \( L \) we have to replace \( X \) by \( PX \) for some \( P \in GL_n(\mathbb{Z}) \) leading to the equation \( q(x) = (PX)^t Q (PX) = X'Q'X \) with \( Q' = P^t Q P \). Since \( P \) is in \( GL_n(\mathbb{Z}) \) it is \( \det(P) = \pm 1 \) and therefore \( \det(Q') = \det(Q) \). So the determinant of \( Q \) is invariant under the change of basis and we will define the determinant of the lattice \( L \) as \( d(L) = \det(Q)^{1/2} \). Recall that \( Q \) is positive definite so \( \det(Q) > 0 \).

Now let us take a look at two nice examples of lattices. Consider an algebraic number field \( K \), which means that it as a finite field extension of the field \( \mathbb{Q} \) of the rational numbers. Then let \( L \) be a subgroup of the additive group of \( K \) which is finitely generated. This may be done by defining \( L \) as the ring \( \mathbb{Z}_K \) of algebraic integers in \( K \) for example. To see the lattice structure of \( L \) we need to define the quadratic form \( q \) which is done by

\[
q(x) = \sum_{\sigma} |\sigma x|^2,
\]
where \( x \in L \) and \( \sigma \) is an element out of the set of field embeddings of \( K \) into the field \( \mathbb{C} \) of the complex numbers.

Elliptic curves will be the second example for lattices. Let \( E \) be an elliptic curve over \( \mathbb{Q} \) given by the Weierstrass equation \( y^2z + a_1xyz + a_3yz^2 = x^3 + a_2x^2z + a_4xz^2 + a_6z^3 \), where all \( a_i \in \mathbb{Q} \). This equation describes the elliptic curve \( E(\mathbb{Q}) \) as a set of points \((x : y : z)\) in the projective plane \( P^2(\mathbb{Q}) \). They form an abelian group, called the Mordell-Weil group of \( E \) over \( \mathbb{Q} \). Its subgroup of elements of finite order is given by \( E(\mathbb{Q})_{\text{tor}} \). Then we get the lattice \( L = E(\mathbb{Q})/E(\mathbb{Q})_{\text{tor}} \) with quadratic form

\[
q(\bar{P}) = \frac{1}{2} \cdot \lim_{n \to \infty} \frac{h(2^nP)}{4^n},
\]

where \( P \in E(\mathbb{Q}) \) and \( \bar{P} \) is the image of \( P \) in \( L \) and the mapping \( h \) is given by \( h(x : y : z) = \log \max \{|x|, |y|, |z|\} \). In this case \( q(\bar{P}) \) is also known as the canonical height of \( P \).

3.2 The Shortest Vector Problem

The Shortest Vector Problem (SVP) is, along with the closely related Closest Vector Problem (CVP), one of the most interesting lattice based problems for current research. Apart from the practical issues that are related to the quick solvability of SVP, this is probably also due to its simplicity. It is easy to formulate and most likely one of the first questions that come into mind when investigating lattices and yet up to now there exists no algorithm that can find a shortest vector in reasonable time, at least for high dimensions. In this chapter we take a look why SVP is so hard to solve. At first we formally give the definition of it.

Definition 3.4 (Shortest Vector Problem). Let \( L \) be a lattice of dimension \( n \) and \( B \) a basis for \( L \). Then the Shortest Vector Problem is finding a vector \( y \in L \) that has minimum nonzero norm, i.e. a solution to

\[
y = \arg\min_{x \in \mathbb{Z}^n, \ x \neq 0} \|Bx\|,
\]

where usually \( \| \cdot \| \) is any \( p \)-norm and usually will be the euclidean norm.

For the sake of completeness and since both problems are related to each other it is also important to know the Closest Vector Problem.
Definition 3.5 (Closest Vector Problem). Let $L$ be a lattice of dimension $n$ with basis $B$ and $x$ be a vector that does not lie in $L$. Then the Closest Vector Problem is finding a vector $y \in L$ that has minimum distance to $x$, i.e. a solution to

$$y = \arg\min_{x \in \mathbb{Z}^n} ||Bx - y||,$$

where again $|| \cdot ||$ can be any $p$-norm.

Although it is very hard to actually determine a shortest vector, the theorem of Minkowski at least gives an upper bound on its norm.

Theorem 3.6 (Minkowski’s Theorem). Let $L$ be a lattice of positive rank $n$. Then it contains a nonzero element $x$ with $q(x) \leq \frac{4}{\pi} \cdot (\frac{n!}{2})^{2/n} \cdot d(L)^{2/n} \leq n \cdot d(L)^{2/n}$. 

Proof. Let $L \in \mathbb{R}^n$ and set $\lambda = \lambda(L) = \min \{q(x) : x \in L, x \neq 0\}$. As usual it is $||x|| = q(x)^{1/2}$. Then the distance of any two lattice elements must be at least $\sqrt{\lambda}$. Now define the open ball $B' = \{z \in \mathbb{R}^n : <z,z> < \lambda/4\}$, then the radius of $B'$ is exactly $\sqrt{\lambda}/2$. If $y + B'$ is the open ball $B'$ centered at the lattice element $y$, these balls are pairwise disjoint by the setting of $B'$. Let $b_1, b_2, \ldots, b_n$ form a basis of $L$ and let $F$ be the set

$$F = \sum_{j=1}^{n} [0,1) b_j = \left\{ \sum_{j=1}^{n} c_j b_j : c_j \in \mathbb{R}, 0 \leq c_j < 1 \text{ for } 1 \leq j \leq n \right\}.$$ 

$F$ is often called a fundamental domain for $L$ since each point $x \in \mathbb{R}^n$ has a unique representation $x = y + z$ with $y \in L$ and $z \in F$. It is easy to see that $d(L) = \det B = \text{vol } F$. More importantly this means that the sets $y + F$ disjointly cover the whole $\mathbb{R}^n$ as $y$ ranges over $L$. Therefore it holds that $\text{vol } B' \leq F = d(L)$. Let $B(1)$ be the ball with radius 1 around the origin. Then we get

$$\text{vol } B' = (\sqrt{\lambda}/2)^n \cdot \text{vol } B(1) = (\sqrt{\lambda}/2)^n \cdot \frac{\pi^{n/2}}{n!},$$

and $\frac{n!}{2}$ is inductively defined by $0! = 1$, $\frac{1!}{2} = \sqrt{\pi}/2$, and $\frac{n!}{2} = \frac{n}{2} \cdot \frac{n-2}{2} \cdot \ldots \cdot \frac{4}{2} \cdot \frac{2}{2}$ for $n \geq 2$. Therefore, after solving the inequality for $\lambda$, we obtain the first inequality in Minkowski’s theorem. For the second one observe, that $B(1)$ contains a cube with edge length $2/\sqrt{n}$ if $n \geq 2$. This yields $\text{vol } B(1) \geq (2/\sqrt{n})^n$ and we obtain

$$\frac{4}{\pi} \cdot (\frac{n}{2})^{2/n} \cdot d(L)^{2/n} = 4 \cdot \frac{1}{\text{vol } B(1)^{2/n}} \cdot d(L)^{2/n} \leq 4 \cdot \frac{1}{(2/\sqrt{n})^{2n/n}} \cdot d(L)^{2/n} = n \cdot d(L)^{2/n}.$$
which proves the second inequality in Minkowski’s theorem.

The Minkowski theorem is really important for the search of shortest vectors in lattices as it guarantees to find a vector with length bounded from above by a bound consisting only of the lattice determinant and the lattice dimension. However, the great problem with Minkowski’s theorem is, that it is non-constructive. The bound is given by a nice theoretical investigation but it is nowhere said how to obtain a vector satisfying that bound. Still up to today there exists no real better solution than just some sort of enumeration of lattice vectors which of course performs very badly in terms of time consumption. Hence one has to be satisfied with approximations to the shortest vector within some factor, what is exactly what the LLL algorithm does.

3.3 Gram-Schmidt Process

To obtain an orthonormal basis of a vector space $V$ the most common approach is to use the Gram-Schmidt procedure. In the first step it transforms the vectors of the basis in order to make them pairwise orthogonal with respect to the inner product $b$. At last every vector is divided by its norm to ensure a length of 1. We will however only need orthogonality and therefore only use the first part of Gram-Schmidt without dividing the vectors by their norms. In the following we will write $<b_i, b_j>$ for the inner product $b(b_i, b_j)$.

**Theorem 3.7.** (Gram-Schmidt). Let $b_i$ be a basis of an Euclidean vector space $E$. Then define by induction:

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^* \quad (1 \leq i \leq n),$$

with

$$\mu_{i,j} = \frac{<b_i, b_j^*>}{<b_j^*, b_j^*>} \quad (1 \leq i \leq n),$$

then the $b_i^*$ form an orthogonal basis of $E$ and if $d(L)$ is the determinant of the lattice it is

$$d(L)^2 = \prod_{1 \leq i \leq n} ||b_i^*||^2.$$

The proof is done by induction in nearly any introductory book for linear algebra, for example in [CK], but is so straightforward that we will not present it here. We can already see why it is not possible to use the exact formulation of Gram-Schmidt on lattices to obtain an orthogonal basis since the $\mu_{i,j}$ are fractions and therefore it is not clear (and in general not true) that every vector of the $b_i^*$ is in fact again a vector of the lattice.
The following lemma is an easy conclusion of the Gram-Schmidt procedure.

**Lemma 3.8.** (Hadamard’s Inequality). Let \((L, q)\) be a lattice with determinant \(d(L)\) and a \(\mathbb{Z}\)-basis \((b_i)_{1 \leq i \leq n}\). Let \((b_i^*)_{1 \leq i \leq n}\) be its Gram-Schmidt Orthogonalization as above. For \(x \in L\) it is \(|x| = q(x)^{1/2}\). Then

\[
d(L) \leq \prod_{i=1}^{n} |b_i|.
\]

**Proof.** By the orthogonality of \(b_i^*\) we get

\[
q(b_i) = |b_i|^2 = |b_i^*|^2 + \sum_{1 \leq j < i} \mu_{i,j} |b_j^*|^2,
\]

hence

\[
d(L)^2 = \prod_{1 \leq i \leq n} |b_i^*|^2 \leq \prod_{1 \leq i \leq n} |b_i|^2.
\]

The first equality holds due to the Gram-Schmidt procedure. \(\square\)

This will be sufficient as a setup to understand the LLL algorithm.

### 3.4 LLL-reduced basis

In the previous section we already saw that lattices are often described by explicitly declaring a \(\mathbb{Z}\)-basis of it. Since there is not one unique possibility to choose such a basis it is apriori not clear which one is to prefer. For many practical issues it turns out that a basis with short elements is desirable. Due to the fact that the determinant of a lattice does not change under basis transformation and because it can also be seen as the volume of the parallelepiped spanned by the basis vectors, a short basis will be fairly orthogonal. Such a basis is also called reduced.

As mentioned earlier there does still not exist an algorithm that returns a shortest vector for a given lattice in reasonable time. This is why the algorithm of Lenstra, Lenstra and Lovász was considered a real breakthrough when it was introduced in 1982. However, it does not solve the original mathematical problem of finding a shortest vector in a lattice because it will only guarantee as output a fairly short vector with respect to a certain bound. It is still very useful in practice as one often does only need a relatively short vector but not necessarily a shortest one.

We will now see the important property of a basis to be LLL-reduced.

**Definition 3.9.** Let \(b_1, b_2, \ldots, b_n\) be a basis of the lattice \(L\). By using Gram-Schmidt orthogonalization as described above we find an orthogonal basis \(b_1^*, b_2^*, \ldots, b_n^*\).
Then we say that the basis $b_1, b_2, \ldots, b_n$ is LLL-reduced if

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

and

$$|b_i^* + \mu_{i,i-1}b_{i-1}^*|^2 \geq \frac{3}{4}|b_{i-1}^*|^2 \quad \text{for } 1 < i \leq n,$$

which is, as $b_i^*$ and $b_{i-1}^*$ are orthogonal, equivalent to

$$|b_i^*|^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right)|b_{i-1}^*|^2.$$

The next theorem will show us why this is such an important definition.

**Theorem 3.10.** Let $b_1, b_2, \ldots, b_n$ be an LLL-reduced basis of a lattice $L$. Then

1. $$d(L) \leq \prod_{i=1}^n |b_i| \leq 2^{n(n-1)/4}d(L),$$
2. $$|b_j| \leq 2^{(i-1)/2}|b_i^*|, \quad \text{if } 1 \leq j \leq i \leq n,$$
3. $$|b_1| \leq 2^{(n-1)/4}d(L)^{1/n},$$
4. For every $x \in L$ and $x \neq 0$ it holds that
   $$|b_1| \leq 2^{(n-1)/2} |x|.$$

Clearly, this leaves $b_1$ to be the candidate of a fairly short vector in $L$ with the condition that it cannot be too far away from an actual shortest vector. In fact in practice it will often be a shortest vector in $L$.

**Proof.** Let $B_i = |b_i^*|^2$ for this proof. The first inequality of (1) is already given as mentioned in the Gram-Schmidt procedure above. Since we know that the $b_i$ are LLL-reduced it holds that $B_i \geq (3/4 - \mu_{i,i-1}^2)B_{i-1} \geq B_{i-1}/2$ as $|\mu_{i,i-1}| \leq 1/2$. Using induction this directly leads
to $B_j \leq 2^{i-j}B_i$ for $i \geq j$. Then we get

$$|b_i|^2 = B_i + \sum_{1 \leq j < i} \mu_{i,j}^2 B_j \leq B_i + B_i \left( \frac{2^i - 2}{4} \right) = B_i \cdot \frac{2^{i-1} + 1}{2},$$

which enables us to write

$$\prod_{i=1}^n |b_i|^2 \leq \prod_{i=1}^n \frac{2^{i-1} + 1}{2} B_i \leq 2^{n(n-1)/2} d(L)^2.$$

This directly proves the second inequality and thus finishes the proof of (1). For (2) we can use the inequalities we just showed and combine them to obtain

$$|b_j|^2 \leq \frac{2^{j-1} + 1}{2} B_j \leq \frac{2^{j-1} + 1}{2} 2^{i-j} B_i \leq (2^{i-2} + 2^{i-j-1}) B_i \quad \text{for all } j \leq i,$$

and thus proved (2).

Now let us set $j = 1$ in (2) and take the product for $i = 1$ to $i = n$, i.e.

$$\prod_{i=1}^n |b_1|^2 \leq 2^{n(n-1)/2} \prod_{i=1}^n B_i = 2^{n(n-1)/2} d(L)^2,$$

and by that obtain (3). The last claim, (4), is probably the most important one because it ensures a good quality for $b_1$. For the proof note that there exists an $i$ such that $x = \sum_{1 \leq j < i} r_j b_j = \sum_{1 \leq j < i} s_j b_j$ where $r_i \neq 0$, $r_j \in \mathbb{Z}$ and $s_j \in \mathbb{R}$. By the definition of $b_j^*$ it follows that $r_i = s_i$ and therefore

$$|x|^2 \geq s_i^2 B_i = r_i^2 B_i \geq B_i.$$

We then use again (2) to get

$$B_i \geq 2^{1-i} |b_1|^2 \geq 2^{1-n} |b_1|^2,$$

which proves (4).

The important point of the observations we have seen above is, that there exists an algorithm that can find a LLL reduced basis which is very simple and, more importantly, very efficient in terms of time consumption. The basic ideas of it work as follows.
3.5 The LLL-algorithm

Let us assume that the vectors $b_1, \ldots, b_{k-1}$ are already LLL-reduced. Obviously this will initially be the case for $k = 2$. It must now be ensured that for the vector $b_k$ the condition $|\mu_{k,j}| \leq 1/2$ is satisfied for all $j < k$. In order to achieve this goal, replace $b_k$ by $b_k - \sum_{j<k} a_j b_j$ for some $a_j \in \mathbb{Z}$ which satisfy the following. If $|\mu_{k,j}| \leq 1/2$ for $l < j < k$, which is initially true for $l = k$, then define $q := \lfloor \mu_{k,l} \rfloor$ as the nearest integer to $\mu_{k,l}$. If we use this $q$ to replace $b_k$ by $b_k - qb_l$, we do not modify any $\mu_{k,j}$ for $j > l$ since $b^*_l$ is orthogonal to $b_l$ for $l < j$ and for $\mu_{k,l}$ we get

$$|\mu_{k,l}| = \frac{\langle b_k - qb_l, b^*_l \rangle}{\langle b^*_l, b^*_l \rangle} = \frac{\langle b_k, b^*_l \rangle - q \langle b_l, b^*_l \rangle}{\langle b^*_l, b^*_l \rangle} = \frac{\langle b_k, b^*_l \rangle}{\langle b^*_l, b^*_l \rangle} - q \leq \frac{1}{2},$$

where the last equality holds because $\langle b_l, b^*_l \rangle = \langle b^*_l, b^*_l \rangle$ and the last inequality is true because of the definition of $q$.

After managing to ensure that the so called size reduction works as desired, we turn our attention to the Lovász condition $B_k > (3/4 - \mu^2_{k,k-1})B_{k-1}$. If it already holds at the beginning we just need to increase $k$ by one and continue with the next vector $b_k$. However, if it does not hold, the algorithm just swaps the vectors $b_k$ and $b_{k-1}$. After this step $k$ must be decreased by one as we can only ensure a LLL-reduced basis for the vectors $b_1, \ldots, b_{k-2}$ anymore.

It is clear that, should the algorithm ever terminate, the output indeed must satisfy the Lovász condition. What is much more difficult to see is that it really always terminates after a finite number of steps. Before proving this statement we will take a look at the description of the LLL Algorithm.

**Definition 3.11.** (The LLL Algorithm).

Given a basis $b_1, b_2, \ldots, b_n$ of a lattice $(L, q)$ the algorithm will transform the basis vectors $b_i$ in such a way that when the algorithm terminates the $b_i$ are LLL-reduced.

1. Set $k \leftarrow 2$, $k_{\text{max}} \leftarrow 1$, $b^*_1 \leftarrow b_1$ and $B_1 \leftarrow \langle b_1, b_1 \rangle$.

2. If $k \leq k_{\text{max}}$ go to step 3. Otherwise, set $k_{\text{max}} \leftarrow k$, $b^*_k \leftarrow b_k$, then for $j = 1, \ldots, k-1$ set $\mu_{k,j} \leftarrow \langle b_k, b^*_j \rangle / B_j$ and $b^*_k \leftarrow b^*_k - \mu_{k,j} b^*_j$. Finally, set $B_k \leftarrow \langle b^*_k, b^*_k \rangle$. If $B_k = 0$ output an error message as the $b_i$ did not form a basis in this case.

3. Execute Sub-algorithm RED($k, k-1$) below. If $B_k < (0.75 - \mu^2_{k,k-1})B_{k-1}$ execute SWAP($k$) below and set $k \leftarrow \max(2, k-1)$ and go to step 3. Otherwise, for $l =
$k - 2, k - 3, \ldots, 1$ execute Sub-algorithm RED($k, l$), then set $k \leftarrow k + 1$.

4. If $k \leq n$, then go to step 2. Otherwise, output the $b_i$ which now form a LLL-reduced basis and terminate the algorithm.

**Definition 3.12.** (Sub-algorithm RED($k, l$)) If $|\mu_{k,l}| \leq 0.5$ terminate the sub-algorithm. Otherwise, set $q$ to the nearest integer of $\mu_{k,l}$, i.e.

$$q \leftarrow \left\lfloor \mu_{k,l} \right\rfloor = [0.5 + \mu_{k,l}].$$

Set $b_k \leftarrow b_k - q b_l$, $\mu_{k,l} \leftarrow \mu_{k,l} - q$, and for all $i$ such that $1 \leq i \leq l - 1$, set $\mu_i \leftarrow \mu_i - q \mu_i$, and terminate the sub-algorithm.

**Definition 3.13.** (Sub-algorithm SWAP($k$)) Exchange the vectors $b_k$ and $b_{k-1}$ and if $k > 2$, for all $j$ such that $1 \leq j \leq k - 2$ exchange $\mu_{k,j}$ with $\mu_{k-1,j}$. Then set $\mu \leftarrow \mu_{k,k-1}$, $B \leftarrow B_k + \mu^2 B_{k-1}$, $\mu_{k,k-1} \leftarrow \mu B_{k-1}/B$, $B_k \leftarrow B_{k-1}B_k/B$ and $B_{k-1} \leftarrow B$. Finally for $i = k + 1, k + 2, \ldots, k_{\text{max}}$ set $t \leftarrow \mu_{i,k}$, $\mu_{i,k} \leftarrow \mu_{i,k-1} - t$, $\mu_{i,k-1} \leftarrow t + \mu_{k,k-1} \mu_{i,k}$ and terminate the sub-algorithm.

**Proof.** It is easily seen that at the beginning of step 4 the LLL conditions hold for $i \leq k - 1$. So if $k > n$ the algorithm will indeed output a LLL-reduced basis due to the observations we saw beforehand. What remains to show is the nontrivial claim that the algorithm will always terminate. This will be done by defining a potential function which maps a lattice basis to some positive integer number. If we can show that this potential function has a finite starting value, is decreased by at least some value $\delta$ in every swap step, bounded from below and not affected by the reduction step, it is clear that the algorithm must terminate eventually. So here is

**Definition 3.14.** Let $B = b_1, b_2, \ldots, b_n$ be a lattice basis. Its potential is defined as

$$D_B := \prod_{i=1}^{n} ||b_i^*||^{n-i+1} = \prod_{i=1}^{n} ||b_1^*|| ||b_2^*|| \ldots ||b_i^*|| = \prod_{i=1}^{n} D_B,i,$$

with $D_{B,i} := \det L_i$ where $L_i$ is the lattice spanned by $b_1, \ldots, b_i$.

By replacing every $||b_i^*||$ with $\max_i ||b_i^*||$ in the formula above and by using $||b_i^*|| \leq ||b_i||$ the value of $D_B$ is bounded from above by $(\max_i ||b_i||)^{n(n+1)/2}$.

Since the Gram-Schmidt basis does not change in the reduction step it will not affect $D_B$ at all. So let us consider the swapping step in the LLL-algorithm. Let $k$ be the index
such that the swapping step exchanges \( b_k \) and \( b_{k-1} \). It is clear that for all \( i \neq k - 1 \) the lattice \( L_i \) is not changed by this step since either no or both vectors \( b_k \) and \( b_{k-1} \) are contained in \( L_i \). So only \( D_{B,k-1} \) changes. Let \( L'_{k-1} \) and \( D'_{B,k-1} \) be the newly obtained values of \( L_{k-1} \) and \( D_{B,k-1} \). Then we get that

\[
\frac{D'_{B,k-1}}{D_{B,k-1}} = \frac{\det L'_{k-1}}{\det L_{k-1}} = \frac{\det L(b_1, b_2, \ldots, b_{k-2}, b_k)}{\det L(b_1, b_2, \ldots, b_{k-1})} = \frac{(\prod_{j=1}^{k-2} ||b_j^*||)||\mu_{k,k-1}b_{k-1}^* + b_k^*||}{\prod_{j=1}^{k-1} ||b_j^*||} = \frac{||\mu_{k,k-1}b_{k-1}^* + b_k^*||}{||b_{k-1}||} < \sqrt{0.75},
\]

where the last inequality is true because the swapping step will only be executed if the condition \( B_k < (0.75 - \mu_{k,k-1}^2)B_{k-1} \) is satisfied. This means that we know that \( D_B \) decreases in every swapping step at least by \( \sqrt{0.75} \) and since we know that \( D_B \) is a nonzero integer which is at least 1 we can conclude that the number of swapping steps must be finite and the algorithm will terminate in polynomial time.

This proof will conclude our investigations of the LLL-algorithm since we do not need more details to understand how it can be used to attack the NTRU cryptosystem which is introduced in the next chapter. It is of course important to note that the LLL-algorithm works much faster than every other known algorithm that can compute a shortest vector in a lattice. But as nearly always in life one has to pay for this advantage with the fact that it will only guarantee that the output vector is somehow close to a shortest vector but is not necessarily one itself. However in practice the output will often still be a shortest vector and even if not, it is possible to use it for further calculations.

### 3.6 Factoring polynomials

In this section we will see one very nice possibility to use lattices and also the LLL algorithm to solve long known mathematical problems. The problem dealt with in this section is to factorize nonzero polynomials in \( \mathbb{Q}[X] \) into irreducible factors in polynomial time.
At first let us take a look at Berlekamp’s algorithm which factorizes polynomials over finite fields. More precisely with the input of a prime number $p$ and a nonzero polynomial $f \in \mathbb{F}_p[X]$ its output is a complete factorization of $f$ into its irreducible factors in $\mathbb{F}_p[X]$. The run time of the algorithm is $O(p \cdot (\log p + \deg f)^c)$ with a positive constant $c$.

At the beginning we assume that $f$ is a nonzero monic polynomial with positive degree and additionally square free. The last assumption can be made without loss of generality since a polynomial that is not square free satisfies $\gcd(f(x), f'(x)) \neq 1$, where $\gcd$ is the greatest common divisor and $f'(x)$ the derivation of $f(x)$. Hence one factor of $f(x)$ is easy to find since it has a common factor with $f'(x)$.

Thus we can assume that $f = \prod_i f_i$ for pairwise distinct monic irreducible polynomials $f_1, f_2, \ldots, f_t \in \mathbb{F}_p[X]$. Consider the ring isomorphism

$$\mathbb{F}_p[X]/(f) \cong \prod_{i=1}^t \mathbb{F}_p[X]/(f_i),$$

where each $\mathbb{F}_p[X]/(f_i)$ is a field containing the subring given by the Frobenius endomorphism, i.e. $\{y \in \mathbb{F}_p[X]/(f_i) : y^p = y\}$ which is equal to the prime field $\mathbb{F}_p$. Therefore one gets $\{y \in \mathbb{F}_p[X]/(f) : y^p = y\} \cong \prod_{i=1}^t \mathbb{F}_p$. This means that $f$ is irreducible if and only if $\{y \in \mathbb{F}_p[X]/(f) : y^p = y\}$ is a one dimensional vector space over $\mathbb{F}_p$. Also if and only if $h$ is a irreducible factor of $f$, we obtain that all $\{y \in \mathbb{F}_p[X]/(f)\}$ with $y^p = y$ reduce to a constant modulo $h$.

This is exactly the property Berlekamp used to design his algorithm. At first it describes the $\mathbb{F}_p$-vector space $\{y \in \mathbb{F}_p[X]/(f) : y^p = y\}$ by finding a basis $g_1, g_2, \ldots, g_t$ of it. This is done by a neat little trick since it is exactly the null-space of the linear map $\mathbb{F}_p[X]/(f) \to \mathbb{F}_p[x]/(f)$ that maps every $y$ to $y^p - y$. This helps a lot because a null-space of a linear map is fairly easy to compute by means of linear algebra.

Now pick $a_1, a_2, \ldots, a_t \in \mathbb{F}_p$ at random in order to obtain the random element $a = \sum_{i=1}^t a_i g_i$. If one was extremely lucky, one already has that $\gcd(f, a) \neq 1$ and therefore found a nontrivial factor of $f$. Since this is very unlikely let us assume that $a$ is not already a divisor of $f$. If there is at least one entry of $a$, but not all, that is equal to minus one, we can consider the polynomial $a(x) + 1$ with entries equal to these of $a$ but added one to every entry. Since at least one entry of $a$ was minus one, one entry of $a + 1$ has to be zero and as we excluded the trivial case where all entries of $a + 1$ are zero we found a non-trivial factor of $f$.

The next question would be, how to make sure that at least one entry of the ran-
domly chosen polynomial \( a \) is minus one. In order to achieve this goal we just map \( a \) to \( a^{(p-1)/2} = (a_1^{(p-1)/2}, a_2^{(p-1)/2}, \ldots, a_t^{(p-1)/2}) \). As every \( a_i \) is an element of \( \mathbb{F}_p \) the result is a vector consisting of only ones and minus ones, which is exactly what we needed.

Now start the algorithm with \( h = f \). As explained above, if all \( g_i \) are congruent to a constant modulo \( h \), \( h \) is irreducible and the algorithm stops. Otherwise, choose a random element \( a \) as described above. If \( \gcd(a^{(p-1)/2} + 1, h) \) is non-trivial we found a factor of \( f \), which happens with probability at least \( 1 - 2^t \). If we were unlucky and did not find a non-trivial factor, just repeat the steps by choosing another random element \( a \).

So one factor of \( f \) is found. By setting \( h = \gcd(a^{(p-1)/2} + 1, h) \) and repeating the steps we either obtain that \( h \) is now irreducible or get another factor of it. Obviously this will go on until we get an irreducible factor of \( f \). To get others we just need to divide \( f \) by this factor and run the algorithm again on this newly obtained polynomial.

In the next step in order to design a factorizing algorithm we take a look at integer polynomials and the corresponding lattices. Let \( e = \sum_i a_i X^i \in \mathbb{Z}[X] \) be a polynomial with integer coefficients and let \( q(e) = \sum_i a_i^2 \) define a norm for those polynomials by setting ||\( e \)|| = \( q(e)^{1/2} \). If \( \mathbb{Z}[X]_n \) is the set of polynomials in \( \mathbb{Z}[X] \) with degree smaller than \( n \), it is easy to check that \( \mathbb{Z}[X]_n \) is in fact, together with \( q \), a lattice of rank \( n \) and determinant \( 1 \). The next lemma will guarantee us that two polynomials with certain properties must have a common factor. This is important for the factorizing algorithm we want to describe.

**Lemma 3.15.** Let \( m \) be a positive integer and let \( h \in \mathbb{Z}[X] \) be a monic polynomial. Additionally let \( f, g \) be nonzero elements of the \( \mathbb{Z}[X] \)-ideal \( (m, h) \). If we have

\[
||f||^{\deg f} \cdot ||g||^{\deg g} < m^{\deg h}, \quad \deg f + \deg g \geq \deg h,
\]

\( f \) and \( g \) have a common factor of positive degree in \( \mathbb{Z}[X] \).

**Proof.** Observe that if we find polynomials \( \lambda \in \mathbb{Z}[X]_{\deg g}, \mu \in \mathbb{Z}[X]_{\deg f} \) with \( \lambda f + \mu g = 0 \) and \( \lambda, \mu \) nonzero, it is implied that \( f \) and \( g \) must have a common factor with positive degree in \( \mathbb{Z}[X] \). Since this proof is done by contradiction assume that the only \( \lambda, \mu \) that satisfy this property are given by \( \lambda = \mu = 0 \).

In this case we can define

\[
M = \{ \lambda f + \mu g : \lambda \in \mathbb{Z}[X]_{\deg g}, \mu \in \mathbb{Z}[X]_{\deg f} \}.
\]

Since the only solution to our equation above is \( \lambda = \mu = 0 \), \( M \) is a sublattice of
$\mathbb{Z}[X]_{\deg f + \deg g}$ because its basis is given by

$$f, Xf, \ldots, X^{\deg g - 1} f, g, Xg, \ldots, X^{\deg f - 1} g.$$  

This also implies that the rank of $M$ is $\deg f + \deg g$. We can easily use Hadamard’s inequality to get $d(M) \leq \|f\|^{\deg g} \cdot \|g\|^{\deg f}$. As we know that $f$ and $g$ are contained in the ideal $(m, h)$ we can deduce that $M$ is contained in $L = (m, h) \cap \mathbb{Z}[X]_{\deg f + \deg g}$, which is also a sublattice of $\mathbb{Z}[X]_{\deg f + \deg g}$. Now we use that $\deg f + \deg g \geq \deg h$ to see that $(m, h) + \mathbb{Z}[X]_{\deg f + \deg g}$ is in fact equal to $\mathbb{Z}[X]$. This leads us to the equation

$$d(L) = \#\mathbb{Z}[X]_{\deg f + \deg g}/L = \#\mathbb{Z}[X]/(m, h) = m^{\deg h},$$  

using that $L$ is a sublattice of $\mathbb{Z}[X]_{\deg f + \deg g}$. Together with Hadamard’s inequality we get

$$\|f\|^{\deg g} \cdot \|g\|^{\deg f} \geq d(M) = (L : M) \cdot d(L) \geq m^{\deg h},$$  

using again that $M$ is a sublattice of $L$. Obviously this is a contradiction to our assumptions. Hence we must drop our hypothesis that the only solution to $\lambda f + \mu g = 0$ is $\lambda = \mu = 0$. But then we have nonzero polynomials $\lambda$ and $\mu$ with $\lambda f = -\mu g$ which guarantees a common factor of $f$ and $g$ in $\mathbb{Z}[X]$.

Finally, we are ready to define the algorithm for factorization of polynomials in $\mathbb{Q}[X]$. Throughout the description of the algorithm we assume that the discriminant $\Delta(f)$ of $f$ is nonzero and that the coefficients of $f$ are in $\mathbb{Z}$. This can be achieved easily by replacing $f$ by $f \mod \gcd(f, f')$ and by multiplying the coefficients of $f$ by a common denominator. Let also $n = \deg f$.

Step (1): Find the least prime number $p$ that does not divide the resultant $R(f, f')$. Up to the sign, this can be done by computing the product of the leading coefficient $a_n$ of $f$ and its discriminant and therefore the polynomial $f \mod p \in \mathbb{F}_p[X]$ has also the degree $n$ and a nonzero discriminant.

Step (2): Use Berlekamp’s algorithm to $f \mod p$ divided by its leading coefficient to obtain a monic irreducible factor $h_0 \in \mathbb{F}_p[X]$ of $f \mod p$. If $\deg h_0 = \deg f$, $f$ is irreducible in $\mathbb{Q}[X]$ and the algorithm stops. So from now on assume that $\deg h_0 < \deg f$.

Step (3): Find the least integer $\mu$ with

$$p^{2\mu \deg h_0} > 2^{n(n-1)} \cdot \frac{2(n-1)}{n-1} \cdot q(f)^{2n-1}.$$
Step (4): In this step we make use of Hensel’s algorithm which we will not describe but can be obtained from [GG]. It outputs a monic polynomial $h \in \mathbb{Z}[X]$ with $h_0 = h \mod p$ and $h \mod p^\mu$ divides $f \mod p^\mu$ in $(\mathbb{Z}/p^\mu\mathbb{Z})[X]$. Hensel’s algorithm only works because $\Delta(f) \neq 0 \mod p$ but it ensures that the polynomial exists and is unique modulo $p^\mu$.

Step (5): Now we make use of the LLL algorithm. The lattice $L$ we would like to apply it on is given as the additive subgroup of $\mathbb{Z}[X]$ with basis $p^\mu, p^\mu \cdot X, \ldots, p^\mu \cdot X^{(\deg h)-1}, h, X \cdot h, \ldots, X^{n-1-\deg h} \cdot h$.

It is easily seen that $L$ is a sublattice of $\mathbb{Z}[X]_n$. After applying the LLL algorithm to this basis one obtains the reduced basis $b_1, \ldots, b_n$ of $L$.

Step (6): In the last step, test if

$$q(b_1) > 2^{n-1} \cdot \binom{2(n-1)}{n-1} \cdot q(f).$$

If the inequality is satisfied $f$ is irreducible and the algorithm terminates. Otherwise, we can find a divisor of $f$ by computing $g = \gcd(b_1, f)$ by using the Euclidean algorithm. Like in Berlekamp’s algorithm we can continue to factor $g$ until it is irreducible. Then factor $f/g$ and combine the solutions to obtain a complete factorization of $f$. This ends the description of the algorithm. Now let us address the issue of its correctness.

Steps (1) to (5) only use algorithms we already presented like Berlekamp’s algorithm or the LLL algorithm. In the case of Hensel’s algorithm we can assume that it runs correctly on suitable input. Apart from these algorithms the algorithm just computes specific integers, so there is no need to check for correctness here. The interesting step is clearly Step(6) where it is decided whether the polynomial $f$ is either irreducible, or otherwise a factor is computed. To see why this step is correct we use the following lemma.

**Lemma 3.16.** The following statements with setting as in the algorithm described above are equivalent:

(i) $f$ is reducible,

(ii) $q(b_1) \leq 2^{n-1} \cdot \binom{2(n-1)}{n-1} \cdot q(f)$,

(iii) $f$ and $b_1$ have a common factor of positive degree in $\mathbb{Z}[X]$.

**Proof.** (iii) $\Rightarrow$ (i) is obvious since $\deg b_1 < n = \deg f$. For (i) $\Rightarrow$ (ii) observe that since $\Delta(f) \neq 0 \mod p$ there exists a $g$, unique up to sign, which is an irreducible factor of $f$
in \( \mathbb{Z}[X] \) and \( h_0 \) divides \( g \mod p \). Hensel’s lemma then gives us that \( h \mod p^\mu \) divides \( g \mod p^\mu \) in \( \mathbb{Z}/p^\mu \mathbb{Z}[X] \). Since we assume \( (i) \) to be true we have \( \deg g < \deg f = n \). Hence \( g \in L \). There is a general inequality of Mignotte, shown in [PRA], which states that

\[
q(g) \leq \left( \frac{2 \cdot \deg g}{\deg f} \right) \cdot q(f) \leq \left( \frac{2(n-1)}{n-1} \right) \cdot q(f).
\]

We also know that \( b_1, \ldots, b_n \) is a LLL reduced basis of \( L \) and therefore it is \( q(b_1) \leq 2^{n-1}q(g) \). Combining these inequalities leads to \( (i) \Rightarrow (ii) \). For the last proof of \( (ii) \Rightarrow (iii) \) let \( m = p^\mu \) and \( g = b_1 \). Then it holds that

\[
\|g\|^{2\deg f} \cdot \|f\|^{2\deg g} \leq \|g\|^{2n} \cdot \|f\|^{2(n-1)} \\
\leq 2^{n(n-1)} \cdot \left( \frac{2(n-1)}{n-1} \right)^n \cdot q(f)^{2n-1} \\
< m^{2\deg h}.
\]

The first inequality is true since \( \deg f = n \) and \( g = b_1 \in L \). The second one is just the assumption of \( (ii) \) and the last one follows from the choice of \( \mu \) in Step (3) of the algorithm. Hence we have \( \|f\|^{\deg g} \cdot \|g\|^{\deg f} \leq m^{\deg h} \). Obviously by the properties of \( h \) the condition \( \deg f + \deg g \geq \deg h \) is also fulfilled. Altogether the setup for Lemma 3.15 is given, thus \( f \) and \( g = b_1 \) have a common factor which finishes the proof.

This proof also explains why the algorithm does indeed work in the desired way. It shows that the property of a lattice basis to be LLL reduced can be helpful in many ways. Hereby we terminate our investigation of lattices and the LLL algorithm to see another practical usage of it in cryptography.
4 The NTRU cryptosystem

In 1996 the three scientists Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman published a new cryptosystem which they called NTRU. This is an abbreviation for N-th degree truncated polynomial ring that is the mathematical structure the cryptosystem is based on. From a cryptographical point of view NTRU, being less than 20 years old by now, is a very young approach for providing cryptographical security. Nevertheless, it is widely used nowadays, especially in mobile devices, due to its relatively low consumption of computational power compared to other cryptosystems in practical use such as RSA.

NTRU is a public key cryptosystem and can therefore be used not only to ensure a cryptographically secure communication but also to provide authorization by digital signatures. The underlying hard problem is the Shortest Vector Problem in lattices that we have already seen in chapter 3. One advantage of this problem is that there are no algorithms known for quantum computers to solve the shortest vector problem in polynomial time. Thus, at least up to now, the NTRU cryptosystem is also secure against attacks that use quantum computers if they will ever become a practical tool for cryptographic attacks. This is not the case for many other cryptoschemes as for example the RSA cryptosystem.

4.1 Notation of NTRU

The NTRU cryptosystem operates in the ring \( R = \mathbb{Z}[X]/(X^n - 1) \) which is the ring of polynomials with integer coefficients and degree at most \( n - 1 \). A polynomial \( f \in R \) can be written as either a polynomial or a vector, i.e.

\[
f = \sum_{i=0}^{n-1} f_i x^i = [f_0, f_1, \ldots, f_{n-1}],
\]

where the \( f_i \) are integers.

The mathematical operation \( \times \) is called the multiplication in \( R \). It is explicitly given as a cyclic convolution product,

\[
f \times g = h, \quad \text{where} \quad h_k = \sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{n-1} f_i g_{n+k-i} = \sum_{i+j \equiv k \mod n} f_i g_j.
\]

A multiplication of \( f \) and \( g \) over \( q \) means that we simply reduce the coefficients of the product modulo \( q \).
To avoid misunderstandings the following notations are mentioned.

\( n \) The dimension of the polynomial ring \( R \) the NTRU cryptosystem works in

\( p \) A positive integer defining the ring \( \mathbb{Z}/p\mathbb{Z} \) where the coefficients of a polynomial will be reduced modulo \( p \) in the encryption and decryption process

\( q \) A positive integer defining the ring \( \mathbb{Z}/q\mathbb{Z} \) where the coefficients of a polynomial will be reduced modulo \( q \) in the encryption and decryption process. Also used for the construction of the public key

\( d_f \) Controls the distribution of the coefficients of the polynomial \( f \)

\( d_g \) Controls the distribution of the coefficients of the polynomial \( g \)

\( d_r \) Controls the distribution of the coefficients of the random polynomial \( r \)

\( f \) A polynomial in \( \mathbb{Z}[X]/(X^n - 1) \)

\( f_p \) The reduction of \( f \) modulo \( p \) which lies in \( \mathbb{Z}[X]/(p, X^n - 1) \)

\( f_q \) The reduction of \( f \) modulo \( q \) which lies in \( \mathbb{Z}[X]/(q, X^n - 1) \)

\( g \) A polynomial used with \( f_q \) to create the public key. It lies in \( \mathbb{Z}[X]/(q, X^n - 1) \)

\( L_f \) The set of polynomials in \( \mathbb{Z}[X]/(x^n - 1) \) with coefficients satisfying \( d_f \)

\( L_g \) The set of polynomials in \( \mathbb{Z}[X]/(x^n - 1) \) with coefficients satisfying \( d_g \)

\( L_r \) The set of polynomials in \( \mathbb{Z}[X]/(x^n - 1) \) with coefficients satisfying \( d_r \)

\( f_p^{-1} \) The inverse of \( f_p \) in \( \mathbb{Z}[X]/(p, X^n - 1) \)

\( f_q^{-1} \) The inverse of \( f_q \) in \( \mathbb{Z}[X]/(q, X^n - 1) \)

\( h \) The public key polynomial in \( \mathbb{Z}[X]/(q, X^n - 1) \)

\( r \) A random polynomial in \( \mathbb{Z}[X]/(q, X^n - 1) \)

\( m \) The plaintext message which is a polynomial in \( \mathbb{Z}[X]/(p, X^n - 1) \)

\( e \) The ciphertext message which is a polynomial in \( \mathbb{Z}[X]/(q, X^n - 1) \)
4.2 Key Generation

Let us assume that Alice wants to transmit a cryptographic secure message to Bob. At first, as in any public key cryptosystem, Bob has to compute and publish his public key in order to enable Alice to encrypt her message with it. For this purpose, Bob randomly chooses two polynomials \( f \in L_f \) and \( g \in L_g \). It is important that the polynomial \( f \) has inverses modulo \( q \) and modulo \( p \). According to \([\text{NTRU}]\) this will be true with high probability for certain parameter choices. The computation of these inverses is fairly easy by using a modification of the Euclidean algorithm. In practice one would simply randomly choose an \( f \) and check if it has inverses modulo \( q \) and \( p \). If this is not the case one would discard \( f \) and create a new one. This process is repeated until one finally chose an appropriate \( f \), which, as mentioned above, should not take too long.

So finally Bob computed \( f^{-1}_q \) and \( f^{-1}_p \) such that

\[
    f^{-1}_q \times f \equiv 1 \mod q \quad \text{and} \quad f^{-1}_p \times f \equiv 1 \mod p.
\]

In the next step Bob computes

\[
    h \equiv pf^{-1}_q \times g \mod q.
\]

This polynomial \( h \) will be the public key of Bob where the corresponding private key will be the polynomial \( f \).

4.3 Encryption

Now we will see what Alice has to do in order to ensure a cryptographically secure communication with Bob. Of course she will have to make use of Bob’s public key in the process. At first she chooses her plaintext \( m \) from the message space

\[
    m \in P_p(n-k),
\]
for a certain $k \leq n - 1$ where $P_p(n - k)$ is the set of all polynomials with degree at most $n - k - 1$ and $\mod p$ coefficients. She will also choose a random polynomial $r \in L_r$. Finally, she can encrypt her message by computing

$$e \equiv r \times h + m \mod q.$$ 

The message $e$ is the encrypted message and will therefore be sent to Bob.

### 4.4 Decryption

Upon receiving message $e$ from Alice, Bob needs to decrypt it by using his private key $f$. It is also helpful if he already precomputed and stored the polynomial $f_p^{-1}$. In the first step of decryption Bob computes the polynomial $a$ by

$$a \equiv f \times e \mod q,$$

where the coefficients of $a$ are chosen such that they lie in the interval from $-q/2$ to $q/2$. As next step Bob needs to compute the polynomial $t \in \mathbb{Z}[X]/(p, X^n - 1)$ by

$$t \equiv f_p^{-1} \times a \mod p.$$

Then the polynomial $t$ should be equal to the original message $m$. Let us take a look why this decryption scheme works. It holds that

$$a \equiv f \times e \mod q$$
$$\equiv f \times r \times h + f \times m \mod q$$
$$\equiv f \times pr \times f_q^{-1} \times g + f \times m \mod q$$
$$\equiv pr \times g + f \times m \mod q.$$  

If the parameter choices satisfy some conditions, it is possible to ensure that almost always when Bob reduces the coefficients of $f \times e$ modulo $q$ into the interval from $-q/2$ to $q/2$ he recovers exactly the polynomial

$$a = pr \times g + f \times m \quad \text{in } R.$$
After that, Bob has to reduce \( a \) modulo \( p \) which gives him the polynomial

\[
f \times m \quad \text{in } R,
\]

and finally after multiplying by \( f_p^{-1} \) we get

\[
t = m \quad \text{in } \mathbb{Z}[X]/(p, X^n - 1).
\]

As mentioned above Bob does indeed recover exactly the original message \( m \) if the reduction step modulo \( q \) has no effect on the polynomial \( pr \times g + f \times m \), i.e. all coefficients were already in the interval from \(-q/2\) to \( q/2\). According to [NTRU] with appropriate parameter choices of \( d_f, d_g \) and \( d_r \) this will be almost always the case. If, however, against all odds a real reduction modulo \( q \) happens, Bob will not be able to recover the correct message \( m \) anymore. In this case both Alice and Bob will have to agree on new security parameters and repeat the en- and decryption steps.

### 4.5 Lattice attack on NTRU

After NTRU was first introduced in 1996, the first notable concept to attack NTRU was given by Coppersmith and Shamir. To that time it was already known that the key of an NTRU encryption is a rather short vector of a certain lattice. But according to the developers of NTRU that does not cause any security problems since the key was somehow surrounded by many other short vectors and the possibility for any known algorithm to find the real key would be so small that it could be neglected.

However Coppersmith and Shamir found an attack that does not really need the exact private key \( f \) but may find an alternative key \( f' \) which is equally useful for decrypting. After learning about this attack possibility, NTRU developers thought about countermeasures where the most obvious one was to increase the parameters of the encryption, leading to higher dimensional lattices. This would make it far more difficult to find suitable vectors for decryption but at the same time enlarge the key size and extend the time consumption of the NTRU cryptosystem.

#### 4.5.1 Setup

At first we need to understand why an attack on NTRU does indeed correspond with the Shortest Vector Problem in lattices. We already know that the parameters \( h, q \) and \( n \)
of any NTRU setup are publicly known to everyone. It is therefore no problem for any attacker to define the so called NTRU lattice $L$

$$L := \begin{pmatrix} \lambda I_n & H \\ 0 & q I_n \end{pmatrix} = \begin{pmatrix} \lambda & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{n-1} \\ 0 & \lambda & \cdots & 0 & h_{n-1} & h_0 & \cdots & h_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda & h_1 & h_2 & \cdots & h_0 \\ 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{pmatrix}.$$ 

Here $I_n$ is the $n$-dimensional identity matrix, $\lambda$ is a parameter that will be chosen later and $H$ is the matrix consisting of the circularly shifted entries of the public key $h$. The relation $h = f_q^{-1} \times g \mod q$ of the NTRU setup leads to $f \times h - qu = g$ for some $u \in R$. This leads directly to the relation

$$(f, -u) \times L = (f, -u) \times \begin{pmatrix} \lambda I_n & H \\ 0 & q I_n \end{pmatrix} = (\lambda f, g).$$

So the vector $(\lambda f, g)$ consisting of the private keys $f$ and $g$ is indeed a short vector in the lattice spanned by $L$. There is a high chance that it is even a shortest vector and therefore inspires the attacker to use reduction algorithms on $L$ to recover $f$ and $g$.

To understand how the attack of Coppersmith and Shamir works we need an additional structure on the polynomial ring $R$.

**Definition 4.1.** For $x \in R$ it is

$$\bar{x} = \frac{1}{n} \sum_{i=0}^{n-1} x_i$$

$$|x|_1 = \left( \frac{n}{\sum_{i=0}^{n-1} (x_i - \bar{x})^2} \right)^{1/2}.$$  

It is easily seen that the definition above is just the standard deviation of the entries of $x$ scaled by the factor $\sqrt{n}$. Therefore it should also be clear that adding the vector $t1$, that is the vector of length $n$ and only $t$ as entries, does not change this norm.
Lemma 4.2. It is

\[ |x \times y|_1 \approx |x|_1 |y|_1. \quad (4.3) \]

Proof. Let \( x_i = \bar{x} + w_i \) and \( y_j = \bar{y} + z_j \). Then we get

\[
|x \times y|^2 = \sum_k [(x \times y)_k - x \times y]^2 \\
= \sum_k (w \times z)_k^2 \\
= \sum_k \left( \sum_i w_i z_{k-i} \right) \left( \sum_j w_j z_{k-j} \right),
\]

where the indices are considered modulo \( n \). Taking a closer look to each product \( w_i z_{k-i} w_j z_{k-j} \) in the sum, we see that the difference between the \( w \)-indices, namely \( j - i \), is the same as the difference between the \( z \)-indices, which is \( (k - i) - (k - j) \). We can therefore rewrite the sum by defining \( d = i - j \) and \( l = k - j \) as

\[
|x \times y|^2 = \sum_d \left( \sum_i w_i w_{i+d} \right) \left( \sum_l z_l z_{l+d} \right) \\
\quad = \left( \sum_i w_i^2 \right) \left( \sum_l z_l^2 \right) + \sum_{d \neq 0} \left( \sum_i w_i w_{i+d} \right) \left( \sum_l z_l z_{l+d} \right) \\
\quad = |x|_1^2 |y|_1^2 + \sum_{d \neq 0} \left( \sum_i w_i w_{i+d} \right) \left( \sum_l z_l z_{l+d} \right).
\]

Now let us take a closer look at the last sum. If we can show that it is somehow small, we proved the lemma. It is quite reasonable to assume that \( w \) and \( z \) behave like random vectors. From statistics it is well known that each of the \( n - 1 \) terms where \( d \neq 0 \) leads to the auto correlation coefficient \( \sum_i w_i w_{i+d} \) which should be smaller than the sum \( \sum_i w_i^2 \) where \( d = 0 \) by a factor nearly about the size of \( 1/\sqrt{N} \). The same holds of course for the sum \( \sum_l z_l z_{l+d} \). So if we put everything together, the product of the two sums should be smaller by a factor of \( 1/N \). Since we assume all these terms to have a random sign there should even be some cancellation and therefore the sum over nonzero values of \( d \) should be
much smaller than the term corresponding to $d = 0$. So overall we have the situation that

$$|x \times y|_\perp^2 = |x|_\perp^2 |y|_\perp^2 + \text{smaller terms}$$

$$|x \times y|_\perp = |x|_\perp |y|_\perp + \text{smaller terms}.$$ 

\[\square\]

Now recall the NTRU decryption process. At some point Bob recovers the polynomial

$$a = pr \times g + f \times m.$$ 

As mentioned in the explanation of the decryption, the process is only guaranteed to work if all entries of $a$ lie in the interval from $-q/2$ to $q/2$. We now want to use our observations from above to estimate the probability of decryption errors. By using approximation (4.3) we get that

$$|pg \times r|_\perp \approx p|g|_\perp |r|_\perp.$$ 

Note that it is possible to control the norm of $r$ by the parameter $d_r$. Analogously we have that

$$|f \times m|_\perp \approx |f|_\perp |m|_\perp.$$ 

Assuming that the vectors of $pg \times r$ and $f \times m$ are nearly orthogonal, the following approximation holds.

$$|a|_\perp^2 = |pg \times r + f \times m|_\perp^2$$

$$\approx |pg \times r|_\perp^2 + |f \times m|_\perp^2$$

$$\approx p^2|g|_\perp^2 |r|_\perp^2 + |f|_\perp^2 |m|_\perp^2.$$ 

This gives us the important approximation

$$|a|_\perp^2 \approx (p^2 |r|_\perp^2)|g|_\perp^2 + (|m|_\perp^2)|f|_\perp^2.$$ 

(4.4)

The last assumption we need is that the entries of $a$ are normally distributed with mean near 0 and standard deviation $\sigma \approx |b|_\perp / \sqrt{n}$. Coppersmith and Shamir investigated what happens if $q/2$ is a multiple of $\sigma$ and computed the probability of an error in the decryption process. They concluded that with $n = 167$ and $q/2 = 5\sigma$ the probability that at least one
of the 167 entries of $a$ is decrypted wrongly is about $3.30 \times 10^{-7}$ which is acceptably low. Therefore they suggested arranging the parameters such that $\sigma < q/10$ to ensure an error free decryption.

### 4.5.2 Alternative decryption

In the last section we saw a direct relation of the reliability of decoding to the ratio $q/\sigma$, with $\sigma \approx |b|_1 / \sqrt{n}$. With equation (4.4) we are able to approximate $|a|_\perp$ by using $|f|_\perp$ and $|g|_\perp$.

The next step is the most important one in the attack. It shows that in many cases it is not even necessary to correctly retrieve the private key $f$ to fully decrypt the messages, but that it suffices to compute certain $f'$ and $g'$ that have an equally good use for decryption. So at first choose another $n$-vector polynomial $f'$. From the usual NTRU relation $h \times f' \equiv pg'$ mod $q$, and since $h$ is publicly known, it is possible to compute a corresponding vector $g'$. Then again we use equation (4.4) to estimate $|a'|_\perp$ and compare it to $|a|_\perp$. If it is not much larger than $|a|_\perp$, we setup a decoder using only $f'$ and $g'$. Putting the notions above in equations we get

\[
pg' \equiv f' \times h \mod q
\]

(4.5)

\[
|a'|_\perp^2 \approx (p^2|r|_\perp^2) |g'|_\perp^2 + (|m|_\perp^2) |f'|_\perp^2.
\]

(4.6)

By keeping $|r|_\perp$ and $|m|_\perp$ constant we can set

\[
\lambda = \frac{|m|_\perp}{p|r|_\perp},
\]

which leads to

\[
\sigma'^2 = \frac{|a'|_\perp^2}{n} \approx \left(\frac{p^2|r|_\perp^2}{n}\right) \left(|g'|_\perp^2 + \lambda^2 |f'|_\perp^2\right).
\]

The next task would be to design a lattice that consists of elements which correspond to the choices of $f'$ and $g'$. Let us take a look at the lattice

\[
L' = \begin{pmatrix}
\lambda I_n & 0 \\
H & qI_n
\end{pmatrix}.
\]

As usual $I_n$ is the $n \times n$ identity matrix and $H$ is the matrix which has columns that consist of circular shifted versions of the vector $hp_{q^{-1}} \mod q$. $p_{q^{-1}}$ is just the inverse of $p \mod q$, i.e. $p_{q^{-1}}p = 1 \mod q$. Now let us take a look at what happens if we multiply $L'$
with a $2n$-dimensional vector of the form $[f', x]^t$, where $x$ is just an arbitrary $n$-dimensional integer vector. It then holds that

$$v'_{f',x} = L' \begin{pmatrix} f' \\ x \end{pmatrix} = \begin{pmatrix} \lambda f' \\ g' \end{pmatrix},$$

where $g'$ must satisfy $pg' = f' \times h \mod q$ and $x$ will get multiplied by $q$.

Now we would like to modify the vectors $\lambda f'$ and $g'$ in such a way, that their mean is zero. Therefore we will simply take each column vector $v$ in the top half of $L'$ and subtract the constant vector $\bar{v}1_n$, where as usual $\bar{v}$ is the mean of $v$. Obviously after this adjustment every vector $v$ will have zero mean. The same procedure is done for every vector $w$ in the bottom half of $L'$.

We then get a new matrix $L$

$$L = \begin{pmatrix} \lambda I_n - (\lambda/n)J_n & 0 \\ H - \alpha J_n & qI_n - (q/n)J_n \end{pmatrix},$$

with $J_n$ being the $n \times n$ matrix of all 1's and $\alpha$ is a certain scalar. This transformation ensured that we now get

$$v'_{f',x} = L \begin{pmatrix} f' \\ x \end{pmatrix} = \begin{pmatrix} \lambda(f' - \bar{f}'1_n) \\ g' - \bar{g}'1_n \end{pmatrix}.$$

Since the mean of $\lambda f'$ and $g'$ is now zero their $L^2$ norm coincides with the $| \cdot |_\perp$ norm. Therefore it holds that

$$|v'_{f',x}|^2 = \lambda^2 |f'|_\perp^2 + |g'|_\perp^2 = \left( \frac{1}{p^2|r|_\perp^2} \right) (|m|_\perp^2 |f'|_\perp^2 + p^2 |r|_\perp^2 |g'|_\perp^2)$$

$$= \left( \frac{1}{p^2|r|_\perp^2} \right) |b'|^2.$$

By varying $x$ we can achieve different values for $|b'|^2$. We already showed above that one would like to have the norm of $b'$ to be as small as possible in order to ensure an error free decryption with the key $f'$. Therefore it makes sense to choose $x$ in such a way that it minimizes the norm of $b'$, i.e.

$$n_{f'} = (p|r|_\perp) \min_x |v'_{f',x}| = |b'|.$$
With this parameter one can directly judge if any short vector, for example as output of the LLL algorithm, can be used for decryption. It was already mentioned that $f$ should satisfy $n_f < q/10$ in order reduce the probability of a decoding failure to an acceptably low level. If $f$ is the private key of Bob and the attacker manages to compute a short vector $f'$ with

$$n_{f'} \leq n_f,$$

it is directly possible to use $f'$ as decryption key, given the very weak additional condition that there must exist an inverse of $f'$ modulo $p$. The attack works because the adversary intercepts the encrypted message $e$ and then computes

$$f' \ast e = f' \ast (r \ast h + m) \mod q$$
$$= f' \ast pr \ast f_q^{-1} \ast g' + f' \ast m \mod q$$
$$= pr \ast g' + f' \ast m \mod q,$$

and again, exactly as in the usual decryption process it is very likely that the last polynomial is recovered correctly since $f'$ has a smaller norm than $f$. So as last step the attacker just needs to reduce this last polynomial modulo $p$ and multiply with $f'_p^{-1}$ to receive the original message $m$. Now let us assume that we found two vectors $f_1$ and $f_2$ with $n_{f_1} = n_{f_2} \leq 2.5 \cdot n_f$. According to [CS] it can still be possible to recover $m$ by means of linear algebra because both of these vectors contain partial information about $m$.

As conclusion let us think about the different scenarios that can occur. In the first one, there are many short vectors satisfying $n_{f'} \leq n_f$. In this case it is very likely that the adversary finds one of these and simply uses it for decryption. In the next case the private key $f$ is much shorter than any other vector in the lattice of $L$, but then it is very likely that the adversary directly recovers $f$. Obviously in both these cases the encryption scheme would be insecure. So the worst case scenario for an attacker to happen, is that there exist many vectors $f'$ in the lattice that are not much longer than $f$, for example $n_{f'}$ is about 10 times larger than $n_f$. It would be hard for any algorithm to find the vector $f$ as it is not much shorter than all the other vectors $f'$ and simultaneously the vectors $f'$ are too large to use them for decryption.

After learning about these lattice based attack types on NTRU the designers used higher security parameters to increase the dimension of the lattice $L$ which makes it very hard to find good vectors that are suitable for decryption. This comes of course with the
price of bigger public keys and longer encryption time.

4.6 Practical results

In the section above we saw a theoretical approach to an attack on the NTRU cryptosystem by the means of lattice reduction. The attack relies heavily on the possibility to find a short vector in the NTRU lattice described earlier. Already in theory there are obvious problems that occur when one will try to use this attack pattern in practice. At first it is, up to now, not possible to obtain a shortest lattice vector in polynomial time with respect to the lattice dimension $n$. Since exponential time algorithms are of no practical use, at least if the lattice dimension is high enough, one has to rely on algorithms that run faster which comes of course with the trade of not exactly solving the Shortest Vector Problem.

One of the most popular algorithms to address this issue is the LLL algorithm designed by Lenstra, Lenstra and Lovász. However it is only guaranteed that it will output a vector with length not too far away of the length of a shortest vector. The first question coming into mind is probably how this inaccuracy will influence the attack scheme in practice. It is also important to remember that a polynomial time algorithm does in no way guarantee a fast solution, it just provides at least a slight chance to work on higher dimensional lattices where the use of an exponential algorithm would be hopeless in terms of time consumption.

For this investigation I implemented the LLL algorithm and the NTRU cryptosystem as described above in sage. It is an open source math program with sufficient features to show the practical side of the NTRU lattice attack. The attack scheme is pretty easy to understand and directly relates to our theoretical investigations. At first the algorithm will create the NTRU lattice out of the publicly known parameters of the cryptosystem. After that it will use the LLL algorithm to obtain a fairly short vector of the lattice and use this one as an alternative private key and decrypt the message.

We already learned that the NTRU lattice defines a lattice consisting of vectors of the type $(\lambda f', g')$, where $pg' = f' \times h \mod q$. We did not exactly state how the parameter $\lambda$ should be chosen in order to maximize the possibility of correct decryption with the obtained short vector given by the LLL algorithm. As the results will show it is possible to choose $\lambda$ to be equal to one but there are other, more promising choices. By using the Gaussian heuristic one gets that the expected size of a shortest vector in a random lattice is between

$$D^{1/N} \sqrt{\frac{N}{2\pi e}} \quad \text{and} \quad D^{1/N} \sqrt{\frac{N}{\pi e}},$$
where \( D = q^n \lambda^n \) is the determinant of our lattice and \( N = 2n \) is its dimension. Plugging these values into the equation we get that the expected length is slightly larger than

\[
s = \sqrt{\frac{n \lambda q}{\pi e}}.
\]

The best probability of the LLL algorithm to correctly recover the vector \((\lambda f', g')\) will be the case where the ratio \( s/|\langle \lambda f', g' \rangle| \) is maximized because then it is assured that \((\lambda f', g')\) is not too far away from being a shortest vector. Squaring this ratio leads to the problem of choosing \( \lambda \) such that

\[
\lambda = \sqrt{\frac{\lambda^2 |f|^2 + \lambda^{-1} |g|^2}{\lambda^2 |f|^2 + |g|^2}} = (\lambda |f|^2 + \lambda^{-1} |g|^2)^{-1},
\]

is maximized, which is fulfilled for \( \lambda = |g|/|f| \). Since these are both publicly known values it is perfectly possible for an attacker to choose \( \lambda \) in such a way. Keep in mind that this is important if the attacker tries to correctly retrieve the original private key.

For the practical experiments I chose the public parameters \( p = 3 \) and \( q = 128 \) just as it is proposed in a moderate security NTRU setup. Then different choices for \( n \), the degree of \( f \), resulted in the following outcome.

<table>
<thead>
<tr>
<th>Lattice dimension</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>22</th>
<th>30</th>
<th>42</th>
<th>46</th>
<th>54</th>
<th>62</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time in seconds</td>
<td>0.05</td>
<td>0.08</td>
<td>0.18</td>
<td>0.76</td>
<td>2.59</td>
<td>9.15</td>
<td>14.95</td>
<td>30.00</td>
<td>65.02</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lattice dimension</th>
<th>70</th>
<th>78</th>
<th>94</th>
<th>102</th>
<th>122</th>
<th>130</th>
<th>146</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time in seconds</td>
<td>104.04</td>
<td>149.56</td>
<td>348.23</td>
<td>485.70</td>
<td>1130.78</td>
<td>1512.58</td>
<td>2597.05</td>
</tr>
</tbody>
</table>

According to [COH] the run time of the LLL algorithm and therefore our attack is at most \( O(n^6 \ln^3 B) \) if \(|b_i|^2 < B \) for all \( i \). Since the basis vectors \( b_i \) of the NTRU lattice cannot get arbitrarily large and many entries are actually zero, the outcome should rely almost completely on the lattice dimension. Therefore I plotted my results and tried to fit them with a polynomial function of degree six.
Clearly, the polynomial describes the results nearly perfectly. Considering that the lowest security NTRU setup introduced in [NTRU] suggests choosing $n$ to be equal to 167, which would lead to a lattice dimension of 334, I would never have a chance to actually launch an attack against it. Also keep in mind that it is not guaranteed to get an alternative key which decrypts the message correctly. In my experiments this was only true about 83% of the time, either because the so found alternative key $f'$ was not invertable modulo $p$ or it was just not suitable for decryption.

Nevertheless, this attack was very important for the NTRU cryptosystem. It forced the company to use larger key sizes in order to maintain high security. It was also inspiring to other attacks, for example of Alexander Mey who exploited the fact that the NTRU lattices contain many zero entries which lead to a faster computation of a short vector.
A

Source Code

This is my implementation of the attack on the NTRU cryptosystem as discussed earlier.
The first part contains every function needed for the LLL algorithm.

#The function RED of the LLL algorithm

```python
def RED(k,l,B,MU):
    if (abs(MU[k-1,l-1]) > 0.5):
        q = (0.5+MU[k-1,l-1]).floor();
        B.set_column(k-1, B.column(k-1) - q*B.column(l-1));
        MU[k-1,l-1] = MU[k-1,l-1] - q;
        for i in range(1,l):
            MU[k-1,i-1]=MU[k-1,i-1] - q* MU[l-1,i-1];
    return
```

#The function SWAP of the LLL algorithm

```python
def SWAP(k,B,MU,B_vector,k_max):
    exchange = B.column(k-1);
    B.set_column(k-1, B.column(k-2));
    B.set_column(k-2,exchange);
    if (k>2):
        for j in range(1,k-1):
            exchange = MU[k-1,j-1]
            MU[k-1,j-1]=MU[k-2,j-1]
            MU[k-2,j-1]=exchange
        mu=MU[k-1,k-2]
        B_skalar = B_vector[k-1]+mu^2*B_vector[k-2]
        MU[k-1,k-2] = mu*B_vector[k-2]/B_skalar
        B_vector[k-1]= B_vector[k-2]*B_vector[k-1]/B_skalar
        B_vector[k-2]= B_skalar
```
for i in range(k+1,k_max+1):
    t=MU[i-1,k-1]
    MU[i-1,k-1] = MU[i-1,k-2] - mu * t
    MU[i-1,k-2] = t + MU[k-1,k-2]*MU[i-1,k-1]
return

#The main part of the LLL algorithm. By using Gram-Schmidt orthogonalization one
#obtains a Gram-Schmidt matrix which is used in the sub-algorithms SWAP and RED.
def LLL(B):
    #Step1
    k = 2;
    k_max = 1;
    n = B.ncols();
    m=B.nrows();
    B_vector = vector(QQ,n);
    MU=matrix(QQ, n,n);
    B_GramSchmidt = matrix(QQ, m,n);
    B_GramSchmidt.set_column(0,B.column(0));
    B_vector[0]= B.column(0)*B.column(0);
    if (B_vector[0] == 0):
        print "Error: The input did not form a basis."
        return;

    #Step2
    while(k <= n):
        if(k>k_max):
            k_max = k;
            for l in [1..k]:
                B_GramSchmidt.set_column(l-1, B.column(l-1));
        for j in range(1,1):
            MU[l-1,j-1] = B.column(l-1)*B_GramSchmidt.column(j-1)
            /B_vector[j-1];
B_GramSchmidt.set_column(l-1, B_GramSchmidt.column(l-1) - MU[l-1,j-1]*B_GramSchmidt.column(j-1));
B_vector[l-1] = B_GramSchmidt.column(l-1) * B_GramSchmidt.column(l-1);
if (B_vector[l-1] == 0):
    print "Error: The input did not form a basis."
    return;

#Step3
RED(k,k-1,B,MU);
while(B_vector[k-1] < (0.75 - MU[k-1,k-2]^2)*B_vector[k-2]):
    SWAP(k,B,MU,B_vector,k_max);
    k=max(2,k-1);
    RED(k,k-1,B,MU);
for l in range (k-2,0,-1):
    RED(k,l,B,MU)
k=k+1;

#Step 4
return B
The following functions are all I used to create the NTRU encryption setup and to launch an attack against it.

```python
#Function to reduce the coefficients of a given polynomial f of degree n into the interval (-p/2,p/2).

def modCoeffs(f,n,p):
    i = 0
    f_mod_p = 0
    R.<x> = ZZ[]
    S = R.quotient(x^n-1, 'x'); x = S.gen()
    for c in f.list():
        c = c % p
        if c > p / 2:
            c -= p
        f_mod_p += c*x^i
        i += 1
    return f_mod_p

#Function to compute an inverse of a polynomial f in the ring of integer polynomials modulo x^n-1 with coefficients reduced modulo q.

def inverse(f,n,q):
    (pgcd,u,v) = xgcd(f,x^n-1);
    p_1 = inverse_mod(pgcd[0],q);
    u = p_1*u;
    f_inverse = modCoeffs(u,n,q);
    return f_inverse;

#Function to create a random polynomial f with maximum degree n and L_f1 ones, L_f2 minus ones and the rest zero coefficients.
```
def createPolynomials(n,L_f1,L_f2):
    from random import shuffle
    list = [1 for i in range(L_f1)] + [-1 for i in range(L_f2)]
    + [0 for i in range(n-L_f1-L_f2)]
    shuffle(list)
    R.<x> = ZZ[]
    f = sum([x^i*list[i] for i in [0..n-1]])
    return f

#Function to create all necessary parts of an NTRU cryptosystem. Namely,
#f and g are the private key pair, f_pINVERSE is the inverse of f
#modulo p and h is the public key.

def createNTRU_CRYPT(n,p,q,L_f1,L_f2,L_g1,L_g2):
    f = createPolynomials(n,L_f1,L_f2)
    print f
    g = createPolynomials(n,L_g1,L_g2)
    f_pINVERSE = inverse(f,n,p)
    f_qINVERSE = inverse(f,n,q)
    print modCoeffs(f*f_pINVERSE,n,p)
    print modCoeffs(f*f_qINVERSE,n,q)
    h = modCoeffs(3*f_qINVERSE*g,n,q)
    p_qINVERSE = inverse_mod(p,n)
    return f,g,f_pINVERSE,h

#Function to encrypt a message m with the private key h. The random
#polynomial consists of L_r1 ones, L_r2 minus ones and the rest zero
#coefficients.

def encryptNTRU(m,h,n,q,L_r1,L_r2):
    r = createPolynomials(n,L_r1,L_r2)
    e = modCoeffs(r*h+m,n,q)
return e

# Function to decrypt a message e by using the private key f and f_pINVERSE.

def decryptNTRU(e, f, f_pINVERSE, n, q, p):
    a = modCoeffs(f*e, n, q)
    t = modCoeffs(a*f_pINVERSE, n, p)
    return t

# Function to shift a vector by n positions.

def shift(l, n):
    return l[len(l)-n:]+l[:len(l)-n]

# Function to create the NTRU lattice out of the public key h and the orthogonal norms of f and g. You can either choose lamda to be equal to one or to be equal to the quotient of the norms of f and g.

def createNTRU_lattice(h, f, g, n, q, p):
    lamda = 1
    p_qINVERSE = inverse_mod(p, q)
    h_vector = vector(h.list())
    H = matrix(RR, [p_qINVERSE*h_vector])
    for i in [1..n-1]:
        h_vector = vector(shift(h.list(), i))
        H = H.stack(matrix([p_qINVERSE*h_vector]))
for i in range(H.nrows()):
    for j in range(H.ncols()):
        H[i,j] = H[i,j] % q
        if H[i,j] > q / 2:
            H[i,j] -= q
H = H.transpose()

#Combine the matrix H with the other parts of the NTRU lattice.
L1 = (alpha * matrix.identity(n)).augment(matrix(n))
L2 = H.augment(q*matrix.identity(n))
L = L1.stack(L2)
return L

#Function to determine the orthogonal norm of a polynomial f of maximum
#degree n.

def orthogonalNorm(f,n):
    f1 = 1/n * sum([f.list()[i] for i in [0..len(f.list())-1]])
    f_norm = sqrt(sum([(f.list()[i]-f1)^2 for i in [0..len(f.list())-1]]))
    return f_norm

#Function to determine the euclidean norm of a vector of a polynomial f of
#maximum degree n.

def norm(f,n):
    f_norm = sqrt(sum([(f.list()[i])^2 for i in [0..len(f.list())-1]]))
    return f_norm
References


[NTRU] Jeff Hoffstein, Daniel Lieman, Jill Pipher and Joseph H. Silverman, NTRU: A Public Key Cryptosystem,

[NIT] Abderrahmane Nitaj, Cryptanalysis of NTRU with two public keys,

[CS] Don Coppersmith and Adi Shamir, Lattice Attacks on NTRU,
http://link.springer.com/chapter/10.1007/3-540-69053-0_5

[LEN] Henrik W. Lenstra Jr., Lattices,
http://www.math.leidenuniv.nl/~psh/ANTproc/06hw1.pdf

[MR] Daniele Micciancio and Oded Regev, Lattice-based Cryptography, 2008,
https://www.cims.nyu.edu/~regev/papers/pqc.pdf

[MAY] Alexander May, Cryptanalysis of NTRU,
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.41.3484&rep=rep1&type=pdf


STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Date .................................................. (signature) ..................................................