Tommi Meskanen

On the NTRU Cryptosystem

Turku Centre *for* Computer Science

TUCS Dissertations
No 63, June 2005

On the NTRU Cryptosystem

by

Tommi Meskanen

To be presented, with the permission of the Faculty of Mathematics and Natural Sciences of the University of Turku, for public criticism in Auditorium XXI of the University on August 26th, 2005, at 12 noon

University of Turku
Department of Mathematics
FIN-20014 Turku, Finland

2005

Supervisors

Doctor Ari Renvall
Department of Mathematics
University of Turku
FIN-20014 Turku
Finland

Professor Juhani Karhumäki
Department of Mathematics
University of Turku
FIN-20014 Turku
Finland

Reviewers

Professor Cunsheng Ding
Hong Kong University of Science and Technology
Department of Computer Science
Clean Water Bay, Kowloon, Hong Kong
China

Doctor Valtteri Niemi
Nokia Research Center
PL 407
FIN-00045 Nokia Group
Finland

Opponent

Privat Dozent Dr. Walter Unger
Lehrstuhl für Informatik I
RWTH Aachen
D-52056 Aachen
Germany

ISBN 952-12-1570-4
ISSN 1239-1883
Painosalama Oy
Turku, Finland
2005

Acknowledgements

I would like to thank my supervisor Doctor Ari Renvall for his constant support during this work. Without his suggestions for research and his valuable comments on my work I would not have finished this thesis. With deepest gratitude, I thank my other supervisor Professor Juhani Karhumäki as well as Academy Professor Hannu Nurmi for their support and for offering outstanding working conditions.

Special thanks are due to Professor Cunsheng Ding and Doctor Valtteri Niemi for the preliminary examination of the thesis and their invaluable remarks. I would like also to thank Dr. Milla Kibble for the thorough English revision of this work.

I would like to thank the Department of Mathematics, the Department of Political Science and Turku Centre for Computer Science (TUCS) for providing excellent working environment, and the personnel of both departments and TUCS for the help and support. In particular, I would like to thank Dr. Jyrki Lahtonen for all the lectures on finite fields and Ph. Lic. Paula Steinby for advice of all kinds.

The support of Tekes, Nokia, Setec, Sonera and the Academy of Finland is gratefully acknowledged.

I would like to thank the math department floorball/orienteering club Luiskaotsat for providing goals and directions besides work.

Finally, I would like to thank my family for everything.

August, 2005

T. M.

Contents

CHAPTER 1

Introduction

There are two kinds of cryptosystems: private key cryptosystems and public key cryptosystems.

The private key cryptosystems are an ancient invention. Ever since the first written language, people have been developing ways to write down their secrets in a way only they can decipher. One of the simplest ways to cipher a message with a fixed alphabet is to replace each letter in the message with the next letter in the alphabet. This method is often called the Caesar method since there is documented evidence that Rome's emperor Caesar used ciphers like this.

One thing common to all private key cryptosystems is that if you know how to cipher a message then you automatically know how to decipher the message. Thus the method of ciphering must be kept secret. Once the ciphering method is compromised the cryptosystem is useless.

All known cryptosystems were private key cryptosystems until the introduction of the ideas of the public key cryptosystems by Diffie and Hellman [6] in 1976. Since then, several different protocols for public key cryptography have been presented (e.g. RSA 1978 [36], the first ideas of ECC in 1985 [2]).

The topic of this thesis is the public key cryptosystem NTRU. It was first introduced by Jeffrey Hoffstein, Jill Pipher and Joseph H. Silverman in 1998 [15]. It operates in the ring of truncated polynomials given by $\mathbb{Z}[X]/(X^N - 1)$. The security of the NTRU cryptosystem is based on the difficulty of finding short vectors in a certain lattice. The larger the parameter N, the more secure the system is.

NTRU is a probabilistic cryptosystem. The encryption process includes a random element and therefore one message has several possible encryptions. The advantage of NTRU over other cryptosystems is that

encryption and decryption are very fast and the key sizes are relatively small. Also the key generation is fast and easy.

The structure of the thesis is as follows:

In Chapter 2 we introduce the concept of public key cryptography.

In Chapter 3 we state the basic definitions and notations related to lattices and give some classical theorems. We also explain the vector problems considered in this work and discuss the properties of certain polynomial rings which will be used later.

There have been several different versions of the NTRU cryptosystem. In Chapter 4 we begin by noting the common ideas behind these different versions. Then we present the different versions in more detail, concentrating in particular on the latest version from 2003.

As already noted, the security of NTRU depends on the difficulty of certain lattice reduction problems. Some NTRU related lattices are easier to reduce than others of the same dimension. In Chapter 5 we explain what is meant by lattice reduction. We present some lattice constants which seem to serve as indicators of the successfulness of the reduction. We also present some tricks to enhance the lattice reduction process. Some of the tricks are previously known, but some appear to be mentioned for the first time in this thesis.

The problem of finding a short vector from a lattice can be solved by finding a short basis of the lattice. There are several algorithms to find a short basis. The faster algorithms, however, tend to find longer bases. In Chapter 6 we present the most commonly used basis reduction algorithms and highlight the properties of the reduced bases they produce.

In Chapter 7 we apply the ideas of the previous chapters to small instances of NTRU private keys. We used a computer to break the private keys from some small instances of NTRU-like private keys. From the breaking times of these small instances, we estimate the breaking times of the standard sized keys.

The short history of NTRU has seen many changes in the standards. The reason for this is that successful attacks have been constructed against earlier versions of the cryptosystem. In Chapter 8 we briefly present some of these attacks. In Chapter 9 we present our own attack against NTRU. Due to this and similar attacks, NTRU was once again modified. Our attack is published in [31].

Finally, in Chapter 10 we present the related NTRUSign signature protocol. Some vulnerabilities discovered by the author are discussed.

In summary, the goal of this thesis is to provide a unified present-

ation on the NTRU cryptosystem. The original content consists of an attack against one version of NTRU. Tools for the best known lattice reduction algorithms to better fit the NTRU environment are developed; thus giving a more accurate security analysis. Some observations on the NTRUSign signature scheme are also given.

CHAPTER 2

Public key cryptography

The idea of symmetric, or private key, cryptosystems is that the communicating parties agree on a common key which they use to encrypt their messages to each other. The same key is used to decrypt, i.e. to unscramble the encrypted message. Eavesdroppers, who can capture the exchanged encrypted messages, are unable to understand the messages as long as the key used remains secret.

One of the weak points of these private key systems is key agreement. Clearly, before the parties are able to communicate securely, they must have a way to agree on their key. But how is this possible if all exchanged messages can be eavesdropped by adversaries.

The development of complexity theory since the 50's has made it possible to solve the problem in a revolutionary way. The main idea was to split the key; one public key for encryption and one private key for decryption. Party A could then encrypt his/her messages to party B by using B's public encryption key. As B's decryption key is private, only B can decrypt the messages. In theory, the knowledge of the encryption key is sufficent to determine the decryption key. However, the system can be constructed in such a way, that the amount of work required for cryptanalysis is beyond the scope of any realistic adversary.

For a long time, the speed of the best symmetric cryptosystems was superior to all suggested public key cryptosystems. Hence it was not sensible to encrypt large amounts of data with public key systems. Instead, one used public key systems to exchange a key for a fast symmetric system. With the development of computers and algorithms, the public key cryptosystems have been used more and more.

To explain the ideas of complexity theory we must introduce some definitions.

The set $\mathcal{O}(f(n))$ comprises all positive functions $g(n)$ for which there exists constants n_0 and c such that $g(n) \leq cf(n)$ when $n > n_0$.

We denote by $\mathbf{P}$ the set of problems that can be solved in polynomial time using a *deterministic* algorithm. That is, those problems for which there is an integer k such that all instances of size n of the problem can be solved in time $\mathcal{O}(n^k)$. Such problems are called *tractable*; all other problems are called *intractable*. Problems that have a polynomial time *non-deterministic* algorithm constitute the set $\mathbf{NP}$. That is, for every problem in $\mathbf{NP}$ we can check in polynomial time whether a given candidate is a solution to the problem.

Trivially, $\mathbf{P} \subseteq \mathbf{NP}$, but it is not known whether $\mathbf{P} = \mathbf{NP}$.

A problem in $\mathbf{NP}$ is called $\mathbf{NP}$-*complete* if finding a polynomial time algorithm for that problem would mean that there is a polynomial algorithm for all problems in $\mathbf{NP}$ and thus we would have $\mathbf{P} = \mathbf{NP}$. These are clearly the hardest problems in $\mathbf{NP}$. The idea of $\mathbf{NP}$-completeness was introduced by Cook in 1971 [5].

The main building blocks of public key cryptosystems are so-called *one-way functions*. Informally, a function f is called one-way if $f(x)$ can be efficiently computed when x is given, but no efficient algorithm exists to find x such that $f(x) = y$ when y is given. In other words, the problem of computing $f(x)$ is in $\mathbf{P}$, but the problem of inverting f is in $\mathbf{NP} \setminus \mathbf{P}$.

Despite several decades of intense research, it is not known whether one-way functions exist or not. But there are some good candidates which are widely used in practice. As there does not exist a proof of one-wayness, it is possible that somebody someday will find a fast algorithm to invert them.

One-way functions cannot be used as cryptosystems: it is impossible to decrypt. An additional property is needed. A *trapdoor one-way function* is a one-way function which can be inverted in polynomial time if some additional information, the trapdoor, is known. These functions suit public key systems perfectly: the trapdoor acts as the private decryption key. For more detailed definitions consult [12].

The first proposal [30] for a public key cryptosystem by Merkle and Hellman in 1978 was based on a *knapsack problem.* This problem is $\mathbf{NP}$-complete so, at first glance, a cryptosystem based on it seemed sufficiently difficult to break. Unfortunately $\mathbf{NP}$-completeness only means that the hardest instances of the problem are difficult. It turned out that on average the knapsack problem was relatively easy, which made

it unsuitable for cryptographic applications.

The earliest public key system still in use is the RSA cryptosystem developed by Ronald Rivest, Adi Shamir and Leonard Adleman [36]. The underlying hard problem in RSA is the integer factorization problem. It is relatively easy to multiply two large integers; even the "school algorithm" is efficient enough, as it works in time $\mathcal{O}(n^2)$. On the other hand, it is very difficult to determine the factorization of a large integer, especially if it is a product of two large prime numbers. Integers with 300 digits are well beyond the best factorization algorithms known today. Although the factoring problem is not known to be an **NP**-complete problem, it is commonly considered hard enough for cryptographic purposes.

Another popular candidate one-way function is modular exponentiation. Given a, e and n, the value $a^e \mod n$ can be computed in time $\mathcal{O}(n^3)$. For the inverse problem, also known as the discrete logarithm problem, no polynomial time algorithm is known. The famous El Gamal cryptosystem is based on this fact [10]. The same system can also be applied in the elliptic curve group [2].

From a theoretical point of view it would be perfect if breaking a cryptosystem required solving an **NP**-complete problem. However, for practical reasons, *all* instances of the problem that may occur in cryptanalysis should be hard. Therefore a minimal requirement is that the problem in question is hard on average. In [1] Miklos Ajtai showed that certain lattice problems are hard on average, provided they are hard in the worst case. As a consequence, it seems a good idea to base a public key system on such problems.

The NTRU encryption system [15] from 1998 is based on the difficulty of finding a short vector in a lattice, or alternatively the closest lattice vector to a given vector. The main advantage of NTRU is its speed; which is comparable to the fastest symmetric systems.

In addition to ordinary message encryption, public key cryptography also has other applications. From a practical point of view, perhaps the most important one is the ability to sign digital documents. Some properties required from a digital signature are impossible to be fulfilled, or require a so-called trusted third party using symmetric cryptosystems. The idea is that each signer again creates a pair of keys: one private key needed to sign documents, and one public key needed to verify the validity of signatures. There are signature protocols related to most public key cryptosystems, including RSA, El Gamal and NTRU; see [29].

CHAPTER 3

Preliminaries

The NTRU cryptosystem deals with polynomial rings, but is closely linked to lattices. In order to study it, we need to introduce the basic properties of lattices. In this chapter we fix notations, present the basic results and introduce the problems which we will consider later in this work.

The presentation is modified from lecture notes of Cynthia Dwork [7]. We end this chapter by considering some properties of polynomial rings.

3.1 BASIC DEFINITIONS

We start this section by defining the *inner*, or *scalar, product* of two vectors $v = (v_1, v_2, \ldots, v_m)$ and $u = (u_1, u_2, \ldots, u_m)$,

$$(v, u) = \sum_{i=1}^{m} v_i u_i.$$

The inner product is commutative and distributive.

The *Euclidean norm* or the *length* of a vector $v = (v_1, v_2, \ldots, v_m)$ is defined as

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

Let $b_1, b_2, \ldots, b_n$ be linearly independent vectors in $\mathbb{R}^m$ and let B be the $n \times m$ matrix with these vectors as rows. The *lattice* generated by vectors $b_1, b_2, \ldots, b_n$ or, alternatively, the *basis matrix* B, is the set

$$L(b_1, b_2, \ldots, b_n) = L(B) = \left\{ \sum_{i=1}^{n} a_i b_i \mid a_i \in \mathbb{Z} \right\}$$

of all linear combinations over integers of the basis vectors. The *dimension* of this lattice is $\dim(L(B)) = n \leq m$. If $m = n$ the lattice is called *full-dimensional.*

Informally, a lattice is a set of intersection points of a regular, infinite n-dimensional grid.

The *vector space* generated by vectors $b_1, b_2, \ldots, b_n$ or B is

$$\text{span}(b_1, b_2, \ldots, b_n) = \text{span}(B) = \left\{ \sum_{i=1}^{n} a_i b_i \mid a_i \in \mathbb{R} \right\},$$

where we have all linear combinations of the basis vectors.

We note that if B' is the result of applying any of the following operations to B then $L(B) = L(B')$:

1. Swap the order of two rows in B.

2. Multiply a row of B by -1.

3. Add an integer multiple of a row to another row of B.

The first two cases are trivial. Also trivially $L(B') \subseteq L(B)$ in the third case. Let $b'_j = b_j + cb_k$, $j \neq k$ and $b'_i = b_i$, for all $i \neq j$. Now for all $a_i \in \mathbb{Z}$, $\sum a_i b_i = \sum_{i \neq k} a_i b'_i + (a_k - a_j c) b'_k$ and thus $L(B) \subseteq L(B')$.

The *determinant of a lattice* $L(B)$ with basis $b_1, \ldots, b_n$ is defined as

$$\det(L(B)) = \sqrt{\det((b_i, b_j)_{1 \leq i,j \leq n})} = \sqrt{\det(BB^T)}.$$

If $n = m$ we have

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

Because basis vectors are linearly independent, two bases of the same lattice have the same number of vectors. In the following we show that the determinant of a lattice does not depend on the selection of the basis.

THEOREM 1 *Let B and B' be $n \times m$ real matrices and $L(B) = L(B')$. We have* $\det(L(B)) = \det(L(B'))$.

Proof. The rows of both matrices are two bases for the same lattice so we have $B = UB'$ and $B' = VB$ for some $U, V \in \mathbb{Z}^{n \times n}$. From these we

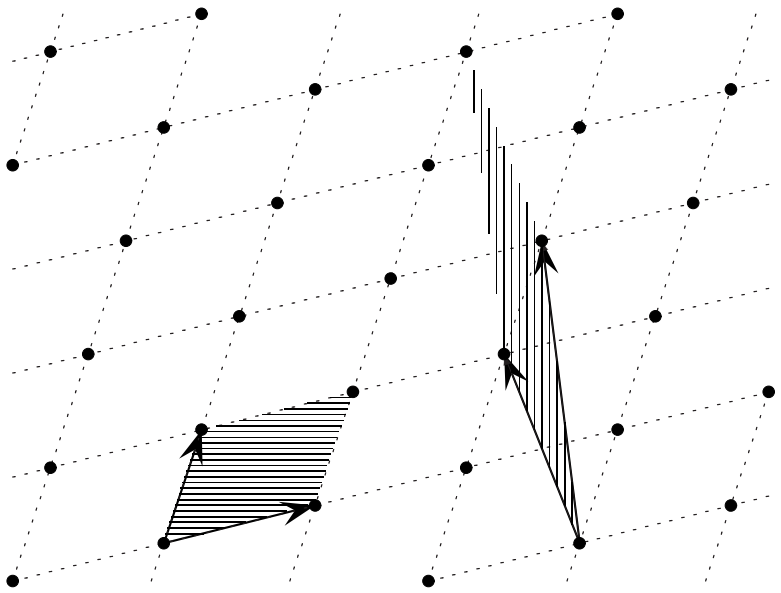


FIGURE 3.1: A lattice of dimension 2 with two bases and two fundamental parallelepipeds. A fundamental parallelepiped forms a partitioning of the space.

obtain $\det(U)\det(V) = 1$. Because $\det(U), \det(V) \in \mathbb{Z}$, it follows that $|\det(U)| = |\det(V)| = 1$. We have

$$\begin{aligned}\det(L(B)) &= \sqrt{\det(BB^T)} \\ &= \sqrt{\det(VBB^TV^T)} \\ &= \sqrt{\det(VB(VB)^T))} \\ &= \sqrt{\det(B'B'^T)} \\ &= \det(L(B'))\end{aligned}$$

■

The *fundamental parallelepiped* associated with B is the set of points

$$P(B) = \left\{ \sum_{i=1}^{n} a_i b_i \mid 0 \leq a_i < 1 \right\}.$$

We see that $v + P(B)$, $v \in L(B)$, form a partition of the space $\text{span}(B)$. In other words, for any $u \in \text{span}(B)$ there exists a unique

lattice point $v \in L(B)$ such that $u \in v + P(B)$. If $L = L(B)$ we also write $P(L) = P(B)$.

Two vectors v and u are called *orthogonal* if $(v, u) = 0$. If S is a space then we denote by $S^{\perp}$ the *orthogonal space* such that every vector of S is orthogonal to every vector of $S^{\perp}$.

For a lattice L with basis $b_1, b_2, \ldots, b_n \in \mathbb{Z}^m$ we can calculate the *Gram-Schmidt* orthogonalization:

$$\begin{aligned} \hat{b}_1 &= b_1 \\ \hat{b}_j &= b_j - \sum_{i=1}^{j-1} \mu_{j,i} \hat{b}_i, \quad j = 2, \ldots, n \end{aligned} \tag{3.1}$$

where

$$\mu_{j,i} = \frac{(b_j, \hat{b}_i)}{(\hat{b}_i, \hat{b}_i)}. \tag{3.2}$$

Note that vectors $\hat{b}_j$ do not necessarily belong to the lattice.

It is easy to see that vectors $\hat{b}_j$ are now orthogonal to each other: Let us assume that vectors $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{k-1}$ are orthogonal, then

$$\begin{aligned} (\hat{b}_k, \hat{b}_j) &= (b_k, \hat{b}_j) - \sum_{i=1}^{k-1} \mu_{k,i} (\hat{b}_i, \hat{b}_j) \\ &= (b_k, \hat{b}_j) - \frac{(b_k, \hat{b}_j)}{(\hat{b}_j, \hat{b}_j)} (\hat{b}_j, \hat{b}_j) = 0 \end{aligned}$$

for all $j < k$.

The Gram-Schmidt orthogonalization process can also be described using matrices:

$$\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} 1 & & & \\ & 1 & 0 & \\ & \mu_{j,i} & \ddots & \\ & & & 1 \end{pmatrix} \begin{pmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \vdots \\ \hat{b}_n \end{pmatrix}.$$

If we define $\mu_{i,i} = 1$ and $\mu_{j,i} = 0$ when $i > j$ we can write the equation above as

$$B = (\mu_{k,j})_{1 \le j,k \le n} \hat{B}.$$

Clearly $\det((\mu_{k,j})_{1\leq j,k\leq n}) = 1$. We have

$$\begin{aligned}\det(L(B)) &= \sqrt{\det(BB^T)} \\ &= \sqrt{\det((\mu_{k,j})_{1\leq j,k\leq n}\hat{B}((\mu_{k,j})_{1\leq j,k\leq n}\hat{B})^T)} \\ &= \sqrt{\det((\mu_{k,j})_{1\leq j,k\leq n}\hat{B}\hat{B}^T(\mu_{k,j})^T_{1\leq j,k\leq n})} \\ &= \sqrt{\det(\hat{B}\hat{B}^T)} \\ &= \prod_{j=1}^{n}||\hat{b}_j||.\end{aligned}$$

Let us consider the two dimensional parallelogram with sides b_1 and b_2. If we name b_1 the base then the height of the parallelogram is $||\hat{b}_2||$. Thus the area is $||b_1||||\hat{b}_2|| = ||\hat{b}_1||||\hat{b}_2||$. Next consider the three dimensional parallelepiped spanned by b_1, b_2, and b_3. The base is the parallelogram with sides b_1 and b_2 and its area is $||\hat{b}_1||||\hat{b}_2||$. The height of the parallelepiped is $||\hat{b}_3||$ and thus the volume is $||\hat{b}_1||||\hat{b}_2||||\hat{b}_3||$. In general we obtain

$$\text{vol}(P(B)) = \prod_{i=1}^{n}||\hat{b}_i||.$$

We obtained the following theorem:

THEOREM 2 *The volume of the fundamental parallelepiped equals the determinant of the lattice.*

REMARK 3.1 When we talk about the volume we mean the natural idea of volume. If we wanted to be more formal, we could consider it as the Lebesgue measure.

3.2 THEOREMS OF BLICHFELDT AND MINKOWSKI

The following theorem is due to Blichfeldt (1914). We use it as a lemma for the more interesting Minkowski's theorem. The more general theorems can be found in [4].

THEOREM 3 *Let k be a positive integer, L a full-dimensional lattice in $\mathbb{R}^n$ and $S \subset \mathbb{R}^n$. If $\text{vol}(S) > k\det(L)$ then there exist $k+1$ distinct points $x_1, x_2, \ldots, x_{k+1}$ such that the differences $x_i - x_j$ are all in L.*

Proof. For each $u \in L$ we denote

$$R(u) = \{v \mid v \in P(L), u + v \in S\}.$$

For any lattice point u, $R(u)$ is the set of relative locations of points of S that lie in the fundamental parallelepiped drawn at u. These are pairwise disjoint sets and

$$S = \bigcup_{u \in L} u + R(u).$$

Thus

$$\mathrm{vol}(S) = \sum_{u \in L} \mathrm{vol}(R(u)).$$

If $\mathrm{vol}(S) > k \det(L)$ we obtain

$$\sum_{u \in L} \mathrm{vol}(R(u)) > k \det(L) = k \,\mathrm{vol}(P(L)).$$

All sets $R(u)$ are contained in the set $P(L)$. By the pigeon hole principle, the sum of the volumes of $R(u)$ cannot be larger than $k \,\mathrm{vol}(P(L))$ unless some point of $P(L)$, say v_0, belongs to at least $k+1$ sets $R(u)$, say

$$v_0 \in R(u_j), \quad 1 \le j \le k+1,$$

where the lattice points u_j are distinct. Now the points

$$x_j = u_j + v_0$$

are in S by the definition of $R(u)$ and

$$x_i - x_j = u_i - u_j \in L \setminus 0,$$

when $i \neq j$. ■

We call a set S *symmetric* if it is symmetric with respect to the origin,

$$x \in S \Rightarrow -x \in S,$$

and *convex* if all points between two points of the set also belong to the set,

$$x, y \in S \Rightarrow \lambda x + (1-\lambda) y \in S,$$

for all $0 \le \lambda \le 1$.

The following is a part of the so-called Minkowski's convex body theorem.

THEOREM 4 *Let $S \subset \mathbb{R}^n$ be a symmetric and convex set. Let k be a positive integer and let L be a full-dimensional lattice in $\mathbb{R}^n$. If $\mathrm{vol}(S) > k2^n \det(L)$ then S contains at least k pairs of points $\pm u_j \in L \backslash 0$, $1 \leq j \leq k$, which are distinct from each other.*

Proof. Let us assume that $\mathrm{vol}(S) > k2^n \det(L)$ and consider the set Q,

$$Q = \{v \mid 2v \in S\}.$$

This set has volume satisfying the inequality $\mathrm{vol}(Q) > k \det(L)$. The previous theorem tells us that there exist $k+1$ distinct points $x_j \in Q$, $1 \leq j \leq k+1$, such that $x_i - x_j \in L$. We write $x_j > x_i$ if the first non-zero component of $x_j - x_i$ is positive. We re-order the points x_j such that $x_j > x_{j+1}$ always. Now $x_j > x_i$ for any $i > j$. Let

$$u_j = x_j - x_{k+1}.$$

These points u_j are in L. If $u_i = u_j$ then $x_i - x_{k+1} = x_j - x_{k+1}$ and thus $i = j$. If $u_i = -u_j$ then $x_i - x_{k+1} = -(x_j - x_{k+1})$ and either $x_{k+1} > x_i$ or $x_{k+1} > x_j$, against our ordering. Thus the points

$$0, \pm u_1, \ldots, \pm u_k$$

are all distinct. Because $x_j \in Q$, we have $2x_j \in S$ and, due to symmetry, $-2x_j \in S$ for all $1 \leq j \leq k-1$. Hence, by convexity and the definition of the u_j, we have

$$u_j = \frac{1}{2} 2x_j + \frac{1}{2}(-2x_{k+1}) \in S.$$

The same is true for $-u_j$. ■

Let us consider an n-dimensional hypercube with sides of length a that is centered at the origin. This hypercube has volume a^n and is inside a ball of radius $\sqrt{\left(\frac{a}{2}\right)^2 n}$. Using this hypercube as set S with $a = 2\sqrt[n]{\det(B)}$ we obtain the following corollary:

COROLLARY 1 *For any full-dimensional lattice $L(B)$ in $\mathbb{R}^n$, there exists a lattice point $x \in L(B) \setminus 0$ such that*

$$||x|| \leq \sqrt{n}\sqrt[n]{\det(B)}.$$

Thus we have an upper limit for the length of the shortest non-zero vector of $L(B)$.

3.3 THE SHORTEST VECTOR PROBLEM

The *Gamma function*, see [24], is defined as

$$\Gamma(n) = 2\int_0^\infty e^{-r^2} r^{2n-1} dr.$$

It has, for example, properties $\Gamma(n) = (n-1)\Gamma(n-1)$ and $\Gamma(\frac{1}{2}) = \sqrt{\pi}$.

THEOREM 5 *An n-dimensional ball with radius R has volume*

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

Proof. We denote by S_n the surface area of the n-dimensional unit ball. The surface area of an n-dimensional ball with radius r is $S_n r^{n-1}$. We obtain the volume of a ball with radius R by integrating over the surface area of all balls with radius smaller than R,

$$V_n(R) = S_n \int_0^R r^{n-1} dr = \frac{S_n R^n}{n}. \tag{3.3}$$

We can write the integral $\int_{\mathbb{R}^n} e^{-|x|^2}$ in two ways, using the coordinate representation of x or the fact that the value of $e^{-|x|^2}$ is constant on the surface of an origin centered ball. We have

$$\begin{aligned} S_n \int_0^\infty e^{-r^2} r^{n-1} dr &= \int_{\mathbb{R}^n} e^{-(x_1^2+\cdots+x_n^2)} dx_1 \cdots dx_n \\ &= \left(\int_{-\infty}^\infty e^{-x^2} dx\right)^n \\ &= \left(2\int_0^\infty e^{-x^2} dx\right)^n. \end{aligned}$$

Using the Gamma function we can write this as

$$\tfrac{1}{2} S_n \Gamma(\tfrac{1}{2}n) = \Gamma(\tfrac{1}{2})^n = (\sqrt{\pi})^n$$

and we obtain

$$S_n = \frac{2\pi^{\frac{n}{2}}}{\Gamma(\frac{1}{2}n)}.$$

We substitute this into (3.3) and obtain

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

■

Stirling's approximation for the Gamma function gives us

$$\Gamma(1+n) \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n.$$

Therefore we obtain

$$V_n(R) \approx \frac{\pi^{\frac{n}{2}} R^n}{\sqrt{\pi n}\left(\frac{n}{2e}\right)^{\frac{n}{2}}}.$$

From this we can solve R,

$$R \approx (\pi n)^{\frac{1}{2n}} \sqrt{\frac{n}{2\pi e}} V_n(R)^{\frac{1}{n}}. \tag{3.4}$$

Let $L(B)$ be a full dimensional lattice. We consider a ball with its center at the origin and with volume $2^n|\det(B)|$. The ball has radius

$$R \approx (\pi n)^{\frac{1}{2n}} \sqrt{\frac{n}{2\pi e}} (2^n|\det(B)|)^{\frac{1}{n}} = (\pi n)^{\frac{1}{2n}} \sqrt{\frac{2n}{\pi e}} |\det(B)|^{\frac{1}{n}}.$$

Using this aproximation we obtain the following corollary from Theorem 4.

COROLLARY 2 *For any full-dimensional lattice $L(B)$ in $\mathbb{R}^n$ there exists a lattice point $x \in L(B) \setminus 0$ such that $||x||$ is at most*

$$(\pi n)^{\frac{1}{2n}} \sqrt{\frac{2n}{\pi e}} |\det(B)|^{\frac{1}{n}}.$$

The *Gaussian heuristic* says that we can estimate for a lattice L the number of lattice points that lie inside some subset S of $\mathrm{span}(L)$ as

$$\frac{\mathrm{vol}(S)}{\det(L)}.$$

We obtain that for some vector $v \in \mathrm{span}(B)$ the distance of v from the closest point of L is approximately equal to the radius of an n-dimensional ball with volume equal to the volume of the fundamental parallelepiped of the lattice. The intuition here is that the point v always lies inside a fundamental parallelepiped that is centered at a lattice point. Thus it would also lie inside any nicely sharped region of the same volume, a ball, for example. We call the radius R of this ball the *critical radius* and have, by (3.4),

$$R \approx (\pi n)^{\frac{1}{2n}} \sqrt{\frac{n}{2\pi e}} \det(L(B))^{\frac{1}{n}} \approx \sqrt{\frac{n}{2\pi e}} \det(L(B))^{\frac{1}{n}}$$

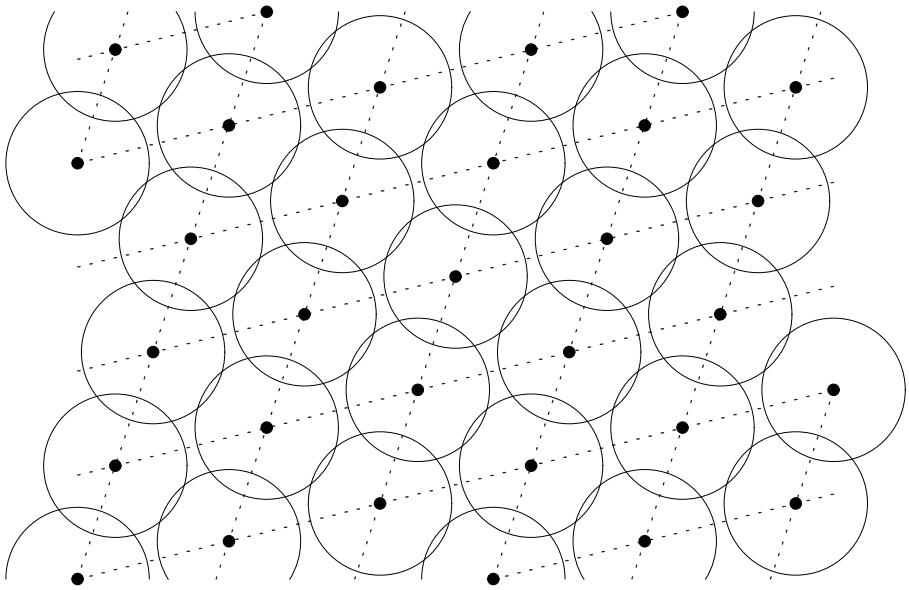


FIGURE 3.2: Gaussian heuristic.

when n is large.

For every lattice L of dimension n we define the *successive minima* $\lambda_1, \ldots, \lambda_n$ as

$$\lambda_i = \lambda_i(L) = \inf \left\{ \max_{j=1,\ldots,i} (||x_j||) \;\middle|\; x_j \in L \text{ linearly independent} \right\}$$

for $i = 1, \ldots, n$. We see that always

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

In the lattices considered in this work, there is always at least one point in L at distance λ_1 from the origin: the shortest vectors of the lattice. It is possible that there are k linearly independent points at distance λ_1 from the origin. In this case $\lambda_1 = \lambda_2 = \cdots = \lambda_k$.

Because the basis vectors $b_1, \ldots, b_n$ are always linearly independent, we have for all $1 \leq i \leq n$,

$$\max_{j=1,\ldots,i} (||b_j||) \geq \lambda_i.$$

It is possible to find a basis of L such that all of the basis vectors are of length at most λ_n.

We define the *Hermite constants* γ_n as

$$\gamma_n = \sup \left\{ \frac{\lambda_1(L)^2}{\det(L)^{\frac{2}{n}}} \,\middle|\, L \subset \mathbb{R}^n \text{ is a full-dimensional lattice} \right\}.$$

For example $\gamma_3 = 2$. This means that any three dimensional lattice with determinant D has a point that is at most of distance $\sqrt{2}D^{\frac{1}{3}}$ from the origin.

From corollary 1 we obtain that $\gamma_n \leq n$. As corollary 2 indicates, this is not a very strict upper limit.

Here we are interested in finding points of lattice that are as close as possible to some special point in space. We state two different problems.

Shortest vector problem (SVP): find the shortest non-zero vector v in a lattice L with basis $b_1, \ldots b_n$,

$$v = k_1 b_1 + k_2 b_2 + \ldots + k_n b_n,$$

where $(k_1, k_2, \ldots, k_n) \in \mathbb{Z}^n \setminus 0^n$.

In other words the shortest vector problem means that we need to find a vector v in the lattice with $||v|| = \lambda_1$. We defined λ_1 as an infinum but we can easily see that such a vector exists when basis vectors $b_1, \ldots, b_n$ have integer elements. The length of every lattice vector is a square root of a natural number and $\lambda_1 \leq ||b_1||$. Thus there are only finitely many possibilities for the length of the shortest vector.

Closest vector problem (CVP): find the vector closest to $a \in \mathbb{Z}^m$ in a lattice L.

A CVP is related to a SVP of dimension $n+1$. Let $b_1, b_2, \ldots, b_n$ be the basis of L and let

$$b_i = (b_{i1}, b_{i2}, \ldots, b_{im})$$

and

$$a = (a_1, a_2, \ldots, a_m).$$

We construct a new lattice L' which has basis $b'_1, b'_2, \ldots, b'_n, a' \in \mathbb{Z}^{m+1}$ where

$$b'_i = (b_{i1}, b_{i2}, \ldots, b_{im}, 0)$$

and

$$a' = (a_1, a_2, \ldots, a_m, 1).$$

If a lattice point

$$c = c_1 b_1 + c_2 b_2 + \cdots + c_n b_n$$

is close to a vector a then $a - c$ is a short vector and there exists a short vector

$$a' - c_1 b'_1 - c_2 b'_2 - \cdots - c_n b'_n$$

in the lattice L'. However, this is not necessarily the shortest vector of L'. For example a' may be shorter, or it may be that

$$2a' - c_1 b'_1 - c_2 b'_2 - \cdots - c_n b'_n$$

is very short for some $c_i \in \mathbb{Z}$. For more on the relationship between CVP and SVP; see [7].

3.4 POLYNOMIAL RINGS

We consider some properties of a polynomial ring $\mathbb{Z}[x]/(x^N - 1)$. The elements of this ring are polynomials of degree at most $N-1$ with integer coefficients. Ring addition is performed componentwise and the product is the *convolution product* of polynomials: for $a = a_0 + a_1 x + \cdots + a_{N-1} x^{N-1}$ and $b = b_0 + b_1 x + \cdots + b_{N-1} x^{N-1}$ this is calculated as

$$a * b = \sum_{i=0}^{N-1} \left(\sum_{j+k \equiv i \ (N)} a_j b_k x^i \right).$$

For example,

$$(1 + 2x^2 + x^5) * (x^2 + 3x^4) = x^2 + 5x^4 + 6x^6 + x^7 + 3x^9 = 1 + 4x^2 + 5x^4 + 6x^6$$

in the polynomial ring $\mathbb{Z}[x]/(x^7 - 1)$.

A polynomial ring $\mathbb{Z}_q[x]/(x^N - 1)$ has the same operations as the ring $\mathbb{Z}[x]/(x^N - 1)$ except that every coefficient is reduced modulo q. In this ring some elements have an inverse.

The *inverse* of a polynomial a in $\mathbb{Z}_q[x]/(x^N - 1)$ is a polynomial $a^{-1} \in \mathbb{Z}_q[x]/(x^N - 1)$ such that $a * a^{-1} = a^{-1} * a = 1$.

If q is a prime and $\mathbb{Z}_q$ is a field, then we find the possible inverse of a polynomial a using the Extended Euclidean Algorithm. If we have that $\gcd(a, x^N - 1) = 1$ then the inverse exists and the algorithm gives us polynomials $u, v \in \mathbb{Z}_q[x]/(x^N - 1)$ such that

$$u * a + v * (x^N - 1) = 1$$

in $\mathbb{Z}_q[x]$. This means that $u * a = 1$ in $\mathbb{Z}_q[x]/(x^N - 1)$.

The complexity of the Extended Euclidean Algorithm is $\mathcal{O}(N^2 \log(q^2))$ using the school arithmetic; see [29].

If q is a power of a prime p, $q = p^t$, the situation is a little more complicated. Using the Extended Euclidean Algorithm we get polynomials $u, v, c \in \mathbb{Z}_q[x]/(x^N - 1)$ such that

$$u * a + v * (x^N - 1) = 1 - pc$$

in $\mathbb{Z}_q[x]$. This means that $u * a = 1 - pc$ in $\mathbb{Z}_q[x]/(x^N - 1)$. We also have

$$\begin{aligned}
(1 + pc) * u * a &= 1 - p^2c^2 \\
(1 + p^2c^2) * (1 + pc) * u * a &= 1 - p^4c^4 \\
&\vdots \\
(1 + p^{2^{s-1}}c^{2^{s-1}}) * \cdots * (1 + p^2c^2) * (1 + pc) * u * a &= 1 - p^{2^s}c^{2^s}
\end{aligned}$$

in $\mathbb{Z}_q[x]/(x^N - 1)$. If $2^s \geq t$ we have

$$(1 + p^{2^{s-1}}c^{2^{s-1}}) * \cdots * (1 + p^2c^2) * (1 + pc) * u * a \equiv 1 \mod q$$

and thus

$$a^{-1} = (1 + p^{2^{s-1}}c^{2^{s-1}}) * \cdots * (1 + p^2c^2) * (1 + pc) * u.$$

If q has at least two prime factors, we apply the process above separately to each prime factor. The inverse is then found using The Chinese Remainder Theorem.

In the following we find the conditions under which the inverse exists when $q = 239$ and $N = 251$. We begin by listing two results for finite fields without proofs. The first one is about the factorization of $x^n - 1$.

Let $\gcd(n, p) = 1$. We define the *cyclotomic coset modulo n* containing s as

$$C_s = \{s, ps, p^2s, \ldots, p^{r_s - 1}s\},$$

where each $p^i s$ is reduced modulo n. Here r_s is the smallest positive integer such that $p^{r_s}s \equiv s \mod n$.

Let $\mathbb{F}_{p^r}$ be the smallest field of characteristic p that contains all of the of roots of $x^n - 1$. The following is Theorem 4.11 from [26]:

THEOREM 6 *Let α be a root of $x^n - 1 = 0$ in $\mathbb{F}_{p^r}$ and let m be its minimal polynomial. Let ζ be a primitive root of unity in $\mathbb{F}_{p^r}$ and*

let $\alpha = \zeta^i$. If s is the smallest element in the cyclotomic coset mod n containing i, then

$$m = \prod_{j \in C_s} (x - \zeta^j).$$

The roots of $x^n - 1$ are powers of a primitive root of unity,

$$x^n - 1 = \prod_{i=0}^{n-1} (x - \zeta^i).$$

Let us denote by $m^{(s)}$ the minimal polynomial of ζ^s. The above theorem implies that

$$x^n - 1 = \prod_s m^{(s)},$$

where s runs through a set of coset representatives modulo n. This is the factorization of $x^n - 1$ into irreducible polynomials over $\mathbb{F}_p$.

Another result we use is The Chinese Remainder Theorem for Polynomials, Theorem 4.1 in [26]:

THEOREM 7 *Let $f_1, f_2, \ldots, f_r$ be distinct and irreducible polynomials over $\mathbb{F}_q$ and let $g_1, g_2, \ldots, g_r$ be arbitrary polynomials over $\mathbb{F}_q$. Then the system of congruences $h \equiv g_i \mod f_i$, $i = 1, 2, \ldots, r$, has a unique solution h modulo $f_1 f_2 \cdots f_r$.*

Let us now consider the case of $\mathbb{Z}_{239}[x]/(x^{251} - 1)$. Because $239^{50} \not\equiv 1 \mod 250$ and $239^{125} \not\equiv 1 \mod 250$, we see that $239^i \not\equiv 1 \mod 250$ for all $0 < i < 250$. Thus the set C_1 has 250 elements and $m^{(1)}$ has degree 250. The polynomial $x^{251} - 1$ has two irreducible factors, $x - 1$ and $m^{(1)} = x^{250} + x^{249} + \cdots + 1 = \varphi(x)$.

Let us assume that neither $x - 1$ nor $\varphi(x)$ divides a polynomial $a \in \mathbb{Z}_{239}[x]$. With the Extended Euclidean Algorithm we can find polynomials A_1 and A_2 in $\mathbb{Z}_{239}[x]$ such that

$$\begin{cases} aA_1 \equiv 1 \mod x - 1, \\ aA_2 \equiv 1 \mod \varphi(x). \end{cases}$$

Theorem 7 states that there exists a polynomial A such that

$$\begin{cases} A \equiv A_1 \mod x - 1, \\ A \equiv A_2 \mod \varphi(x). \end{cases}$$

This means that

$$\begin{cases} aA = 1 + B(x-1), \\ aA = 1 + C\varphi(x) \end{cases}$$

for some polynomials B and C. Furthermore,

$$aA = 1 + D(x-1)\varphi(x)$$

for some polynomial D and thus A is the inverse of a in $\mathbb{Z}_{239}[x]/(x^{251}-1)$.

Conversely, let us assume that a has an inverse A in $\mathbb{Z}_{239}[x]/(x^{251}-1)$. We can write

$$aA = 1 + D(x-1)\varphi(x)$$

for some polynomial D. Thus neither $x-1$ nor $\varphi(x)$ divides a.

We have shown:

THEOREM 8 *A polynomial $a \in \mathbb{Z}_{239}[x]/(x^{251}-1)$ has an inverse if and only if $a(1) \not\equiv 0 \mod 239$ and not all coefficients of a are equal.*

We need this theorem later when we generate the keys of NTRUEncrypt.

CHAPTER 4

NTRUEncrypt

NTRU is a public key cryptosystem proposed by J. Hoffstein, J. Pipher and J. Silverman. The first version of the NTRU encryption system was presented at the Crypto '96 conference; see [15]. The mathematical basis of these systems lies in polynomial algebra, the fundamental tool being the reduction of polynomials with respect to two different moduli. It is the efficiency of NTRU that makes it a potential practical system. It is significantly faster than its main rivals RSA and ECC (or any other public key system). Moreover, the computations are very simple, which makes it suitable for devices with restricted resources, such as smart cards. On the other hand, the security of these systems is still somewhat questionable. This is partly due to the relatively short time (so far) spent studying it. Even more importantly, the NTRU signature scheme has been broken and subsequently redesigned several times during its existence.

4.1 SYSTEM DESCRIPTIONS

The NTRU encryption system and the related signature scheme are both built on polynomial algebra. The basic objects are truncated polynomials in the ring $\mathbf{R} = \mathbb{Z}[x]/(x^N - 1)$ and the basic tool is the reduction of polynomials with respect to two relatively prime moduli. The security of the systems is (hoped to be) based on the difficulty of finding a "short" factorization for such polynomials. This latter problem is equivalent to finding a short vector in a certain $2N$ dimensional lattice, a commonly known and also widely studied hard problem.

4.2 OVERVIEW

NTRU polynomials $a(x)$ are frequently reduced modulo p and q, the small and large moduli. The large modulus q is an integer, so reduction of $a(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{N-1}x^{N-1} \mod q$ means just reduction of each a_i modulo q. The small modulus p can also be an integer, but in some old versions of the NTRU cryptosystem $p = 2 + x$. In this case reduction modulo p requires a somewhat more complex algorithm. It is required that p and q are relatively prime: $\gcd(p, q) = 1$.

The main objects in the systems are "small" polynomials; i.e. polynomials with small coefficients, or polynomials with a small norm (Euclidean length of the coefficient vector).

The public key h is defined by an equation $f * h = p * g \mod q$, where f and g are small polynomials. The polynomial f should always have inverses modulo p and q,

$$f * f_p \equiv 1 \mod p \quad \text{and} \quad f * f_q \equiv 1 \mod q.$$

Moreover, the parameters N, p and q are also public, and can be used as common domain parameters for all users. Polynomials f and g are private to the key owner. The polynomial g is needed only in key generation.

REMARK 4.1 For a polynomial a, $a(1)$ is the sum of the coefficients of a and therefore it reveals the average of these coefficients. If all coefficients are positive, $a(1)$ is a good indicator of the "smallness" of a.

ENCRYPTION

NTRU is a probabilistic public key cryptosystem, hence one plaintext message has several possible encryptions.

Encryption of message m is performed by first selecting a *message representative* i and a *blinding polynomial* r, and then computing the cryptotext

$$e \equiv i + r * h \mod q.$$

The selection of i and r is performed by first choosing some random data b and then computing $i = \varphi(m, h, b)$ and $r = \rho(m, b)$. Without going into the details yet, both will be polynomials with very small coefficients. The function φ is invertible: m and b can be computed given $i = \varphi(m, h, b)$.

DECRYPTION

Decryption is a bit cumbersome, as it does not always succeed. First the cryptotext e is processed to obtain a candidate message representative i', which is then tested to see whether it is valid. If it is not valid, new candidates are generated until the correct one is found. It is also possible that decryption fails completely, although the encryption is performed as intended.

The first task is to compute $a \equiv f * e \mod q$. The coefficients of a are reduced to an interval $[A, A+q)$, where A is a so-called *average decryption coefficient*. A candidate message representative is obtained by calculating $i' \equiv a * f_p \mod p$. Then candidates m' and b' are computed by reversing φ, and from these one gets the candidate blinding polynomial $r' = \rho(m', b')$. Now the decryptor re-encrypts m', $e' \equiv i' + r' * h \mod q$, and if $e = e'$ then $m = m'$ and decryption is finished. If $e \neq e'$ then the value of A is modified and the process is repeated.

The decryption (usually) works, because we always have that $a \equiv \alpha \mod q$, where

$$\alpha = f * i + p * r * g.$$

Clearly, because $f * f_p \equiv 1 \mod p$, reduction of $\alpha * f_p$ modulo p gives the correct i. Hence, if α is known exactly (not only modulo q), decryption succeeds. Because f, i, p, r and g are all small polynomials, the coefficients of α most likely lie in an interval of length less than q. If this is the case, α is obtained by reducing the coefficients of a to a proper interval. The question is how to find the proper interval?

The candidate for the proper interval is first selected as $[A, A+q)$, where A is called the average coefficient of α, even though the actual value for the expected average coefficient of α is $A + q/2$. This is easily obtained if $\alpha(1)$ is known. We know that $\alpha(1) = f(1)i(1) + r(1)p(1)g(1)$, where only $i(1)$ is unknown. But $\alpha(1) \equiv a(1) \mod q$, so we can compute $i(1) \mod q$. As i is a random binary polynomial, it is very probable that $N/2 - q/2 \leq i(1) < N/2 + q/2$. Using this assumption, we get the average decryption coefficient A and the first guess on the proper interval.

If i' turns out to be invalid, we must assume that a *wrap failure* has occurred; i.e. we have selected a wrong interval. To correct this, the value of A is modified and the process is repeated. If the modifications of A do not help, two alternatives remain. Either the coefficients of α do not lie in any interval of length q (this situation is called a *gap failure*), or the cryptotext we started with was not a valid cryptotext. In these

cases there is no option but to quit (and possibly to send the encryptor a request to send a new encryption).

REMARK 4.2 There are at least two reasons to adopt the "check back" decryption algorithm described above. Decryption always produces some candidate message representative i'. If a wrap or gap failure has occurred, then i' is incorrect. Using the method above, one can check whether this is the case. Secondly, this method guarantees that the encryptor can create a valid cryptotext e only if (s)he knows the corresponding plaintext m. This *plaintext awareness* property is advantageous from the point of view of hindering some of the most powerful cryptographic attacks, such as adaptive chosen cryptotext attacks [3]. Specifically, it is a valid counter-measure against the reaction attack presented in [22].

4.3 THE CURRENT VERSION OF NTRUENCRYPT

At the moment there are two parameter sets in the standard [9], but others can be added later. Some sets are designed for speed, others for better security. The parameter set ees251ep4 of [9] states that

- Degree parameter $N = 251$,
- Large modulus $q = 239$,
- Small modulus $p = 2$, and
- The number of coefficients equal to 1 in polynomials F, g, and r is $d_f = 72$, $d_g = 72$, and $d_r = 72$, respectively.

The hash function is SHA-1. A *Hash function* is a one-way function that transforms a string of any length into a string of fixed length.

The recent paper [20] by the researchers at NTRU presents a method for choosing parameters for different security levels.

KEYS AND KEY GENERATION

The private key is a polynomial $f \in \mathbf{R}$. It is generated as follows: Let F be a polynomial in $\mathbf{R}$ with d_f randomly chosen coefficients set to 1 and the rest set to 0. Now $f = 1 + 2F$. We also require that there exists a polynomial $f^{-1} \in \mathbf{R}$ such that $f * f^{-1} \equiv 1 \mod q$.

We need another polynomial $g \in \mathbf{R}$. Just like F, it has d_g randomly chosen coefficients set to 1 and the rest set to 0. We also require that there exists a polynomial g^{-1} such that $g * g^{-1} \equiv 1 \mod q$. This polynomial is only needed for the generation of the public key. However this polynomial must be kept secret (or forgotten) because it, with the public key, gives the secret key.

The public key h is the polynomial in the ring $\mathbb{Z}_q[x]/(x^N - 1)$ such that $f * h \equiv 2g \mod q$. This polynomial could also be defined as $h \equiv f^{-1} * 2g \mod q$. Since we require f^{-1} to exist, the polynomial h also exists.

REMARK 4.3 The private key f can be presented as a bit string s where $s_j = 1$ if and only if f_{j-1} in f is 2 or 3. Only one coefficient, f_0, is odd. It is possible to specify f using N bits.

The public key h needs more storage space. Every coefficient h_j is in the range $[0, q-1]$ or, alternatively, in $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$. The size of the key is about N times $\log_2 q$ bits, 8 times greater than the size of the private key when $q = 239$.

REMARK 4.4 If the polynomial f has an inverse, f^{-1}, such that $f * f^{-1} \equiv 1 \mod q$, then we can find this inverse using the Extended Euclidean algorithm. Theorem 8 states that, because $f(1) = 145$, f^{-1} always exists. Also g^{-1} always exists because $g(1) = 72$.

REMARK 4.5 Because f^{-1} always exists, there are exactly

$$\binom{251}{72} \approx 1.19 \times 10^{64}$$

different private keys f with the parameter set ees251ep4. This is also the number of different polynomials g.

THEOREM 9 *If q is odd and $2\min(d_f, d_g) + 1 < q$, then no two different private keys correspond to the same public key.*

Proof. Let f and f' be two different private keys and g and g' two binary polynomials in $\mathbf{R}$ such that $g(1) = g'(1) = d_g$ and

$$f^{-1} * 2g \equiv f'^{-1} * 2g' \mod q.$$

We multiply the above by $f * f'$ and get

$$\begin{aligned} 2f' * g &\equiv 2f * g' \mod q \\ 2(1 + 2F) * g &\equiv 2(1 + 2F') * g' \mod q \\ 2g + 4F * g &\equiv 2g' + 4F' * g' \mod q. \end{aligned}$$

Because $F(1) = F'(1) = d_f$ and $g(1) = g'(1) = d_g$, the coefficients of the polynomials on both sides of the above equation are non-negative, even and smaller than $2 + 4\min(d_f, d_g) < 2q$. We have

$$\begin{aligned} 2g + 4F * g &= 2g' + 4F' * g' \\ 2g &\equiv 2g' \mod 4 \\ g &= g'. \end{aligned}$$

We required that g^{-1} exists and thus

$$\begin{aligned} 2f' * g &\equiv 2f * g \mod q \\ 2f' &\equiv 2f \mod q \\ f' &= f. \end{aligned}$$

We have shown that the private keys must be the same and so there do not exist two different private keys that correspond to the same public key. ■

REMARK 4.6 From this theorem and the previous remark we see that we can have exactly

$$\binom{251}{72}^2 \approx 1.40 \times 10^{128}$$

different possible public keys h using the parameter set ees251ep4.

ENCRYPTION

For the sake of simplicity, we assume in the following that the parameter set ees251ep4 is used.

Let m be the binary message which we would like to encrypt. The length of m must be at most 160 bits when $N = 251$. We select 80 bits of random data b. We construct a bit string $M \in \{0,1\}^{251}$ as follows:

$$M = b \mid l \mid m \mid \bar{0},$$

where l is the length of m presented as a binary number of length 8 and $\bar{0}$ is a sequence of zeroes such that the length of M is 251 bits.

Let $hTrunc$ be a sequence 80 bits in length calculated from the public key h and let OID be a 24 bit constant string defined by the standard. The binary string

$$z = \text{OID} \mid m \mid b \mid hTrunc$$

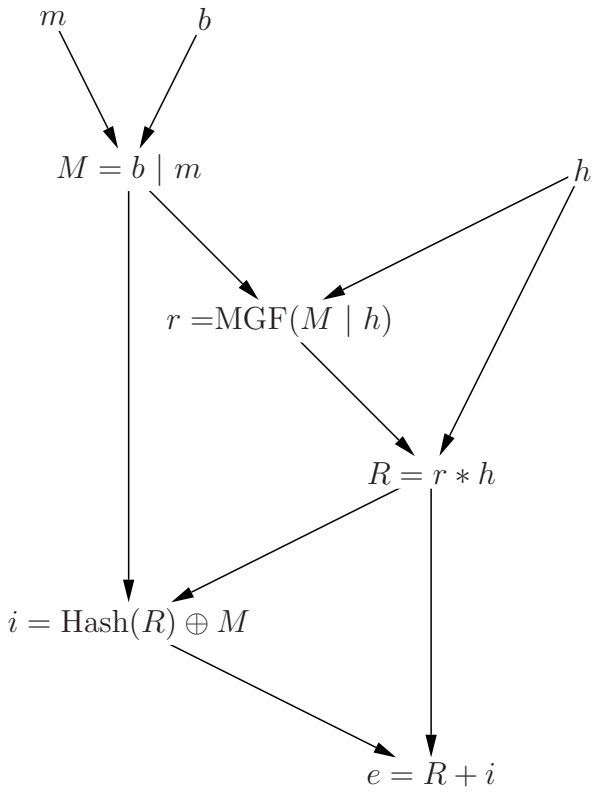


FIGURE 4.1: The encryption of message m with randomness b and public key h.

is used as a seed for the pseudo random generator to produce the blinding polynomial r. We give details of the generation of r later.

We need a so-called *mask generation* function MGF from the ring $\mathbb{Z}_q[x]/(x^N - 1)$ to $\mathbb{Z}_2^N$. This operation is also discussed in more detail later.

Let

$$R \equiv r * h \mod q$$

and

$$m' = \mathrm{MGF}(R) \oplus M.$$

Let $i \in \mathbf{R}$ be the polynomial $i = \sum_{j=0}^{N-1} m'_j x^j$, where m'_j is the $(j+1)^{\text{th}}$ bit of the binary string m'. Now if

$$\frac{N-q}{2} < i(1) < \frac{N+q}{2},$$

then the cryptotext is

$$e \equiv R + i \mod q.$$

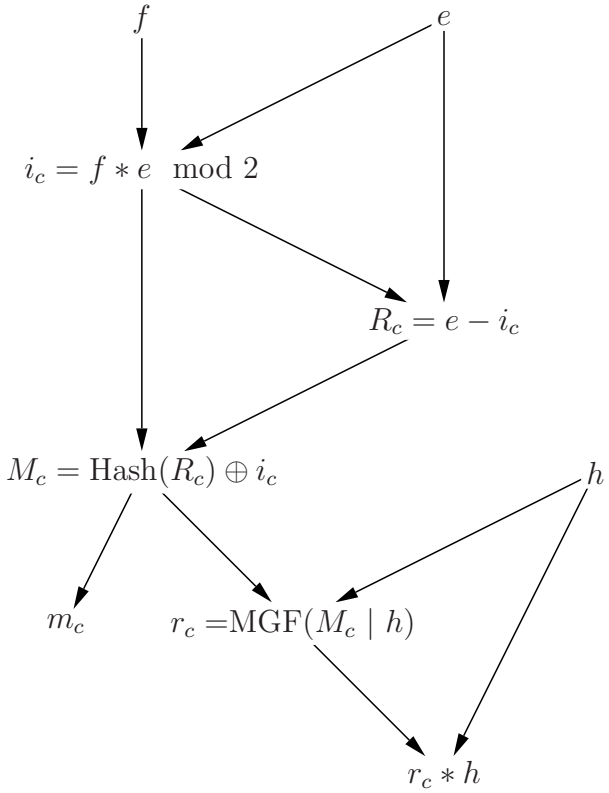


FIGURE 4.2: The decryption of cryptotext e with public key h and private key f.

Otherwise, we need to try again with a different b. The probability of having to try again with our parameter set is

$$\begin{aligned} &P(\text{At least 245 0's in } m') + P(\text{At least 245 1's in } m') \\ &\quad = \binom{251}{245} 2^{-245} + \binom{251}{245} 2^{-245} \approx 10^{-62}. \end{aligned}$$

REMARK 4.7 In practice the number of 1's in the message representative is always in the desired interval. However an attacker may want to use message representatives that have very many or very few 1's. Therefore we must discard those, at least when decrypting.

DECRYPTION

Let $e \in Z_q[x]/(x^N - 1)$ be the cryptotext which we would like to decrypt. The key part of the decryption is to guess $\alpha = r * 2g + i * f$. We know

that $\alpha \equiv i \mod 2$ because $f = 1 + 2F$. We also know that

$$\alpha \equiv r * h * f + i * f \equiv f * e \mod q.$$

Let $a = f * e$. First we compute $i(1)$. From above we see that using the parameter set ees251ep4 we have

$$i(1) \equiv \frac{a(1) - 2r(1)g(1)}{f(1)} \equiv \frac{a(1) - 10368}{145} \equiv 212 - 89a(1) \mod 239.$$

We also know that $6 < i(1) < 245$ and so we can calculate the exact value of $i(1)$. If all of the coefficients of α are about the same size then they should be in the interval $[A + 1, A + q]$, where

$$A = \left\lfloor \frac{2r(1)g(1) + i(1)f(1)}{N} \right\rfloor - \left\lceil \frac{q}{2} \right\rceil = \left\lfloor \frac{10368 + 145i(1))}{251} \right\rfloor - 120.$$

We choose the coefficients of a modulo q from this interval and hope that they equal those of α.

Let $i_c \in \mathbb{Z}_2[x]/(x^N - 1)$ and $i_c \equiv a \mod 2$. This is our candidate for the message representative i. Also let

$$R_c \equiv e - i_c \mod q.$$

Let m'_c be a binary string of length 251 where the $(j+1)^{\text{th}}$ bit is the coefficient of x^j in i_c.

We use the same hash function as in the encryption and get our candidate for M,

$$M_c = Hash(R_c) \oplus m'_c.$$

We denote the fist 80 bits of M_c as b_c and interpret the next 8 bits as the binary representation of l_c. The next l_c bits we call m_c. There should be some bits left over and those should be zeroes. Otherwise, the decryption has failed.

Now we proceed as in the encryption: Let $hTrunc$ be a sequence 80 bits long calculated from the public key h and

$$z_c = \text{OID} \mid m_c \mid b_c \mid hTrunc.$$

This is used as a seed for the pseudo random generator to produce the candidate for the blinding polynomial r_c.

If $h * r_c$ is not equal to R_c modulo q then the decryption has failed. Otherwise, m_c is the decrypted message.

CORRECTNESS

We assume that the decryptor has guessed $\alpha = r * 2g + i * f$ correctly as shown above. Now $\alpha \equiv i \mod 2$ because $f = 1 + 2F$. Therefore the candidate for the message representative equals the message representative which the encryptor used: $i_c = i$. The cryptotext was computed as

$$e \equiv R + i \mod q$$

and so

$$R_c \equiv e - i_c \equiv R \mod q.$$

Thus

$$M_c = \mathrm{MGF}(R_c) \oplus \mathrm{MGF}(R) \oplus M = M.$$

Therefore $b_c = b$, $l_c = l$, $m_c = m$, and the seed for computing r_c equals the seed for r. Finally,

$$h * r_c \equiv h * r \equiv R \equiv R_c \mod q$$

and the decryption is successful.

If the decryptor was unable to guess α we have with very high probability that $i_c \neq i$. Thus

$$R_c \equiv e - i_c \not\equiv e - i \equiv R \mod q$$

and $M_c \neq M$. Again, with very high probability $b_c \neq b$, $m_c \neq m$, and $r_c \neq r$. There is now no reason why $R(= e - i_c)$ should be anything like $h * r_c$.

REMARK 4.8 As expected, the private key f is needed to correctly compute a and to decrypt successfully.

THE FUNCTIONS NEEDED

We saw earlier that we need a pseudo random number generator to produce the blinding polynomial r from the binary string z. The pseudo random number generator is constructed using a hash function, SHA-1. SHA-1 generates an output of 160 bits from any bit string of length less than 2^{64}.

To produce an output longer than 160 bits we use a counter c and compute $\mathrm{Hash}(z \mid c)$ repeatedly for increasing values of c. Here c is the binary representation of the counter using 32 bits. The procedure is as follows:

We interpret every 160 bits of the output as 20 binary numbers of 8 bits in the range $[0, 255]$ when $N = 251$. If N is larger, then we need more bits per number. To start with we take r to be the zero polynomial. Let j_1 be the first number produced by the pseudo random number generator. If $j_1 < N$ we set the coefficient of x^{j_1} in r to 1. We repeat this for the next number j_2 if it differs from j_1. Then we continue reading the output until we have found d_r different values smaller than N, in which case r has exactly d_r coefficients set to 1. Every time we run out of pseudo random numbers we increase c by one and compute again the value of $\mathrm{Hash}(z \mid c)$.

We produce the mask generation function for encryption in a similar way. Let R be a polynomial in $\mathbf{R}$ and R_2 a binary string of length N, where the j^{th} bit is 1 if and only if the coefficient of x^{j-1} in R is odd. The mask generated from R is then the first N bits of

$$\mathrm{Hash}(R_2 \mid 0^{32}) \mid \mathrm{Hash}(R_2 \mid 0^{31}1),$$

where Hash is SHA-1. Again, if N is larger than two times 160, we merge more hashes.

The bit string hTrunc is simply formed by joining the binary representation of the first 10 coefficients $h_0, \ldots, h_9$ of h using 8 bits.

NOTES

The small modulus $p = 2$ must not divide the large modulus q: The cryptotext is computed as

$$e \equiv R + i \equiv r * h + i \equiv r * f^{-1} * p * g + i \mod q.$$

If $p \mid q$ then $e \equiv i \mod p$ and the message can be revealed without key. In general the gcd of p and q should be 1.

It is possible that not all of the coefficients of α lie in the predicted interval. According to the researchers at NTRU [43], the probability of this happening when using the parameter set ees251ep4 is about 10^{-31} with $i(1) = 125$. In this case the decryptor is unable to decrypt the message even when it is normally encrypted. The encryptor cannot know beforehand whether the decryption will be successful or not.

The polynomials r, g, f, and i have only very small coefficients. Therefore it is possible to successfully approximate the size of the coefficients of $\alpha (= r * 2g + i * f)$. Some coefficients of h are large and hence the coefficients of $e (= r * h + i)$ differ significantly in size. If these coefficients

were in some interval, for example $[B, B+q)$, where

$$B = \left\lfloor \frac{r(1)h(1) + i(1)}{N} \right\rfloor - \left\lceil \frac{q}{2} \right\rceil,$$

then it would be easy to find the message representative i: We would guess $i(1)$, lift e into this interval and compute modulo h. Note that $i(1) < N$ and so $i(1)$ has little effect on the formula.

It is not mandatory to use SHA-1 as the hash function. There are also other options in the standard.

AN EXAMPLE

We give a short example to illustrate NTRUEncrypt. Let $N = 7$, $q = 11$ and $d_f = d_g = d_r = 2$. Our private key is $f = 1+2x^4+2x^6$ and our public key is $h = 1+8x+7x^2+8x^3+x^4+5x^5+6*x^6 \equiv f^{-1}*2(x+x^4) \mod 11$. We shall encrypt a message m using randomness b. Let the blinding polynomial be $r = \rho(m,b) = x^2 + x^3$ and the message representative $i = \varphi(m,h,b) = 1 + x + x^4 + x^5$. The cryptotext is now

$$e = 7 + x + 7x^2 + 9x^3 + 5x^4 + 5x^5 + 9x^6 \equiv r * h + i \mod 11.$$

The decryptor obtains

$$a \equiv f * e \equiv 5 + 3x + 2x^2 + 4x^3 + 7x^4 + 3x^5 + 4x^6 \mod 11.$$

This reduced modulo 2 is the message representative, from which the message m and the randomness b can be derived. By re-encrypting m we check that the decryption was successful. Note that

$$i(1) = \frac{28 - 2 \cdot 2 \cdot 2}{5} = 4$$

and

$$A = \left\lfloor \frac{2 \cdot 2 \cdot 2 + 4 \cdot 5}{7} \right\rfloor - \left\lceil \frac{11}{2} \right\rceil = -2.$$

Thus the desired interval for the coefficients of α was [-1,9].

4.4 COMPARISON

According to several estimates (see for example [16]) the security level of NTRU with $N = 251$ is comparable to RSA with 1024 bit numbers or to

		NTRU 251	RSA 1024	ECC 163
public key	(bits)	2008	1024	164
secret key	(bits)	251	1024	163
plaintext block	(bits)	160	702	163
ciphertext block	(bits)	2008	1024	163
encrypt speed	(blocks/sec)	22727	1280	458
	(Mbits/sec)	3.6	0.90	0.075
decrypt speed	(blocks/sec)	10869	110	702
	(Mbits/sec)	1.7	0.077	0.11

TABLE 4.1: A comparison of NTRUEncrypt, RSA and the elliptic curves cryptosystem made using a 800MHz Pentium III computer. The speeds are from the NTRU website [34].

the elliptic curves cryptosystem [2] with a 163 bit field. We now make some comparisons between these cryptosystems. The numerical data can be seen in table 4.1.

The key generation of NTRU is very fast. All we need to do is select random f and g, find the inverse of f and multiply it by $2g$. The inverse is found using the Extended Euclidean algorithm for f and $x^N - 1$. The key generation is about 500 times faster than the key generation of RSA, which requires finding two large primes, multiplying them and finding the secret exponent.

The encryption and decryption are, with NTRU, one or two orders faster than with RSA or with elliptic curves under the same security level. Because of the padding schemes, one block in RSA 1024 can encrypt only 702 bits and NTRU message representatives of length 251 bits correspond to a message of 160 bits.

Because the message representative in NTRU is a binary string and the cryptotext is a polynomial with coefficients as large as q, the encrypted message is considerably longer than the original message.

The key sizes of NTRUEncrypt are about the same as with RSA and elliptic curves, the public key of NTRU is twice the length of the RSA public key.

4.5 EARLIER VERSIONS

The three main versions of NTRU are from the years 1998, 2002 and 2003. In Chapter 9 we will consider in detail the version of NTRU from 2002. Here we introduce this and the version from 1998 for the sake of

N	q	p	d_f	d_g	d_r
107	64	3	15	12	5
167	128	3	61	20	18
263	128	3	50	24	16
503	256	3	216	72	55

TABLE 4.2: Parameter sets of the first version of NTRU.

completeness.

In the version of NTRU from the year 1998, the small modulus p was always 3. There were different parameter sets for different security levels. The parameter N was chosen to be a *safe prime,* that is to say a number N such that both N and $\frac{N-1}{2}$ are primes. The large modulus was a power of 2.

Let us denote by $T(s,t)$ the polynomials in $\mathbf{R}$ with exactly s coefficients equal to 1, t coefficients equal to -1 and remaining coefficients equal to 0.

We select the polynomial f from the set $T(d_f, d_f - 1)$, g from the set $T(d_g, d_g)$ and the blinding polynomials r from the set $T(d_r, d_r)$. The coefficients of the message representative belong to the set $\{-1, 0, 1\}$. The coefficients of α are centered around zero and thus $A = -\frac{q}{2}$. The parameter sets are listed in table 4.2.

A different version of NTRU was presented in draft 4 of the Efficient Embedded Security Standard [8] in 2002. The difference is that the small polynomial is now $2 + x$. Thus the message representative can only take coefficients from the set $\{0, 1\}$.

The generation procedure for the private key f is also different. We can either select an $F \in T(d_f, 0)$ and set $f = 1 + (2 + x) * F$ or we can choose positive polynomials $f_1, f_2, f_3 \in \mathbf{R}$ with $f_i(1) = d_{f_i}$ and compute

$$f = 1 + (2 + x) * (f_1 * f_2 + f_3).$$

Using this procedure, f always has an inverse modulo $2 + x$. Because the inverse $f_p = 1$ we can omit the multiplication by f_p from the decryption phase. We only need to check that f has an inverse modulo q.

The polynomial g is selected from the set $T(d_g, 0)$ and the blinding polynomial is generated from three smaller parts: $r = r_1 * r_2 + r_3$ where $r_1, r_2, r_3 \in \mathbf{R}$ are positive polynomials and $r_i(1) = d_{r_i}$.

Because the coefficients of the polynomials are now no longer centered around zero we need to select a different interval for the coefficients of a

N	q	p	d_f	$d_{f_1}, d_{f_2}, d_{f_3}$	d_g	$d_{r_1}, d_{r_2}, d_{r_3}$
139	64	$2+x$	40		40	0,0,40
251	128	$2+x$	72	8,8,8	72	8,8,8
347	128	$2+x$	64	7,8,8	173	7,8,8
503	256	$2+x$	420	20,20,20	251	12,13,14

TABLE 4.3: Parameter sets of the 2002 version of NTRU.

during decryption.

We first select $[A, A+q)$ to be the candidate for the proper interval, where A is the expected average coefficient of $\alpha = f*i+p*r*g$. This is obtained if $\alpha(1)$ is known. Clearly $\alpha(1) = f(1)i(1)+p(1)r(1)g(1)$, where only $i(1)$ is not known. But as $\alpha(1) \equiv a(1) \mod q$, we can compute $i(1) \mod q$. Because i is a random binary polynomial, it is highly probable that $N/2 - q/2 \leq i(1) < N/2 + q/2$. If we make this assumption, we get the average decryption coefficient A and the first guess on the proper interval.

A wrap (or gap) error only occurs if at least one coefficient differs by $q/2$ from the average. In this case we try the decryption again with slightly increased and decreased A's until we give up and declare failure.

REMARK 4.9 Even though the small modulus p is now a polynomial $(2+x)$, it is easy to see that representatives modulo p are (almost) exactly the binary polynomials in R.

REMARK 4.10 Generating F and g from three parts increases the efficiency of the calculations. At the encryption phase we must compute $r*p*h$. The multiplication $p*h$ need only be calculated once for every public key. However, r is different every time. We have that

$$r*p*h = r_1*(r_2*(p*h)) + r_3*(p*h).$$

When the multiplier has small weight, we can replace the convolution product by a small number of rotations and additions. We can replace the normal convolution product here by some rotations of $p*h$ and $d_{r_1} + d_{r_2} + d_{r_3} - 2$ additions in the ring.

We can do the same trick with f, thus also making the decryption faster.

CHAPTER 5

Lattice reduction and security of NTRU

In this chapter we consider the following problem: given N, q and a polynomial h of degree at most $N-1$, find polynomials f and g in the ring $\mathbf{R} = \mathbb{Z}[x]/(x^N - 1)$ with small coefficients such that $f * h \equiv 2g \mod q$. If we are able to solve this problem then we are able to break the NTRU cryptosystem. In the later chapters we estimate the security of NTRU by estimating the complexity of solving this problem.

One of the solutions of this problem is the private key corresponding to a public key h. With Gaussian heuristic we can see that with high probability there are no solutions with shorter vectors f and g. Also with high probability there are no solutions with slightly longer vectors. However, there are several solutions of the same length.

REMARK 5.1 When there is no possibility of misunderstanding, we denote by f both the polynomial $f_0 + f_1 x + \cdots + f_{N-1}x^{N-1} \in \mathbf{R}$ and the vector $\begin{pmatrix} f_0 & f_1 & \dots & f_{N-1} \end{pmatrix} \in \mathbb{Z}^N$.

We call the vectors

$$f^i = \begin{pmatrix} f_{N-i} & f_{N-i+1} & \cdots & f_{N-1} & f_0 & f_1 & \cdots & f_{N-i-1} \end{pmatrix}$$

and the polynomials $f * x^i$ in $\mathbf{R}$ the *rotations of* f.

If $f*h \equiv 2g \mod q$ then also $f*x^i*h \equiv 2g*x^i \mod q$. Thus if there is one solution to our problem then there are several solutions, namely all the rotations of f and g. Clearly they are all of the same length. Our aim is to find any one of these solutions. Once we have found one rotation of f, we have only N possibilities left for the private key f. The

right one can be found by brute force or by some more sophisticated technique. For example, in the newest version of NTRU, f_0 is the only odd coefficient of f and can thus easily be spotted in any rotation of f.

We can solve the main problem of this chapter by finding the shortest vector of the lattice $L(B)$ with

$$B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{2N} \end{pmatrix} = \left(\begin{array}{ccccc|ccccc} 1 & & & & & h_0 & h_1 & h_2 & \cdots & h_{N-1} \\ & 1 & & 0 & & h_{N-1} & h_0 & h_1 & \cdots & h_{N-2} \\ & & 1 & & & h_{N-2} & h_{N-1} & h_0 & \cdots & h_{N-3} \\ & 0 & & \ddots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ & & & & 1 & h_1 & h_2 & h_3 & \cdots & h_0 \\ \hline & & 0 & & & & & qI & & \end{array}\right)$$

$$= \begin{pmatrix} I & \overrightarrow{h} \\ 0 & qI \end{pmatrix}.$$

We note that the rows of B are linearly independent.

In general, given a polynomial h, there is no guarantee that there exist two small polynomials f and g such that $f * h \equiv 2g \mod q$. However, in all the cases we are interested in such small polynomials exist.

THEOREM 10 *If a polynomial $h \in \mathbf{R}$ has a factorization $h \equiv f^{-1} * 2g \mod q$, $f^{-1}, g \in \mathbf{R}$ then the lattice $L(B)$ contains the vector $\begin{pmatrix} f & 2g \end{pmatrix}$, where $f * f^{-1} \equiv 1 \mod q$.*

Proof. The equation $h \equiv f^{-1} * 2g \mod q$ can be written as a system of equations:

$$\sum_{i+j \equiv k \ (N)} f_i h_j \equiv 2g_k \mod q$$

for $0 \leq k < N$. We can write this system in matrix form as

$$\begin{pmatrix} f_0 & f_1 & \cdots & f_{N-1} \end{pmatrix} \begin{pmatrix} h_0 & h_1 & \cdots & h_{N-1} \\ h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ h_1 & h_2 & \cdots & h_0 \end{pmatrix} \equiv \begin{pmatrix} 2g_0 & 2g_1 & \cdots & 2g_{N-1} \end{pmatrix},$$

where every element is taken modulo q. A vector v belongs to the lattice $L(B)$ if there exists some $u \in \mathbb{Z}^{2N}$ such that

$$u \begin{pmatrix} I & \overrightarrow{h} \\ 0 & qI \end{pmatrix} = v.$$

We have

$$\begin{pmatrix} f & r \end{pmatrix} \begin{pmatrix} I & \overrightarrow{h} \\ 0 & qI \end{pmatrix} = \begin{pmatrix} f & 2g \end{pmatrix}$$

for some $r \in \mathbb{Z}^N$ or, equivalently, using the corresponding polynomials, $f * h + qr = 2g$. ■

Above we considered the problem of finding the private key from the public key. Next we study a weaker form of attack where we try to find the plaintext given the cryptotext.

Let r be a blinding polynomial that was used to generate a cryptotext c and let h be the private key. In the lattice $L(B)$ there is the vector $\begin{pmatrix} r & r * h \end{pmatrix}$. If we append the basis of this lattice by $\begin{pmatrix} 0 & e \end{pmatrix}$ we get a basis matrix

$$B_e = \begin{pmatrix} I & \overrightarrow{h} & \bar{0}^T \\ 0 & qI & \bar{0}^T \\ \bar{0} & e & 1 \end{pmatrix}.$$

Here we increase the dimension by one to keep the basis linearly independent. We see that in this new lattice there is a short vector $\begin{pmatrix} r & -i & -1 \end{pmatrix}$, where i is the used message representative, for some polynomial $s \in \mathbf{R}$:

$$\begin{pmatrix} r & s & -1 \end{pmatrix} \begin{pmatrix} I & \overrightarrow{h} & \bar{0}^T \\ 0 & qI & \bar{0}^T \\ \bar{0} & e & 1 \end{pmatrix} = \begin{pmatrix} r & r * h - e + qs & -1 \end{pmatrix} = \begin{pmatrix} r & -i & -1 \end{pmatrix}.$$

Finding the message representative i reveals the plaintext. Note that both r and i are binary and unknown to the attacker.

5.1 LATTICE CONSTANTS

For a "random" lattice L, the Gaussian heuristic says that the length of the shortest non-zero vector is usually approximately

$$\sigma(L) \approx \sqrt{\frac{\dim(L)}{2\pi e}} \det(L)^{\frac{1}{\dim(L)}}.$$

Let us denote the actual shortest vector of the lattice L by $\lambda(L)$. Practical experiments have shown that the current lattice reduction techniques find the actual shortest vector faster when the lattice constant,

$$c = \frac{\lambda(L)}{\sigma(L)} \sqrt{\dim(L)},$$

gets smaller. In other words, it is easier to find shortest vectors that are small relative to the expected shortest vector.

The remark above suggests that we have two possible ways of facilitating our search for the shortest vector: We can modify the lattice in order to make the shortest vector shorter, or we can make the lattice more sparse and thus enlarge the size of the expected shortest vector.

For a lattice L with basis matrix

$$\left(\begin{array}{ccccc|ccccc} \lambda & & & & & h_0 & h_1 & h_2 & \cdots & h_{N-1} \\ & \lambda & & 0 & & h_{N-1} & h_0 & h_1 & \cdots & h_{N-2} \\ & & \lambda & & & h_{N-2} & h_{N-1} & h_0 & \cdots & h_{N-3} \\ & 0 & & \ddots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ & & & & \lambda & h_1 & h_2 & h_3 & \cdots & h_0 \\ \hline & & 0 & & & & & qI & & \end{array}\right)$$

and balancing constant $\lambda \neq 0$ we have

$$\sigma(L) = \sqrt{\frac{2N}{2\pi e}}\,(\lambda q)^{N\frac{1}{2N}} = \sqrt{\frac{Nq\lambda}{\pi e}}$$

and

$$\lambda(L) \approx \sqrt{||\lambda f||^2 + ||2g||^2}.$$

If we assume that $||f|| \approx ||2g||$, then the value of

$$c = \sqrt{\frac{2\pi e}{q\lambda}(\lambda^2||f||^2 + ||2g||^2)}$$

is minimal when $\lambda = 1$. The minimal value with the parameter set ees251ep4 is $c \approx 6.4$.

For the lattice $L(B_e)$, the length of the expected shortest vector is about the same $(\sqrt{\frac{Nq}{\pi e}})$ but the the length of the actual shortest vector is $\lambda(L(B_e)) \approx \sqrt{N}$. About half of the components are 1 or -1 and the other half are 0. In this case, the constant c is approximately 4.2.

5.2 REDUCING THE LATTICE CONSTANTS

In the following, we manipulate the lattice according to extra information we have about f and g. Let us assume that h is a public key generated

according to the latest version of NTRUEncrypt. There exist polynomials $F, g \in \mathbf{R}$ with d coefficients equal to 1 and $N-d$ coefficients equal to 0 such that $(1+2F) * h \equiv 2g \mod q$. We present some tricks to obtain lattices with smaller lattice constants.

In the parameter set ees251ep4, which we are mainly interested in, we have $N = 251$, $q = 239$ and $d = 72$.

TRICK ONE

The shorter the shortest vector of a lattice, the easier it usually is to find. The length of vector $\begin{pmatrix} f & 2g \end{pmatrix}$ is

$$\sqrt{||f||^2 + ||2g||^2} \approx 2\sqrt{2d}.$$

Note that we know that there exist vectors f and $2g$ with d 2's. We need to find a vector that is always close to f (and $2g$ since they are similar). We take this vector to be $a\bar{1} = \begin{pmatrix} a & a & \cdots & a \end{pmatrix}$. We need to minimize the distance between $\begin{pmatrix} f & 2g \end{pmatrix}$ and $\begin{pmatrix} a\bar{1} & a\bar{1} \end{pmatrix}$:

$$\begin{aligned} \sqrt{||f-a||^2 + ||2g-a||^2} &\approx \sqrt{2(d(2-a)^2 + (N-d)a^2)} \\ &\approx \sqrt{2Na^2 - 8da + 8d}. \end{aligned}$$

The distance is minimal when $a \approx \frac{2d}{N}$. Thus, the minimum distance is approximately

$$\sqrt{2N\left(\frac{2d}{N}\right)^2 - 8d\frac{2d}{N} + 8d} = 2\sqrt{2d - \frac{2d^2}{N}}.$$

It is often advantageous to add this vector, $a\bar{1}$, to the lattice and solve the new shortest vector problem. This is called the *embedding attack.* In our case $\frac{2d}{N} \approx \frac{1}{2}$ and we multiply every element of the lattice by 2 to make them all integers. In matrix form this is

$$\begin{pmatrix} f & r & -1 \end{pmatrix} \begin{pmatrix} 2I & \overrightarrow{2h} \\ 0 & 2qI \\ \bar{1} & \bar{1} \end{pmatrix} = \begin{pmatrix} 2f - \bar{1} & 4g - \bar{1} \end{pmatrix}$$

for some $r \in \mathbb{Z}^N$.

After this trick,

$$\sigma(L) \approx 2\sqrt{\frac{Nq}{\pi e}}, \qquad \lambda(L) \approx \sqrt{2\left((N-d)\cdot 1^2 + d \cdot 3^2\right)}$$

and $c \approx 5.4$.

Now we have two problems. The first one is that the dimension of the lattice is smaller than the number of basis vectors. Therefore the basis vectors are linearly dependent. The second one is that the vector $\begin{pmatrix} a\bar{1} & a\bar{1} \end{pmatrix}$ is the shortest vector of the lattice. This is not the shortest vector which we are interested in.

We solve these problems by finding a linearly independent basis and using some tricks so that $\begin{pmatrix} a\bar{1} & a\bar{1} \end{pmatrix}$ is no longer in the lattice. We do not give the details here.

REMARK 5.2 Above we used only approximations of the length of f. The length of f is actually either $\sqrt{4d+1}$ or $\sqrt{4(d-1)+9}$, depending on whether f_0 is 1 or 3. We used the approximation $\sqrt{4d}$.

The embedding is also useful with $L(B_e)$. The length of the shortest vector in the lattice generated by matrix

$$B_e = \begin{pmatrix} 2I & \overrightarrow{2h} & \bar{0}^T \\ 0 & 2qI & \bar{0}^T \\ \bar{1} & 2e - \bar{1} & 1 \end{pmatrix}.$$

is $\sqrt{2N+1}$. Every coefficient of the shortest vector is ± 1. Thus we have the lattice constant $c \approx 3.0$.

Tricks two and three, discussed below, do not work with $L(B_e)$ or most other lattices. They are specific to the way the private key is generated in NTRUEncrypt.

TRICK TWO

We now consider the problem from the point of view of an attacker trying to break a given NTRU key. The attacker knows that there exists an f such that $f = 1 + 2F$. Trivially we can write

$$2F \cdot 2 - \sum_{i=0}^{N-1} x^i = 4F - \sum_{i=0}^{N-1} x^i$$

and because

$$(1 + 2F) * 2h + 2qr - \sum_{i=0}^{N-1} x^i = 4g - \sum_{i=0}^{N-1} x^i,$$

we have

$$2F * 2h + 2qr - \left(\sum_{i=0}^{N-1} x^i - 2h\right) = 4g - \sum_{i=0}^{N-1} x^i.$$

In matrix form we can write the above as

$$\begin{pmatrix} 2F & r & -1 \end{pmatrix} \begin{pmatrix} 2I & \overrightarrow{2h} \\ 0 & 2qI \\ \bar{1} & \bar{1} - 2h \end{pmatrix} = \begin{pmatrix} 4F - \bar{1} & 4g - \bar{1} \end{pmatrix}.$$

We move the scalar 2 and get

$$\begin{pmatrix} F & r & -1 \end{pmatrix} \begin{pmatrix} 4I & \overrightarrow{4h} \\ 0 & 2qI \\ \bar{1} & \bar{1} - 2h \end{pmatrix} = \begin{pmatrix} 4F - \bar{1} & 4g - \bar{1} \end{pmatrix}.$$

This means that

$$F * 4h + 2qr + 2h = 4g$$

and because the attacker knows h he also knows the parity of the coefficients of r if $2 \nmid q$. Let $r = 2r' + r''$ where $r'' \in \mathbb{Z}_2[x]/(x^N - 1)$ is known by the attacker. We now have

$$\begin{pmatrix} F & r' & -1 \end{pmatrix} \begin{pmatrix} 4I & \overrightarrow{4h} \\ 0 & 4qI \\ \bar{1} & \bar{1} - 2h - 2qr'' \end{pmatrix} = \begin{pmatrix} 4F - \bar{1} & 4g - \bar{1} \end{pmatrix}.$$

After trick two

$$\sigma(L) \approx 4\sqrt{\frac{Nq}{\pi e}}, \qquad \lambda(L) = \sqrt{2\left((N-d) \cdot 1^2 + d \cdot 3^2\right)}$$

and $c \approx 2.7$.

Again we note that we do not have a linearly independent basis. However, we can easily find one. Unlike in the previous case, now the length of $\begin{pmatrix} \bar{1} & \bar{1} - 2h - 2qr'' \end{pmatrix}$ is quite large compared to the length of $\begin{pmatrix} 4F - \bar{1} & 4g - \bar{1} \end{pmatrix}$, so this does not give us any problems.

REMARK 5.3 The rotations of $4F - \bar{1}$ and $4g - \bar{1}$ do not belong to the lattice. The last row of the matrix breaks the symmetry. For example, the lattice point we get using the vector corresponding to polynomials $F * x$ and $r' * x$,

$$\begin{pmatrix} F^1 & r'^1 & -1 \end{pmatrix} \begin{pmatrix} 4I & \overrightarrow{4h} \\ 0 & 4qI \\ \bar{1} & \bar{1} - 2h - 2qr'' \end{pmatrix}$$
$$= \begin{pmatrix} 4F^1 - \bar{1} & 4g^1 - 2h^1 + 2h - 2qr''^1 + 2qr'' - \bar{1} \end{pmatrix},$$

is not short. The first part $(4F^1 - \bar{1})$ is short, but the second is not.

TRICK THREE

The attacker knows the weight of f, g, h, and r''. Therefore (s)he can calculate the weights of r and r':

$$r(1) = \frac{2g(1) - f(1)h(1)}{q} \quad \text{and}$$
$$r'(1) = r(1) - r''(1).$$

For any scalar m we have

$$\begin{pmatrix} F & r' & -1 \end{pmatrix} \begin{pmatrix} 4I & \overrightarrow{4h} & \bar{0}^T \\ 0 & 4qI & \bar{m}^T \\ \bar{1} & \bar{1} - 2h - 2qr'' & mr'(1) \end{pmatrix} = \begin{pmatrix} 4F - \bar{1} & 4g - \bar{1} & 0 \end{pmatrix}.$$

Alternatively, (call this trick 3B) we can use the knowledge we have about F. The weight of F is d. We can write

$$\begin{pmatrix} F & r' & -1 \end{pmatrix} \begin{pmatrix} \overrightarrow{4+\bar{1}} & \overrightarrow{4h+\bar{1}} \\ 0 & 4qI \\ \overline{1+d} & \overline{1+d} - 2h - 2qr'' \end{pmatrix} = \begin{pmatrix} 4F - \bar{1} & 4g - \bar{1} \end{pmatrix}.$$

This time we have added 1 to every element in the first N rows of the matrix of trick two and added d to every element of the last row. In the end these cancel each other out.

The length of the target vector $\lambda(L)$ stays the same after either version of trick three but the determinant of L increases together with the dimension of L. The quantity $\sigma(L)$ is now slightly larger than before, resulting in a small decrease in c. The size of this decrease depends on the value of h and is much smaller than the decrease in c obtained using tricks one or two.

We cannot use trick three for the lattice $L(B_e)$ because we do not know the weights of r or i.

By applying trick one, we succeeded in reducing the lattice constant of $L(B_e)$ to 3.0. On the other hand, the lattice constant of $L(B)$ is 2.7 after trick two. Thus it seems that finding the private key using $L(B)$ is easier than finding the plaintext using $L(B_e)$.

REMARK 5.4 In the paper [16], the writers give an alternative definition of the lattice constants. The writers define the length of a vector $v = (v_1, v_2, \ldots, v_n)$ as the centered norm

$$||v||_c = \sqrt{\sum_{i=1}^{n} \left(v_i - \frac{1}{n} \sum_{j=1}^{n} v_j \right)^2} = \sqrt{\sum_{i=1}^{n} v_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} v_i \right)^2}.$$

After trick two their constants are the same as the constants defined here. The last row is selected in such a way that the length of the shortest vector is its centered norm (that is, the average value of its components is 0).

TRICKS FOR INVERSE OF PUBLIC KEY

Above we tried to find small polynomials f and g such that $f * h \equiv 2g \mod q$. We could also write this equation as $f \equiv 2g * h^{-1} \mod q$. Because g has an inverse, we know that $h^{-1} \equiv f * 2^{-1} g^{-1} \mod q$ also exists. In matrix form we have

$$\begin{pmatrix} 2g & s \end{pmatrix} \begin{pmatrix} I & \overrightarrow{h^{-1}} \\ 0 & qI \end{pmatrix} = \begin{pmatrix} 2g & f \end{pmatrix}$$

for some vector $s \in \mathbb{Z}^N$ or, equivalently, using corresponding polynomials, $2g * h^{-1} + qs = f$. As with trick one, we have

$$\begin{pmatrix} 2g & s & -1 \end{pmatrix} \begin{pmatrix} 2I & \overrightarrow{2h^{-1}} \\ 0 & 2qI \\ \bar{1} & \bar{1} \end{pmatrix} = \begin{pmatrix} 4g - \bar{1} & 2f - \bar{1} \end{pmatrix}.$$

Trick two does not work completely. We get

$$\begin{pmatrix} g & s & -1 \end{pmatrix} \begin{pmatrix} 4I & \overrightarrow{4h^{-1}} \\ 0 & 2qI \\ \bar{1} & \bar{1} \end{pmatrix} = \begin{pmatrix} 4g - \bar{1} & 4F + 2e_1 - \bar{1} \end{pmatrix} \tag{5.1}$$

where $e_1 = \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}$ and

$$g * 4h^{-1} + 2qs = 4F + 2.$$

We could proceed as with trick two, divide s into two parts $s = 2s' + s''$ and replace $2qI$ by $4qI$ in the matrix in (5.1). Doing so would result in

the loss of an important property of the lattice. Until now, all of the rotations of $4g$ and $2F+2$ have belonged to the lattice. If we follow trick two we need to fix the location of the element to which 2 is added.

If we want the lattice to be symmetric such that all rotations of $2g$ and f belong to the lattice, the best we can do is to use (5.1) with the ideas of trick three. We have

$$s(1) = \frac{f(1) - 2g(1)h^{-1}(1)}{q}$$

and, for any scalar m,

$$\begin{pmatrix} g & s & -1 \end{pmatrix} \begin{pmatrix} 4I & \overrightarrow{4h^{-1}} & \bar{0}^T \\ 0 & 2qI & \bar{m}^T \\ \bar{1} & \bar{1} & ms(1)) \end{pmatrix} = \begin{pmatrix} 4g - \bar{1} & 4F + 2e_1 - \bar{1} & 0 \end{pmatrix}.$$

5.3 ZERO FORCING

Next we consider a method called zero forcing. The idea of this attack is due to Alexander May [27, 28].

Let us still consider the NTRU-lattice L with matrix

$$\begin{pmatrix} I_N & \overrightarrow{h} \\ 0 & qI_N \end{pmatrix}.$$

If we know or guess that, for example, the coefficient of x^{42} in f is 0, then we can reduce the dimension of this lattice. We obtain a new lattice L', and a basis for it, by removing the row corresponding to x^{42} from the basis matrix of L. As a result, the new basis matrix has a column consisting of zeros. This column can naturally also be removed. We obtain the square matrix

$$\begin{pmatrix} I_{N-1} & H' \\ 0 & qI_N \end{pmatrix}$$

which has dimension $2N-1$. Let f' be the vector in $\mathbb{Z}^{250}$ with $f'_i = f_i$ for $0 \leq i \leq 41$, and $f'_i = f_{i+1}$ for $42 \leq i \leq 249$. For the original matrix we have that

$$\begin{pmatrix} f & r \end{pmatrix} \begin{pmatrix} I_N & \overrightarrow{h} \\ 0 & qI_N \end{pmatrix} = \begin{pmatrix} f & 2g \end{pmatrix}.$$

Thus, for the new matrix, if our guess was right, we have

$$\begin{pmatrix} f' & r \end{pmatrix} \begin{pmatrix} I_{N-1} & H' \\ 0 & qI_N \end{pmatrix} = \begin{pmatrix} f' & 2g \end{pmatrix}.$$

Let us consider the lattice constants. We have

$$\sigma(L') \approx \sqrt{\frac{2N-1}{2\pi e}} q^{\frac{N}{2N-1}} < \sigma(L)$$

and

$$\lambda(L') = \lambda(L).$$

Thus c gets smaller and the lattice reduction techniques become faster.

REMARK 5.5 In his original article, May had a different approach to the reduction. Instead of removing a row and a column, he multiplied a column by a large integer θ. Thus the element corresponding to this column was zero in all the short vectors of the resulting lattice. The length of the shortest vector of the new lattice was in this case

$$\sqrt{\frac{2N}{2\pi e}} \sqrt{q} \theta^{\frac{1}{2N}}.$$

If θ is large enough, the shortest vector is longer than above. However, a huge θ slows down the calculations and thus the method of removing a row and a column is considered better.

Instead of guessing that one coefficient of f is zero we can guess a whole pattern of coefficients to be zeroes. The more coefficients we guess to be zero, the faster the reduction becomes.

It is enough that there is a matching pattern of zeroes in one of the rotations of f. If $f' = f \cdot x^i$, $g' = g \cdot x^i$, and $r' = r \cdot x^i$ we see that the equality

$$\begin{pmatrix} f' & r' \end{pmatrix} \begin{pmatrix} I_N & \overrightarrow{h} \\ 0 & qI_N \end{pmatrix} = \begin{pmatrix} f' & 2g' \end{pmatrix}$$

holds for any i and that $||f|| = ||f'||$ and $||g|| = ||g'||$.

We can always find an index i such that $f'_{42} = 0$ when $f' = f \cdot x^i$. With our parameter set we can even find an index i such that $f'_{42} = f'_{43} = f'_{44} = f'_{45} = 0$. There are always at least four consecutive zero coefficients in f. With high probability there are more than four consecutive zeroes.

THEOREM 11 *The probability of guessing correctly a pattern of r zeroes in any rotation of $f \in \mathbb{Z}[x]/(x^N - 1)$ with $N - d$ zeroes is approximately*

$$1 - \left(1 - \prod_{i=0}^{d-1} \left(1 - \frac{r}{N-i}\right)\right)^N.$$

Proof. The probability of guessing correctly a pattern of r zeroes in f is

$$\frac{\binom{N-d}{r}}{\binom{N}{r}} = \frac{(N-r)(N-r-1)\cdots(N-r-d+1)}{N(N-1)\cdots(N-d+1)} = \prod_{i=0}^{d-1}\left(1-\frac{r}{N-i}\right).$$

If we assume that probability of success is always the same for all the rotations of f we get

$$\begin{aligned}\text{Prob(succes with at least one rotation)}&\\ &= 1 - \text{Prob(failure with all rotations)}\\ &\approx 1 - \text{Prob(failure with } f)^N\\ &\approx 1 - (1 - \text{Prob(success with } f))^N\\ &\approx 1 - \left(1 - \prod_{i=0}^{d-1}\left(1-\frac{r}{N-i}\right)\right)^N.\end{aligned}$$

■

The problem with this approach is that the zero patterns in different rotations of f are not independent. The least probable pattern of r zeroes turns out to be r consecutive zeroes. The best choice seems to be guessing a random pattern of zeroes. Doing so would mean that the generator of the keys cannot prepare against the attack.

Let us next consider the advantage in guessing a pattern. Later in this thesis we state that the logarithm of the time required to solve the CVP of size N is about $\alpha N + \beta$ for some α and β. Let $P(r)$ be the probability of guessing a valid pattern of length r and let $T(r)$ be the time needed to solve the CVP after we have guessed a pattern of length r. The expected advantage of guessing is measured by

$$\begin{aligned}\text{Gain} &= P(r)\cdot\frac{T(0)}{T(r)}\\ &\approx \left(1 - \left(1 - \prod_{i=0}^{d-1}\left(1-\frac{r}{N-i}\right)\right)^N\right)10^{\alpha r}.\end{aligned}$$

Using this formula we can estimate the optimal size for the pattern.

REMARK 5.6 Here we used the function

$$T(r) = 10^{\alpha(N-r)+\beta}$$

for the time it takes to solve the CVP after we have guessed correctly a pattern of zeroes of length r. This is at best a rough estimate. The reduced matrices have different balancing constants and the ratio of ones and zeroes in the target vectors is also different.

PARALLELIZATION

Next we shall consider how much faster the CVP can be solved if we have several computers instead of one. We need to know which pattern length is optimal for some number of computers. We try to minimize the function which tells us how much time we need to solve the CVP using K computers, after we have guessed K patterns of length r:

$$\begin{aligned}\text{TotalTime}(K,r) &= \frac{T(r)}{P(\text{Guessed right at least once})} \\ &= \frac{T(r)}{1-P(\text{Guessed wrong every time})} \\ &= \frac{T(r)}{1-(1-P(r))^K} \\ &\approx \frac{10^{\alpha(N-r)+\beta}}{1-\left(1-\prod_{i=0}^{d-1}\left(1-\frac{r}{N-i}\right)\right)^{KN}}.\end{aligned}$$

Using this function we can find the optimal pattern length r_K for every K. The expected advantage of using K computers is

$$\text{Gain}(K) = \frac{\text{TotalTime}(1,r_1)}{\text{TotalTime}(K,r_K).}$$

Remark 5.6 also applies here.

5.4 OTHER WAYS TO REDUCE THE DIMENSION

Let us fist recall that the rows of matrix

$$L = \begin{pmatrix} I_N & \overrightarrow{h} \\ 0 & qI_N \end{pmatrix}$$

are the basis vectors $b_1, \ldots, b_{2N}$. We call the first N rows the *h-rows* and the last N rows the *q-rows*.

Above we eliminated one of the h-rows in matrix L. This resulted in a zero column in the matrix. We then removed this column and afterwards the matrix had dimension $2N-1$ and determinant q^N.

What if we remove one of the q-rows, for example the row number $(251+42)=293$? The target vector we are interested in, $\begin{pmatrix} f & g \end{pmatrix}$, is still in the lattice if the coefficient of x^{41} in g equals the coefficient of x^{41} in $f*h$ (that is, if the reduction modulo q is redundant with this coefficient). If this is true, the CVP for the new, lower dimensional lattice solves the CVP for the original lattice. Or, instead, we could replace the basis vector b_{293} with its multiple, $2b_{293}$. This trick works if the coefficient of x^{41}in g equals the coefficient of x^{41} in $f*h$ modulo $2q$. In this case the determinant of the resulting lattice is $2q^N$.

Instead of rows, one could remove some columns. If we remove one of the columns on the right hand side, then one of the q-rows turns into a zero row. If we also remove this row then the resulting matrix has dimension $2N-1$ and determinant q^{N-1}.

The remaining option is to remove one of the columns on the left hand side. Doing so would make the rows linearly dependant thus enabling us to remove a row. Let b_i be the basis vector that lost its 1 on the left hand side when we removed the column i. Let us assume that one of the coefficients of h, h_j say, is 1 (or -1). Let $k \equiv i+j \mod 251$. We can present b_{251+k} as a linear combination of other vectors b_ℓ:

$$b_{251+k} = qb_i + \sum_{\substack{1\le \ell \le 251 \\ \ell \ne k}} a_\ell b_{251+\ell},$$

where $a_\ell \in \mathbb{Z}$. Thus we must remove vector b_{251+k} from the basis. The resulting matrix has dimension $2N-1$ and determinant q^{N-1}.

5.5 SUMMARY

Our tricks (one to three) are alternatives to the other techniques mentioned at the end of this chapter. Because the rotations of the keys do not belong to the changed lattices, we need to guess correctly a zero run in f, rather than in any of the rotations of f. Thus the zero forcing technique is not efficient in this case. Similarly the parallelization method above is now useless.

However, we could always remove some columns, as in Section 5.4, and speed up the computations slightly.

CHAPTER 6

On lattice reduction algorithms

In this chapter we shall present some algorithms for lattice reduction. The algorithms aim to find a basis that has the shortest possible vectors. In particular we hope to find a basis where one of the basis vectors is the shortest vector of the lattice. These algorithms work for any lattice, not just the lattices which we have presented earlier. The same algorithms can be used, for example, to factorize polynomials.

All of our lattices are *integral lattices*, i.e. all of the elements of the basis vectors are integers.

We presented the Gram-Schmidt orthogonalization $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n$ of a basis $b_1, b_2, \ldots, b_n$ in Chapter 3:

$$\begin{aligned} \hat{b}_1 &= b_1 \\ \hat{b}_j &= b_j - \sum_{i=1}^{j-1} \mu_{j,i} \hat{b}_i, \quad j = 2, \ldots, n, \end{aligned}$$

where

$$\mu_{j,i} = \frac{(b_j, \hat{b}_i)}{(\hat{b}_i, \hat{b}_i)}.$$

6.1 THE LLL ALGORITHM

In their seminal paper [25], Lenstra, Lenstra and Lovász present a polynomial time algorithm for finding a short vector of a lattice. A basis of

a lattice is called *LLL-reduced* if it is *size-reduced,* i.e.

$$|\mu_{j,i}| \leq \frac{1}{2}, \quad 1 \leq i < j \leq n, \tag{6.1}$$

and if

$$\delta||\hat{b}_i||^2 \leq \mu_{i+1,i}^2||\hat{b}_i||^2 + ||\hat{b}_{i+1}||^2 \quad \text{when } i = 1, \ldots, n-1 \tag{6.2}$$

for some $\delta \in (\frac{1}{4}, 1)$.

The first part of the algorithm, size-reducing, is easy. We can size-reduce vector b_j by replacing it in the basis by vector

$$b_j^{\text{new}} = b_j - \sum_{i=1}^{j-1} [\mu_{j,i}]\, b_i,$$

where $[\mu_{j,i}]$ denotes the integer closest to $\mu_{j,i}$. Clearly the new basis generates the same lattice as the original.

The second part is more tricky. The right hand side of inequality (6.2) is the square of the length of the component of b_{i+1} orthogonal to space $\text{span}(b_1, b_2, \ldots, b_{i-1})$. The length of the component of b_i orthogonal to this space is $||\hat{b}_i||$. If inequality (6.2) does not hold, then we can make it hold by swapping the basis vectors b_i and b_{i+1}. This changes the quantities $\mu_{i,j}$ and $\mu_{i+1,j}$. Furthermore, inequality (6.2) may no longer hold for $i - 1$.

The parameter δ controls how strict inequality (6.2) is. If δ is close to 1, the resulting basis may have shorter vectors but it will take more swaps and time to find them.

Later we shall present a polynomial time algorithm for LLL-reduction, but first we consider some properties of LLL-reduced lattices.

The following lemma gives us an upper bound for the length of the LLL-reduced basis vectors b_i using the length of the orthogonalized vectors $\hat{b}_j$.

LEMMA 1 *Let $b_1, b_2, \ldots, b_n$ be an LLL-reduced basis for a lattice L in $\mathbb{R}^n$ and let $\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n$ be defined as above. We have*

$$||b_i||^2 \leq \left(\frac{4}{4\delta - 1}\right)^{j-1} ||\hat{b}_j||^2 \quad \textit{for } 1 \leq i \leq j \leq n.$$

Proof. From (6.2) and (6.1) we have

$$||\hat{b}_{i+1}||^2 \geq (\delta - \mu_{i+1,i}^2)||\hat{b}_i||^2 \geq \frac{4\delta - 1}{4}||\hat{b}_i||^2$$

for $1 \leq i < n$ and, by induction,

$$||\hat{b}_i||^2 \leq \left(\frac{4}{4\delta - 1}\right)^{j-i} ||\hat{b}_j||^2, \tag{6.3}$$

where $1 \leq i \leq j \leq n$. Because the vectors $\hat{b}_i$ are orthogonal, from (3.1) and (6.1) we obtain

$$\begin{aligned}
||b_j||^2 &= ||\hat{b}_j||^2 + \sum_{i=1}^{j-1} \mu_{j,i}^2 ||\hat{b}_i||^2 \\
&\leq ||\hat{b}_j||^2 + \sum_{i=1}^{j-1} \frac{1}{4} \left(\frac{4}{4\delta - 1}\right)^{j-i} ||\hat{b}_j||^2 \\
&= \left(1 + \frac{1}{4} \frac{\left(\frac{4}{4\delta-1}\right)^j - \frac{4}{4\delta-1}}{\frac{4}{4\delta-1} - 1}\right) ||\hat{b}_j||^2 \\
&= \left(1 + \frac{4\delta - 1}{4} \cdot \frac{1}{5 - 4\delta} \left(\left(\frac{4}{4\delta - 1}\right)^j - \frac{4}{4\delta - 1}\right)\right) ||\hat{b}_j||^2 \\
&\leq \left(\frac{4}{4\delta - 1}\right)^{j-1} ||\hat{b}_j||^2
\end{aligned}$$

for all $1 \leq j \leq n$. From this and (6.3) we get

$$||b_i||^2 \leq \left(\frac{4}{4\delta - 1}\right)^{i-1} ||\hat{b}_i||^2 \leq \left(\frac{4}{4\delta - 1}\right)^{j-1} ||\hat{b}_j||^2$$

for all $1 \leq i \leq j \leq n$. ■

The following lemma shows that we can use the length of the LLL-reduced basis vectors to obtain a bound for the length of linearly independent vectors.

LEMMA 2 *Let $b_1, b_2, \ldots, b_n$ be an LLL-reduced basis for a lattice L in $\mathbb{R}^n$ and, for any $t \leq n$, let $x_1, x_2, \ldots, x_t$ be linearly independent in L. We have that*

$$||b_t||^2 \leq \left(\frac{4}{4\delta - 1}\right)^{n-1} \cdot \max(||x_1||^2, ||x_2||^2 \ldots, ||x_t||^2).$$

Proof. We choose k such that x_i is in $L(b_1, \ldots, b_k)$ for all $i = 1, \ldots, t$ but such that x_ℓ is not in $L(b_1, \ldots, b_{k-1})$ for some $\ell \in \{1, \ldots, t\}$. The

distance of x_ℓ from the closest point in $L(b_1, \ldots, b_{k-1})$ is some integer multiple of $||b_k||$. Therefore

$$||x_\ell||^2 \geq ||\hat{b}_k||^2.$$

We know that $k \geq t$: we cannot represent t linearly independent vectors as linear combinations of less than t vectors b_i. Using the previous lemma we get

$$||b_t||^2 \leq \left(\frac{4}{4\delta - 1}\right)^{k-1} ||\hat{b}_k||^2 \leq \left(\frac{4}{4\delta - 1}\right)^{n-1} ||\hat{b}_k||^2 \leq \left(\frac{4}{4\delta - 1}\right)^{n-1} ||x_\ell||^2.$$

■

We defined earlier the successive minima $\lambda_1, \ldots, \lambda_n$ as

$$\lambda_i = \lambda_i(L) = \inf \left\{ \max_{j=1,\ldots,i} (||x_j||) \,\middle|\, x_j \in L \text{ linearly independent} \right\}$$

for $i = 1, \ldots, n$.

Using the two lemmata above we can prove:

THEOREM 12 *Let $b_1, b_2, \ldots, b_n$ form an LLL-reduced basis for a lattice L in $\mathbb{R}^n$ and let $\lambda_1, \lambda_2, \ldots, \lambda_n$ denote the successive minima on L. Then we have*

$$\left(\frac{4}{4\delta - 1}\right)^{1-i} \lambda_i^2 \leq ||b_i||^2 \leq \left(\frac{4}{4\delta - 1}\right)^{n-1} \lambda_i^2 \quad \textit{for } 1 \leq i \leq n.$$

Proof. The upper bound follows directly from the previous lemma. For the lower bound we note that

$$\lambda_i^2 \leq \max(||b_1||^2, ||b_2||^2 \ \ldots, ||b_i||^2)$$

and, by Lemma 1,

$$||b_i||^2 \leq \left(\frac{4}{4\delta - 1}\right)^{j-1} ||\hat{b}_j||^2 \leq \left(\frac{4}{4\delta - 1}\right)^{j-1} ||b_j||^2$$

when $1 \leq i \leq j \leq n$. ■

In practice, the bounds for $||b_i||^2$ are much tighter. In particular, $||b_1||$ is much smaller than the upper limit given above. However, as n gets larger, b_1 is usually not the shortest vector of the lattice.

The algorithm for LLL-reduction is:

$k = 2$
While $k \leq n$ **do**
    size-reduce the vector b_k and update μ_{kj} for $j = 1, \ldots, k-1$
    If $\delta||\hat{b}_{k-1}||^2 > \mu_{k,k-1}^2||\hat{b}_{k-1}||^2 + ||\hat{b}_k||^2$
        Then Swap(b_k, b_{k-1})
            $k = \max(k-1, 2)$
        Else $k = k + 1$
Return $b_1, \ldots, b_n$

Let us consider the number of arithmetic operations needed. We define

$$d_i = \det((b_j, b_k)_{1 \leq j,k \leq i}) \tag{6.4}$$

for $0 \leq i \leq n$. If we consider b_j and $\hat{b}_j$ as row vectors, it can be easily seen that

$$\begin{aligned} d_i &= \det((b_1, \ldots, b_i)(b_1, \ldots, b_i)^T) \\ &= \det((\hat{b}_1, \ldots, \hat{b}_i)(\hat{b}_1, \ldots, \hat{b}_i)^T) \det((\mu_{k,j})_{1 \leq j,k < i})^2 \\ &= \det((\hat{b}_j, \hat{b}_k)_{1 \leq j,k \leq i}) \\ &= \prod_{j=1}^{i} ||\hat{b}_j||^2 \end{aligned} \tag{6.5}$$

for $0 \leq i \leq n$. This is due to $(\mu_{k,j})_{1 \leq j,k < i}$ being a lower triangular matrix, and the fact that $\mu_{j,j} = 1$, and $(\hat{b}_j, \hat{b}_k) = 0$ for all $j \neq k$. We also define

$$D = \prod_{i=1}^{n-1} d_i.$$

THEOREM 13 *Let $L \subset \mathbb{Z}^n$ be a lattice with basis $b_1, b_2, \ldots, b_n$, and let B be a real number greater than or equal to 2 such that $||b_i||^2 \leq B$ for all $1 \leq i \leq n$. Then the number of arithmetic operations needed by the basis reduction algorithm is $\mathcal{O}(n^5 \log B)$, and the integers on which these operations are performed each have length $\mathcal{O}(n \log B)$.*

Proof. At the beginning of the algorithm we have $d_i \leq B^i$ and thus $D \leq B^{\frac{n(n-1)}{2}}$. Every time we perform a swap, some d_{k-1} is reduced by a factor greater than δ and all other d_i's remain unchanged. Hence D is reduced by a factor greater than δ. From (6.4) we see that every $d_i \in \mathbb{Z}$

and from (6.5) we see that $d_i > 0$ because $||\hat{b}_j||^2 > 0$. Thus D is always greater than or equal to 1 and a swap is performed at most $\mathcal{O}(n^2 \log B)$ times. Between two swaps k can be incremented at most $\mathcal{O}(n)$ times. Every time k changes we need $\mathcal{O}(n^2)$ operations for size-reduction and updating μ_{kj}. In total at most $\mathcal{O}(n^5 \log B)$ arithmetic operations can be performed before D reduces to a number less than one.

We noted earlier that $d_i \leq B^i$ and therefore the d_i's are of the desired length. Let us first show that the d_i's are the only denominators we need. The following three properties are proved below:

$$||\hat{b}_j||^2 = \frac{d_j}{d_{j-1}} \quad 1 \leq i \leq n \tag{6.6}$$

$$d_{j-1}\hat{b}_j \in \mathbb{Z}^n \quad 1 \leq j \leq n \tag{6.7}$$

$$d_i \mu_{j,i} \in \mathbb{Z} \quad i \leq i < j \leq n. \tag{6.8}$$

Equation (6.6) is obvious from (6.5). For (6.7) we write

$$b_j - \hat{b}_j = \sum_{i=1}^{j-1} s_{j,i} b_i, \quad \text{where } s_{j,i} \in \mathbb{R}.$$

When we take the inner product of both sides with different b_k, we get the following system of equations:

$$(b_j, b_k) = \sum_{i=1}^{j-1} s_{j,i}(b_i, b_k), \quad 1 \leq k \leq j-1.$$

The determinant of this system is

$$\det((b_i, b_k)_{1 \leq i,k \leq j-1}) = d_{j-1}.$$

Using Cramer's rule we get that $d_{j-1}s_{j,i} \in \mathbb{Z}$. Thus

$$d_{j-1}\hat{b}_j = d_{j-1}b_j - \sum_{i=1}^{j-1} d_{j-1}s_{j,i}b_i \in \mathbb{Z}^n$$

for all $1 \leq j \leq n$. This proves (6.7). We then get (6.8) directly from (6.6) and (6.7):

$$d_i \mu_{j,i} = d_i \frac{(b_j, \hat{b}_i)}{(\hat{b}_i, \hat{b}_i)} = d_{i-1}(b_j, \hat{b}_i) = (b_j, d_{i-1}\hat{b}_i) \in \mathbb{Z}.$$

Now we consider the size of the b_j, $\hat{b}_j$, and $\mu_{j,i}$. At the start of the algorithm we have $||\hat{b}_j||^2 \leq ||b_j||^2 \leq B$ and during the algorithm $\max(||\hat{b}_j||)$ does not increase. So we have

$$||\hat{b}_j||^2 \leq B$$

all of the time.

Now we consider b_j. Because $\mu_{j,i}^2 \leq \frac{1}{4}$, after every size-reduction of b_j we get

$$\begin{aligned} ||b_j||^2 = \sum_{i=1}^{j} \mu_{j,i}^2 ||\hat{b}_i||^2 &\leq ||\hat{b}_j||^2 + \frac{j-1}{4} \max_{i=1,\ldots,j-1} (||\hat{b}_i||^2) \\ &\leq \left(1 + \frac{j-1}{4}\right) B. \end{aligned} \tag{6.9}$$

Using the Cauchy-Schwarz inequality we get

$$\mu_{j,i}^2 = \frac{(b_j, \hat{b}_i)^2}{||\hat{b}_i||^4} \leq \frac{||b_j||^2 ||\hat{b}_i||^2}{||\hat{b}_i||^4} \leq \frac{||b_j||^2}{||\hat{b}_i||^2} = \frac{d_{i-1} ||b_j||^2}{d_i}. \tag{6.10}$$

After size-reduction of b_k the above inequality holds for all $i < k$. We obtain an upper limit in terms of B^j using (6.9), (6.10) and the inequalities $1 \leq ||d_j|| \leq B^j$,

$$\mu_{j,i}^2 \leq \left(\frac{3+j}{4}\right) B^i.$$

Therefore the numerators are of length at most $\mathcal{O}(n \log B)$.

Let us now consider $\mu_{j,i}^2$ during the size-reduction of b_j. Because

$$b_j^{new} = b_j - [\mu_{j,i}]\, b_i$$

then we also have that

$$\mu_{j,k}^{new} = \mu_{j,k} - [\mu_{j,i}]\, \mu_{i,k}$$

for all $k = 1, \ldots, j-1$. Let us denote by M_j the largest of the $\mu_{j,i}$ for $1 \leq i \leq j$. Clearly after one reduction $M_j^{new} \leq M_j + \frac{1}{2}(M_j + \frac{1}{2})$. Note that, because $i < j$, b_i is already size-reduced and $|\mu_{i,k}| \leq \frac{1}{2}$. We see that each of the $j-1$ size-reductions of b_j increases M_j by at most about a factor of $\frac{3}{2}$. Because at the start of the algorithm

$$\mu_{j,i}^2 \leq \left(\frac{3+i}{4}\right) B^j,$$

we have

$$\mu_{j,i}^2 \leq \left(\frac{3}{2}\right)^{2(j-1)} \left(\frac{3+i}{4}\right) B^j$$

during the size-reduction.

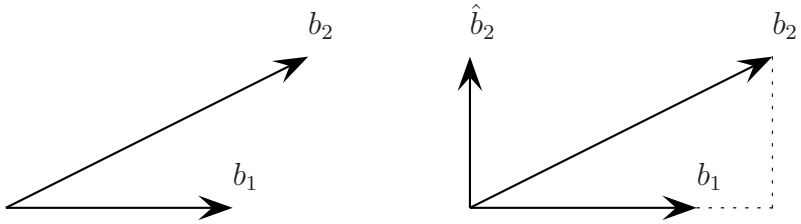
To summarize, b_j, $\hat{b}_j$ and $\mu_{j,i}$ can all be represented as rational numbers with numerators and denominators of length $\mathcal{O}(n \log B)$.

■

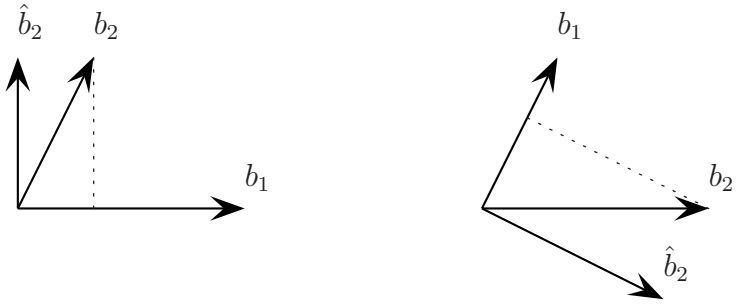
REMARK 6.1 The number of arithmetic operations required by the LLL-reduction algorithm is actually $\mathcal{O}(n^4 \log B)$, if we keep track of all of the $\mu_{k,j}$'s all of the time and update the values from previous values, rather than using the definitions and inner products.

AN LLL EXAMPLE

To get a better understanding of the behaviour of the LLL-algorithm we present a small example. We consider a basis b_1, b_2 where $b_1 = (3, 0)$ and $b_2 = (4, 2)$. Here we use $\delta = 0.9$.



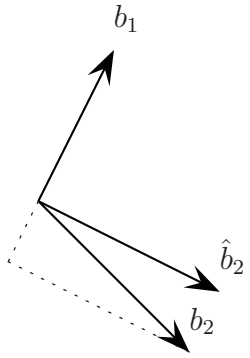
We have $\mu_{2,1} = 1\frac{1}{3}$ and $\hat{b}_2 = (0, 2)$. This means that if we subtract b_1 from b_2 then the basis will get shorter.



Now $b_2 = (1, 2)$ and $\mu_{2,1} = \frac{1}{3}$. Therefore the basis is size-reduced. However, we must swap vectors b_1 and b_2 because

$$0.9 \times 9 > \frac{1}{3^2} \times 9 + 4.$$

This means that b_2 is considerably closer than b_1 to the subspace generated by the previous basis vectors (in this case, 0). Now we have $\mu_{2,1} = \frac{3}{5}$ and $\hat{b}_2 = (2\frac{2}{5}, -1\frac{1}{5})$. Again, we must subtract b_1 from b_2 to make the basis shorter.



We have $b_2 = (2, -2)$ and $\mu_{2,1} = -\frac{2}{5}$. The basis is size-reduced and b_1 is shorter than b_2:

$$0.9 \times 5 \leq \frac{2^2}{5^2} \times 5 + 7\frac{1}{5}.$$

Basis b_1, b_2 is now LLL-reduced and the candidate for the shortest vector is b_1.

DEEP INSERTIONS

There exist several improvements to the LLL-algorithm. For example, the so-called deep insertion of b_k; see [37].

Let us consider the following basis: $b_1 = (16, 0, 0)$, $b_2 = (8, 14, 0)$ and $b_3 = (0, 0, 15)$. This basis is LLL-reduced for any δ and $\hat{b}_2 = (0, 14, 0)$ and $\hat{b}_3 = (0, 0, 15)$. However, we would like to find a basis where the first vector is as short as possible. In this case this is b_3, b_1, b_2. This is also an LLL-reduced basis of the same lattice. In the LLL-algorithm we consider only the swapping of two consecutive basis vectors. In the deep insertion variant we check whether swapping the basis vector with any of the previous ones gives us a better basis.

The improved version of the algorithm is as follows:

$k = 2$
While $k \leq n$ **do**
    size-reduce the vector b_k and update μ_{kj} for $j = 1, \ldots, k-1$
    $j = 1$
    While $j < k$ **do**
        If $\delta||\hat{b}_j||^2 > \sum_{i=j}^{k-1} \mu_{k,i}^2||\hat{b}_i||^2 + ||\hat{b}_k||^2$
            Then $(b_i, b_{i+1}, \ldots, b_k) = (b_k, b_i, \ldots, b_{k-1})$
                $k = \max(j-1, 1)$
            Else $j = j + 1$
    $k = k + 1$
Return $b_1, \ldots, b_n$

This algorithm cannot be shown to be polynomial time using Theorem 13. Let us consider the loop invariant

$$D = \prod_{i=1}^{n-1} ||\hat{b}_i||^{2(n-i)}$$

in the proof of Theorem 13. With the normal LLL-algorithm, D decreases with every swap. This is not the case with the deep insertions version. Let us again consider the basis b_1, b_2, b_3. Here $D = 16^4 \times 14^2 = 3584^2$. After deep insertion, the basis is b_3, b_1, b_2 and $D = 15^4 \times 16^2 = 3600^2$. Thus the proof of Theorem 13 cannot be applied to this algorithm.

However, if we specify that, for some constant c, either $j < c$ or $k - j < c$, then the deep insertions version of the algorithm can be shown to be polynomial time.

6.2 BLOCK KORKIN-ZOLOTAREV REDUCTION

It turns out that, in practice, the LLL-reduction is not efficient enough to find the shortest vector when n is large. We now consider a slightly different reduction method from [42].

Let

$$\pi_i : \mathbb{R}^n \to \text{span}(b_1, b_2, \ldots, b_{i-1})^{\perp}$$

denote the orthogonal projection so that

$$v - \pi_i(v) \in \text{span}(b_1, b_2, \ldots, b_{i-1})$$

for any vector v.

REMARK 6.2 We note that

$$\pi_i(b_j) = \sum_{s=i}^{j-1} \mu_{j,s}\hat{b}_s + \hat{b}_j$$

because

$$b_j - \sum_{s=i}^{j-1} \mu_{j,s}\hat{b}_s - \hat{b}_j = \sum_{s=1}^{i-1} \mu_{j,s}\hat{b}_s \in \text{ span}(b_1, b_2, \ldots, b_{i-1}).$$

Using these notions we can write condition (6.2) as

$$\delta||\pi_i(b_i)|| \leq ||\pi_i(b_{i+1})|| \quad i = 1, \ldots, n-1.$$

If L is a lattice then the set $\pi_i(L)$ is also a lattice. First we show that the mapping π_i is linear.

Let $a, b \in \mathbb{R}$ and $u, v \in \mathbb{R}^n$. We show that

$$a\pi_i(u) + b\pi_i(v) = \pi_i(au + bv)$$

by proving that

$$\begin{aligned} a\pi_i(u) + b\pi_i(v) - \pi_i(au + bv) \in &\text{span}(b_1, b_2, \ldots, b_{i-1}) \\ &\cap \text{ span}(b_1, b_2, \ldots, b_{i-1})^{\perp}. \end{aligned}$$

We have

$$\begin{aligned} & au + bv - \pi_i(au + bv) \in \text{ span}(b_1, b_2, \ldots, b_{i-1}) \\ \Rightarrow \quad & a(u - \pi_i(u)) + a\pi_i(u) + b(v - \pi_i(v)) + b\pi_i(v) - \pi_i(au + bv) \\ & \qquad \in \text{ span}(b_1, b_2, \ldots, b_{i-1}) \\ \Rightarrow \quad & a\pi_i(u) + b\pi_i(v) - \pi_i(au + bv) \in \text{ span}(b_1, b_2, \ldots, b_{i-1}) \end{aligned}$$

because $u - \pi_i(u), v - \pi_i(v) \in \text{ span}(b_1, b_2, \ldots, b_{i-1})$. But we also have

$$a\pi_i(u) + b\pi_i(v) - \pi_i(au + bv) \in \text{ span}(b_1, b_2, \ldots, b_{i-1})^{\perp}$$

and thus

$$a\pi_i(u) + b\pi_i(v) = \pi_i(au + bv).$$

If L has a basis $b_1, \ldots, b_n$ then for any $v \in \pi_i(L)$ we have

$$v = \pi_i(a_1 b_1 + \cdots + a_n b_n) = a_i\pi_i(b_i) + \cdots + a_n\pi_i(b_n).$$

On the other hand, all the $\pi_i(b_j)$ for $i \leq j \leq n$ are linearly independent. If they were not we would have for some $a_i \in \mathbb{R}$

$$\pi_i(b_j) = \sum_{s=i}^{j-1} a_s \pi_i(b_s),$$

which would mean that $b_j \in \text{span}(b_1, \ldots, b_{j-1})$. But this is not true as all vectors b_i are linearly independent. Thus $\pi_i(L)$ is a lattice with basis $\pi_i(b_i), \ldots, \pi_i(b_n)$.

A basis is a *Korkin-Zolotarev basis* if it is size-reduced and $\hat{b}_i$ is the shortest vector in the lattice $\pi_i(L)$. This means that b_1 is the shortest vector of the lattice L. If we could find a Korkin-Zolotarev basis of any lattice in a feasible time then we could also find the shortest vector of the lattice in a feasible time. Therefore it seems improbable that there is a fast algorithm for finding a Korkin-Zolotarev basis.

There is, however, an algorithm for finding the shortest vector in any lattice which runs in super-exponential time. It was presented by Kannan [23].

THEOREM 14 *The shortest nonzero vector of lattice L with an LLL-reduced basis $b_1, \ldots, b_n$ can be found after at most $n^{\frac{n}{2}+o(n)}$ arithmetic operations.*

Proof. We denote by L_2 the lattice with basis $b_2, \ldots, b_n$. Let us assume first that $||b_1|| \leq \lambda_1(L_2)$. All lattice vectors $v = \sum_{i=1}^{n} v_i b_i$ shorter than b_1 satisfy

$$||v||^2 = \sum_{j=1}^{n} \left(\sum_{k=j}^{n} v_k \mu_{k,j} \right)^2 ||\hat{b}_j||^2 < ||b_1||^2.$$

As every term on the left hand side is non-negative and $\mu_{j,j} = 1$, we get

$$\left(v_j + \sum_{k=j+1}^{n} v_k \mu_{k,j} \right)^2 ||\hat{b}_j||^2 < ||b_1||^2$$

for all $1 \leq j \leq n$. Further, we have

$$\sum_{k=j+1}^{n} v_k \mu_{k,j} - \frac{||b_1||}{||\hat{b}_j||} < v_j < \sum_{k=j+1}^{n} v_k \mu_{k,j} + \frac{||b_1||}{||\hat{b}_j||}.$$

Thus, there are at most $\left\lfloor 2\frac{||b_1||}{||\hat{b}_j||} \right\rfloor + 1$ possible integer values for every v_j, when $(v_{j+1}, \ldots, v_n)$ is fixed.

If $||\hat{b}_j|| > 2||b_1||$, there is only one possibility for v_j so the basis vector b_j can be omitted when we count the number of possibilities. So, without loss of generality, we can assume that we always have that $||\hat{b}_j|| \leq 2||b_1||$.

Now in total at most

$$\prod_{j=1}^{n} \left(\left\lfloor 2\frac{||b_1||}{||\hat{b}_j||} \right\rfloor + 1 \right)$$

choices for $(v_1, \ldots, v_n)$ exist. Because $\frac{||b_1||}{||\hat{b}_j||} \geq \frac{1}{2}$, this number of choices is at most

$$\begin{aligned} 4^n \prod_{j=1}^{n} \frac{||b_1||}{||\hat{b}_j||} &\leq 4^n \frac{\lambda_1(L_2)^{n-1}}{\det(L_2)} \\ &\leq 4(16\gamma_{n-1})^{\frac{n-1}{2}} \\ &\leq 4(16n)^{\frac{n-1}{2}} = n^{\frac{n}{2}+o(n)}. \end{aligned}$$

Here the first inequality is due to the assumption that $||b_1|| \leq \lambda_1(L_2)$, the second to the definition of γ_n, and the third to the fact that $\gamma_n \leq n$.

Similarly, if we have that $||b_1|| > \lambda_1(L_2)$ then the number of choices for $(v_1, v_2, \ldots, v_n)$ is at most

$$\begin{aligned} \prod_{j=1}^{n} \left(\left\lfloor 2\frac{\lambda_1(L_2)}{||\hat{b}_j||} \right\rfloor + 1 \right) &\leq 4^n \frac{\lambda_1(L_2)}{||b_1||} \frac{\lambda_1(L_2)^{n-1}}{\det(L_2)} \\ &\leq 4(16n)^{\frac{n-1}{2}} = n^{\frac{n}{2}+o(n)}. \end{aligned}$$

The theorem follows when we note that we need n^2 multiplications to compute $||v||$. ■

The above theorem describes an algorithm for finding the shortest vector in a lattice. First we find an LLL-reduced basis and then we find recursively the shortest vector of the lattice $L(b_i, \ldots, b_n)$ with the help of the shortest vector of $L(b_{i+1}, \ldots, b_n)$. There is always a limited number of candidates for the shortest vector. Unfortunately this algorithm does not work in polynomial time. To make the algorithm more efficient and usable in practice we can relax somewhat the requirements.

A basis is called *block reduced with block size* β if it is size-reduced and there is no non-zero vector in the lattice $\pi_i(L(b_i, \ldots, b_{\min(i+\beta-1,n)}))$ shorter than $\delta||\hat{b}_i||$ for some $\delta < 1$.

Here we are required to find the shortest vector in the lattice generated by β basis vectors. This vector can be found using the algorithm of Kannan described above. Because β is constant the algorithm of Kannan finds the shortest vector in polynomial time.

We do not find a block reduced basis just by finding the shortest vector of a lattice of dimension β n times. Every time we find a new shortest vector to our basis, the previous $\beta - 1$ basis vectors may no longer be the shortest vectors in the lattice generated by their next $\beta - 1$ basis vectors.

The following algorithm by Schnorr and Euchner [42] finds a basis that is block reduced with block size β.

LLL$(b_1, \ldots b_n)$
$z = 0$
$j = 0$
While $z < n - 1$ **do**
  $j = j + 1$
  $k = \min(j + \beta - 1, n)$
  If $j = n$ **Then** $j = 1$
    $k = \beta$
  Using the Algorithm of Kannan Find Shortest

$$v = \sum_{i=j}^{k} v_i \pi_j(b_i)$$

  Set $b^{\text{new}} = \sum_{i=j}^{k} v_i b_i$
  $h = \min(k + 1, n)$
  If $\delta ||\pi_j(b_j)||^2 > ||v||^2$
  Then LLL$(b_1, \ldots b_{j-1}, b^{\text{new}}, b_j, \ldots, b_h)$ **starting from** b_j
    $z = 0$
  Else LLL$(b_1, \ldots b_h)$ **with** $\delta = .99$ **starting from** b_{h-1}
    $z = z + 1$

In the algorithm, j goes cyclically through all integers $1, \ldots, n - 1$ until

$$\delta ||\hat{b}_j||^2 \leq \lambda_1(\pi_j(L(b_j, b_{j+1}, \ldots, b_k)))^2 \tag{6.11}$$

for $n - 1$ consecutive values of j. When the inequality (6.11) holds for all j the basis is block reduced. The number of consecutive successes is counted by variable z.

If condition (6.11) does not hold for j then we add the smaller vector b^{new} in the set of basis vectors before b_j. The new set of basis vectors is now linearly dependant and we use the LLL-algorithm to find a new linearly independent basis. Note that the basis vectors $b_1, b_2, \ldots, b_{j-1}$ remain unchanged and so are LLL-reduced and we know the values of $\mu_{i,k}$, $0 \leq i, k \leq j-1$. After this the number of basis vectors is again reduced to n.

If condition (6.11) holds then we use the LLL-algorithm for the vectors $b_1, b_2, \ldots, b_h$ to make sure that the basis used by the algorithm of Kannan at the next step is LLL-reduced.

The next theorem gives us information on the size of the basis vectors we get using block reduction. The proof is from a paper [38] by Schnorr.

LEMMA 3 *For every block size β block reduced basis $b_1, \ldots, b_n$ with $n \geq \beta$ we have for $M = \max(||\hat{b}_{n-\beta+2}||, \ldots, ||\hat{b}_n||)$ that $||b_1|| \leq \gamma_\beta^{\frac{n-1}{\beta-1}} M$.*

Proof. We extend the basis $b_1, \ldots, b_n$ by $\beta - 2$ additional vectors to

$$b_{-\beta+3}, \ldots, b_{-1}, b_0, b_1, \ldots, b_n$$

so that the new vectors are orthogonal to all basis vectors, and all of the new vectors are of the same length as b_1. This basis is still block reduced with block size at least β. Thus $\hat{b}_i$ is the shortest vector in the lattice $\pi_i(L(b_i, \ldots, b_{i+\beta-1}))$ and

$$\det(L(b_i, \ldots, b_{i+\beta-1})) = ||\hat{b}_i||||\hat{b}_{i+1}|| \cdots ||\hat{b}_{i+\beta-1}||$$

for $i = -\beta+3, \ldots, n-\beta+1$. By definition of the Hermite constant γ_β we have

$$||\hat{b}_i||^\beta \leq \gamma_\beta^{\frac{\beta}{2}} ||\hat{b}_i||||\hat{b}_{i+1}|| \cdots ||\hat{b}_{i+\beta-1}|| \quad \text{for } i = -\beta+3, \ldots, n-\beta+1.$$

When we multiply all of these $n-1$ inequalities together we get

$$\begin{aligned} ||\hat{b}_{-\beta+3}||^\beta ||\hat{b}_{-\beta+4}||^\beta \cdots ||\hat{b}_{n-\beta+1}||^\beta \leq \\ \gamma_\beta^{\frac{\beta(n-1)}{2}} ||\hat{b}_{-\beta+3}||^1 ||\hat{b}_{-\beta+4}||^2 \cdots ||\hat{b}_1||^{\beta-1} ||\hat{b}_2||^\beta \cdots ||\hat{b}_{n-\beta+1}||^\beta \\ ||\hat{b}_{n-\beta+2}||^{\beta-1} \cdots ||\hat{b}_{n-1}||^2 ||\hat{b}_n||^1. \end{aligned}$$

We can simplify this to

$$\begin{aligned} ||\hat{b}_{-\beta+3}||^{\beta-1} ||\hat{b}_{-\beta+4}||^{\beta-2} \cdots ||\hat{b}_0||^2 ||\hat{b}_1||^1 \leq \\ \gamma_\beta^{\frac{\beta(n-1)}{2}} ||\hat{b}_{n-\beta+2}||^{\beta-1} \cdots ||\hat{b}_{n-1}||^2 ||\hat{b}_n||^1. \end{aligned}$$

Here the vectors on the left hand side of the equation are all of the same length and those on the right hand side are all of length at most M. We get

$$||b_1||^{\frac{\beta(\beta-1)}{2}} \leq \gamma_\beta^{\frac{\beta(n-1)}{2}} M^{\frac{\beta(\beta-1)}{2}}$$

and thus

$$||b_1|| \leq \gamma_\beta^{\frac{n-1}{\beta-1}} M.$$

■

Using this lemma we get a limit for the size of the block reduced basis vectors.

THEOREM 15 *Every block size β block reduced basis $b_1, \ldots, b_n$ of lattice L satisfies*

$$\lambda_i^2 \leq \gamma_\beta^{2\frac{i-1}{\beta-1}} \frac{i+3}{4} ||b_i||^2$$

for $i = 1, \ldots, n$.

Proof. We have by definition

$$\lambda_i^2 \leq \max(||b_1||^2, \ldots, ||b_i||^2).$$

It follows from the inequality $||b_j||^2 \leq ||\hat{b}_j||^2 + \sum_{k=1}^{j-1} \mu_{j,k}^2 ||\hat{b}_k||^2$ that

$$\begin{aligned} \lambda_i^2 \leq & \left(1 + (i-1)\frac{1}{2^2}\right) \max(||\hat{b}_1||^2, \ldots, ||b_i||^2) \\ \leq & \frac{i+3}{4} \max(||\hat{b}_1||^2, \ldots, ||\hat{b}_i||^2). \end{aligned} \tag{6.12}$$

Lemma 3 applied to a basis $\pi_j(b_j), \ldots, \pi_j(b_i)$ yields the inequalities

$$||\hat{b}_j|| \leq \gamma_\beta^{\frac{i-j}{\beta-1}} \max(||\hat{b}_{i-\beta+2}||^2, \ldots, ||\hat{b}_i||^2) \tag{6.13}$$

for $1 \leq j \leq i - \beta + 1$. For $i - \beta + 2 \leq j \leq i$, we have

$$||\hat{b}_j|| \leq ||\pi_j(b_i)|| \leq ||b_i|| \tag{6.14}$$

because the basis was block reduced with block size $\beta > i - j$. From this and (6.13) we see that the inequality

$$||\hat{b}_j|| \leq \gamma_\beta^{\frac{i-j}{\beta-1}} ||b_i|| \tag{6.15}$$

holds for $1 \leq j \leq i - \beta + 1$. Because $\gamma_\beta \geq 1$, (6.14) implies that (6.15) also holds for $i - \beta + 2 \leq j \leq i$. Applying (6.15) to (6.12) proves the theorem. ■

There is a relationship between block reduced and LLL-reduced bases.

THEOREM 16 *Let $\frac{1}{3} \leq \delta \leq 1$. A basis $b_1, b_2, \ldots, b_n \in \mathbb{R}^n$ is reduced with block size 2 if and only if it is LLL-reduced.*

Proof. Assume first that a basis $b_1, b_2, \ldots, b_n$ is reduced with block size 2. Then

$$\delta||\hat{b}_k||^2 \leq ||\pi_k(v_k b_k + v_{k+1} b_{k+1})||^2$$

for all $(v_k, v_{k+1}) \in \mathbb{Z}^2 \setminus (0,0)$, where $1 \leq k \leq n-1$. This implies that

$$\delta||\hat{b}_k||^2 \leq ||\pi_k(b_{k+1})||^2 = ||\mu_{k+1,k}\hat{b}_k + \hat{b}_{k+1}||^2.$$

Hence $b_1, b_2, \ldots, b_n$ is LLL-reduced.

Conversely, we assume that $b_1, b_2, \ldots, b_n$ is LLL-reduced and show that the inequality

$$||\pi_k(v_k b_k + v_{k+1} b_{k+1})||^2 = (v_k + \mu_{k+1,k} v_{k+1})^2 ||\hat{b}_k||^2 + v_{k+1}^2 ||\hat{b}_{k+1}||^2 \geq \delta||\hat{b}_k||^2$$

holds for all $(v_k, v_{k+1}) \in \mathbb{Z}^2 \setminus (0,0)$. We have three cases. If $v_{k+1} = 0$ then $|v_k| \geq 1$ and the inequality holds. If $v_{k+1} = \pm 1$ then $|\mu_{k+1,k} v_{k+1}| \leq \frac{1}{2}$ and the minimal value for the left hand side occurs when $v_k = 0$. Now the inequality is of the form

$$\mu_{k+1,k}^2 ||\hat{b}_k||^2 + ||\hat{b}_{k+1}||^2 \geq \delta||\hat{b}_k||^2,$$

which holds because the basis is LLL-reduced. If $|v_{k+1}| \geq 2$, the lower bound follows from the inequalities

$$4||\hat{b}_{k+1}||^2 \geq 4(\delta - \frac{1}{4})||\hat{b}_k||^2 \geq \delta||\hat{b}_k||^2.$$

Here the first inequality follows from the fact that the basis is LLL-reduced and the second inequality holds because $\delta \geq \frac{1}{3}$.

■

The algorithm for calculating the block Korkin-Zolotarev reduction is not known to be polynomial. However, if we need to find the smallest vector of the lattice, it usually gives the result faster than LLL.

We can further speed up the calculations using *block Korkin-Zolotarev reduction with the Schnorr-Hörner pruning heuristic* [41]:

Suppose that we have a lattice L with basis $b_1, \ldots, b_n$ and that we are trying to find a vector $v = \sum_{i=1}^n c_i b_i$ shorter than length c. Let us

assume that we have gone through basis vectors $b_{t+1}, \ldots, b_n$ and fixed integers $v_{t+1}, \ldots, v_n$. We denote $L_2 = L(b_1, \ldots, b_t)$,

$$u_1 = \sum_{j=t+1}^{n} \left(\sum_{k=j}^{n} v_k \mu_{k,j} \right) \hat{b}_j$$

and

$$u_2 = \sum_{j=1}^{t} \left(\sum_{k=t+1}^{n} v_k \mu_{k,j} \right) \hat{b}_j.$$

At this point, our objective is to find a point w in lattice L_2 such that $v = w + u_2 + u_1$ and $||v|| < c$. We have $w, u_2 \in \text{span}(L_2)$ and $u_1 \in \text{span}(L_2)^{\perp}$. Thus

$$||v||^2 = ||w + u_2||^2 + ||u_1||^2$$

and

$$||w + u_2|| < \sqrt{c^2 - c_1^2},$$

where we denote $c_1 = ||u_1||$.

Let $S(r)$ be the volume of a t-dimensional ball with radius r. The Gaussian heuristic states that there are about

$$\frac{\text{vol}(S(\sqrt{c^2 - c_1^2}))}{\det(L_2)}$$

points w in L_2 that are closer than $\sqrt{c^2 - c_1^2}$ to the point $-u_2$.

The pruning means that if this ratio is smaller than 2^{-p} for a selected pruning parameter p, we decide that we cannot find a short vector with the chosen $v_{t+1}, \ldots, v_n$. In other words, if finding a vector short enough becomes too improbable, we retreat in the recursion and try to find it later with different integers $v_{t+1}, \ldots, v_n$.

The algorithm for this and the basic BKZ-reduction can be found in [42].

CHAPTER 7

Computational results

In the following our aim is to approximate the time taken to find out the private key of NTRUEncrypt when the parameter set ees251ep4 [9] is used. According to this parameter set we have $N = 251$, $q = 239$, $p = 2$ and $d_f = d_g = 72$. To this end we generated some smaller instances of NTRU-lattices with the same characteristics and studied them.

The key characteristics of the NTRU-lattice

$$L = \begin{pmatrix} I & \overrightarrow{h} \\ 0 & qI \end{pmatrix}$$

with dimension $2N$ are

$$a = \frac{N}{q}$$

and

$$c = \frac{\lambda(L)}{\sigma(L)}\sqrt{\dim(L)},$$

where $\sigma(L)$ denotes the expected length of the shortest vector of lattice L obtained using the Gaussian heuristic, and $\lambda(L)$ denotes the actual shortest vector of lattice L which gives us the private key. Using the parameter set ees251ep4 and trick 3B of Chapter 5, the values of these lattice constants are $a \approx 1.05$ and $c \approx 2.7$.

For any N we can now find q, d_f and d_g with which we can generate NTRU keys that correspond to an NTRU-lattice with dimension $2N$ and with the same lattice constants as in the real case. Furthermore, we select these parameters so that q is odd and $d_f = d_g$ is even. For instance, with

$N = 100$, we choose $q = 95$ and $d_F = d_g = 30$. Now $a \approx 1.053$ and

$$c \approx \frac{\sqrt{2\left((N-d)\cdot 1^2 + d\cdot 3^2\right)}}{4\sqrt{\frac{Nq}{\pi e}}}\sqrt{2N} = \sqrt{\frac{(70+30\cdot 9)\pi e}{4\cdot 95}} \approx 2.764.$$

Our procedure was as follows: first we generated the small NTRU-like private and public keys randomly, five instances of every size. Then we used trick 3B on the NTRU-lattice of every instance. Finally we solved the shortest vector problem of every lattice using the Block Korkin-Zolotarev reduction. We always started with a small block size and increased it until the resulting basis revealed the private key. If the block size was too small, the small vector revealed was usually one of the original basis vectors – a vector with one component equal to $4q$ and the rest equal to zero. It is conjectured, based on extensive tests at NTRU, that the smallest successful block size increases linearly with the dimension (when the lattice constants remain the same).

The reductions have been computed using the NTL C-library and a 733MHz Pentium computer. The results are shown in table 7.1. The times given are for reduction with the smallest successful block size. We omitted the time waisted on unsuccessful tries when the block size was too small.

We have plotted the logarithms of the breaking times of these small instances in figure 7.1. It has been conjectured by the researches at NTRU that these points are approximately on a straight line. This seems more or less to be the case also here. If we are convinced that the instances with $N = 251$ are also on this line we can approximate how hard it would be to break the private key with this method.

We can calculate from the data that the equation of the regression line is

$$\log_{10} T = 0.1749 \times N - 14.15.$$

When $N = 251$, the breaking time would be about 5.6×10^{29} seconds with this computer. We can transform this into MIPS-years by multiplying by 733 and dividing by the number of seconds in a year. The result is about 1.3×10^{25} MIPS-years.

We have also tried different versions of Block Korkin-Zolotarev reduction. The best variant, according to our tests, used Givens rotations (see [13]) instead of Gram-Schmidt orthogonalization. The parameter δ was 0.99.

One way to improve the performance of the reduction is to use some pruning constant. With a small pruning constant, 6, the algorithm indeed

N	b_1	T_1	b_2	T_2	b_3	T_3	b_4	T_4	b_5	T_5
100	20	2000	22	6500	21	1500	21	2400	21	1500
101	22	2400	20	1000	22	10 000	21	7400	21	4000
102	22	9600	22	8500	22	580	22	7200	22	33 000
103	23	10 400	23	21 000	23	29 000	21	5000	18	800
104	22	7800	22	11 000	22	15 000	22	5400	23	7800
105	22	16 000	23	5900	23	37 000	23	17 000	22	11 000
106	23	80 000	23	15 000	22	23 000	22	53 000	22	24 000
107	22	5300	23	40 000	22	36 000	22	38 000	23	59 000
108	24	33 000	23	156 000	23	122 000	22	5300	22	12 000
109	24	140 000	23	30 000	23	60 000	23	180 000	23	100 000
110	22	33 000	24	550 000	22	6000	24	210 000	23	200 000
111	23	260 000	23	8500	24	440 000	24	470 000	24	1100 000
112	23	450 000	23	230 000	24	1200 000	24	370 000	24	720 000
113	25	1400 000	23	13 000	24	260 000	24	190 000	24	1200 000
114	24	1700 000	24	1300 000	23	190 000	23	220 000	24	1500 000
115	24	980 000	25	1600 000	25	4100 000	24	630 000	25	590 000

TABLE 7.1: A table of the block size needed to reveal the secret key and the time taken in seconds for five different instances for each size.

gave us the result faster once the right block size was found. We noticed that the block size increased rapidly with the dimension. Furthermore, due to the more erratic nature of the algorithm, block sizes larger than the smallest successful one sometimes failed to give the result. We discarded this method because finding a successful block size for which the answer was returned quickly took too many tries. With a pruning constant of 10, the algorithm behaved almost as if there had been no pruning and no increase in speed was detected.

The researchers at NTRU performed a similar experiment to approximate the breaking time. They solved several instances of NTRU lattices with lattice constants $c = 1.73$ and $a = 0.535$. These constants are smaller than those used by us. They used the same method as we did to approximate the breaking time of an NTRU lattice of size $N = 251$. In general, the reduction terminates faster for lattices when either of the lattice constants is reduced in size. Thus their approximation works as a lower limit for a lattice with larger lattice constants. Their result for the breaking time was about 7×10^{14} MIPS-years.

Furthermore, they employed the zero-forcing methods to reduce the lower limit of the breaking time to 1.37×10^{13} MIPS-years. This result

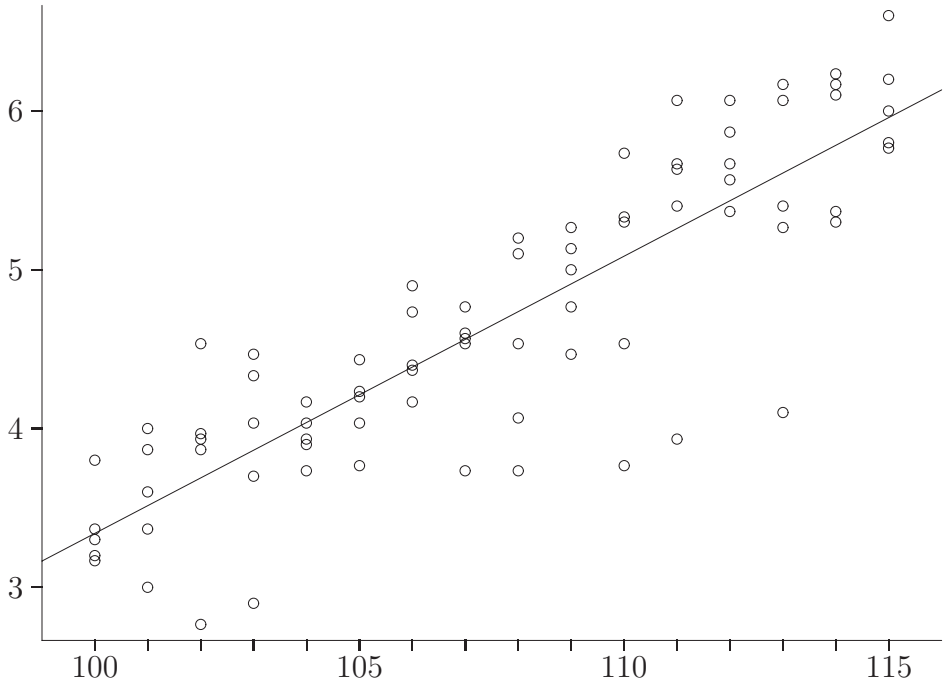


FIGURE 7.1: Base 10 logarithms of breaking times for $N = 100 \ldots 115$ and the linear regression line.

requires that all of the rotations of the short vector are also in the lattice. However, this is not the case here. As we noted earlier this is true only before trick two or using lattices derived from h^{-1}. In these cases the lattice constants are even larger than in our experiments.

REMARK 7.1 The matrix of trick 3B has one more row than column. We simply omit the first row to get a square matrix and a lattice corresponding to it. This means that we have guessed that the constant coefficient of the private key f is 1. The probability of this guess being correct is $\frac{179}{251} \approx 71\%$. Alternatively, we could have guessed that the constant coefficient equals 3 and combined the first and last row into one row. Strictly speaking we should multiply our speed estimates by a small constant greater than one.

REMARK 7.2 Before applying the reduction algorithm we randomized the order of the rows. This seemed to speed up the process.

CHAPTER 8

Attacks against NTRU

In this chapter we present some attacks against different versions of NTRU. We outline two chosen cryptotext attacks against previous versions of NTRU as well as the most well-known attack against any NTRU version. First we consider how the parameter N, the degree of the reducing polynomial, must be chosen.

If the parameter N were to be selected as a power of 2, then we could use the Fast Fourier Transformation to compute the convolution product. But in article [11] Gentry showed that N must be a prime. If d divides N we define

$$f_{(d)} = \sum_{i=0}^{d-1} \left(\sum_{j=0}^{\frac{N}{d}-1} f_{dj+i} \right) x^i.$$

Clearly $f_{(d)} \in \mathbb{Z}[x]/(x^d-1)$. If $f*h = g$ then also $f_{(d)}*h_{(d)} = g_{(d)}$. Instead of a lattice of dimension $2N$ we can now try to find a short vector in a lattice of dimension $2d$ defined by the matrix

$$\begin{pmatrix} I_{(d)} & h_{(d)} \\ 0 & qI_{(d)} \end{pmatrix}.$$

A short vector in this lattice gives us significant partial information about the private key.

8.1 ATTACK CONTROLLING THE CRYPTOTEXT

The message, or the message representative, did not have enough padding in the first versions of NTRU. As a consequence, it was vulnerable to

chosen-ciphertext attacks. Several attacks and padding improvements were presented in [22, 32, 14].

The basic idea of these attacks was simple. Decryption failures leak out information. Let us suppose that the encryptor finds out whether a decryption is a success or not. A decryption failure means that not all of the coefficients of $f * e$ are in the desired interval. Because the encryptor knows and can even choose e, (s)he can gain information about f. For example, if (s)he can choose almost any e, then (s)he can make small changes to it until the decryption fails. When (s)he finds several cryptotexts similar to each other, only some of which can be decrypted, (s)he can deduce the private key f.

Let us say that for some e the decryption is a success and for $e' = e+1$ it fails. This means that $f * e$ has at least one coefficient very close to one of the limits of the interval. Suppose that the coefficient of x^ℓ was large. Because the decryption of e' is taken to fail, the polynomial

$$f * e' = f * e + f$$

has at least one coefficient that is not in the interval. With high probability this is the coefficient of x^ℓ. This means that the coefficient f_ℓ is larger than zero. Now the success of decryptions of $e+x^j$ reveals whether the $f_{\ell+j}$ is larger than zero or not.

To render this attack impossible the generation procedure of message representatives was changed.

The main idea is that the decryption never succeeds if the encryption is not of the correct form. Therefore the attacker cannot select the cryptotext e freely, making the previous attack impossible.

8.2 ATTACK CONTROLLING THE MESSAGE REPRESENTATIVE

There are several attacks even against the revised version of NTRU. In late 2002, attacks were published by at least three different groups [18, 35, 31] (two of these papers were later combined [17]).

The main idea of the attacks is as follows. The difference between the largest and the smallest coefficient of $\alpha = f * i + p * r * g$ is on average 62 when $N = 251$. The attacker can cause a wrap failure if (s)he has the means to add about 33 to the largest coefficient of α. Note that the largest coefficient of α is on average $A+31$ and a coefficient of size $A+\frac{128}{2}$ causes a wrap failure.

Because we need to be able to find out m and b from i at the decryption phase, the mapping $\varphi(m, h, b)$ must be invertible. The attacker uses this property to find a message m and a randomness b that generate the message representative i which (s)he wants to use. In effect (s)he can choose the i in α.

In their attack, the authors of [17] use message representatives that have the same zero coefficients as the polynomial $(1+x)*D*F_{15}$, where $D \in T(4,0)$ and $F_{15} \in T(15,0)$. The message representative is always binary but $(1+x)*D*F_{15}$ does not need to be. We have that

$$f * i \approx (1+x)*(2+x)*F*D*F_{15}.$$

Let us consider $F*F_{15}$. The constant coefficient of this product is large if F and $F_{15}(x^{-1})$ have several common nonzero coefficients. If all nonzero coefficients of $F_{15}(x^{-1})$ match nonzero coefficients of F, then one of the coefficients of $3x*F*F_{15}$ is at least 45. With high probability this causes a wrap error.

In the attack we try to find polynomials F_{15} that generate wrap errors with several different polynomials D. In this way we can make sure that the wrap errors occur because of the F_{15}. When we have found enough different polynomials F_{15} that generate wrap errors we combine the results to get the polynomial F.

8.3 MEET-IN-THE-MIDDLE ATTACK

In the so-called meet-in-the-middle attack (see [19]), the key is broken in time $\mathcal{O}(2^{\frac{M}{2}})$ if the private key is chosen from a space of 2^M elements.

The idea of the attack is as follows. We divide the private key space into two equally large parts; f_1 goes through the first part, and we store the results of multiplications f_1*h. Next we let f_2 go through the second of the two parts and check whether we have an f_1*h stored such that

$$f_1*h + f_2*h \equiv pg \mod q,$$

where g is a binary polynomial.

In practice we have $f = 1+2F$ and F with 72 coefficients set to 1. We set 36 of the first 126 coefficients of F_1 to be one and 36 of the last 125 coefficients of F_2 to be one. All other coefficient of F_1 and F_2 are zeroes. Now there always exists a rotation of F that is equal to some F_1+F_2. Note that the 1 in the formula of f causes us problems and we

do not necessarily have a rotation of f equal to $1 + 2F_1 + 2F_2$. But we do have $f * x^i = x^i + 2F_1 + 2F_2$ for some i.

There are $\binom{126}{36}$ different polynomials F_1. In the first step we go through all of these polynomials and multiply them by h. We save some storage space if we do not store the result, but instead store only the most significant bit of the first k coefficients. We call the place where the bits of $F_1 * h$ are stored the *bin* of $F_1 * h$.

We are interested in the polynomials F_1 and F_2 with the property that $F_1 * h + (F_2 + \frac{1}{2}x^i) * h$ is a binary polynomial.

In the second step we go through all $i, 127 \leq i \leq 250$, for all possible polynomials F_2 and calculate $-(F_2+\frac{1}{2}x^i)*h$. If in the first step we found an $F_1 * h$ belonging to this bin, then we check whether $2F_1 + 2F_2 + x^i$ is a rotation of the private key. This is done by checking whether the polynomial $F_1 * h + (F_2 + \frac{1}{2}x^i) * h$ is binary. We also need to check the polynomials $F_1 * h$ from the bins corresponding to any of the 2^k polynomials we get by adding one to any of the first k coefficients of $-(F_2 + \frac{1}{2}x^i) * h$.

REMARK 8.1 Let the first four coefficients of $-(F_2 + \frac{1}{2}x^i) * h$ be $238, 7, 127$ and 144. This polynomial belongs to bin $(1, 0, 0, 1)$ if $k = 4$. We also need to check the polynomials $F_1 * h$ in bins $(0, 0, 0, 1)$, $(0, 0, 1, 1)$ and $(1, 0, 1, 1)$ because, for example,

$$(0, 7, 128, 144) - (238, 7, 127, 144) \equiv (1, 0, 1, 0) \mod 239.$$

Let us now estimate the running time and memory consumption of this attack. We denote by T_c the time it takes to calculate $f_1 * h$ and by T_l the time it takes to find f_1 from some bin or to store f_1 in a bin.

The expected running time of the first step is

$$T_1 = \binom{126}{36}(T_c + T_l).$$

In the second step we do the convolution, check a certain number of bins and, if some of the bins are nonempty, recalculate the corresponding convolution products. The expected number of bins to check is $\left(1 + \frac{2}{239}\right)^k$ and the expected size of a bin is

$$\frac{\binom{126}{36}}{2^k}.$$

The expected running time of the second step is thus

$$T_2 = 125\binom{125}{36}\left(T_c + \left(1 + \frac{2}{239}\right)^k \left(T_l + \frac{\binom{126}{36}}{2^k}T_c\right)\right).$$

It can be seen that the larger the selected k, the faster the attack. On the other hand, with a larger k the memory consumption also increases. The storage of the first step requires at least

$$\binom{126}{36}k$$

bits of memory.

CHAPTER 9

An Attack against old NTRUEncrypt

Our attack is based on the observation that blinding polynomials r with large coefficients generate wrap errors more frequently than polynomials with small coefficients. Therefore, by a careful selection of pairs (m, b) (which determine r), one can increase the probability of wrap errors. This probability gives us useful information about the private key.

Throughout this chapter we assume that the parameter set ees251ep1 of [8] is used. Thus, $N = 251$, $p = 2+x$ and $q = 128$. The private key f is of the form $f = 1 + p * F$, where $F(1) = 72$ (and thus $f(1) = 217$). Both of these polynomials have non-negative coefficients. The polynomial g is binary with 72 1's: $g(1) = 72$.

We assume that the blinding polynomial is generated from three components: $r = r_1 * r_2 + r_3$, where each $r_i(1) = 8$ (algorithm 3.3.2.2 in [8]). Most of the coefficients of the r_i's are therefore 0, only a few equal 1. Coefficients larger than 1 are possible, but they are few and far between. It follows that the coefficients of r are also very small: $r(1) = 72$. Most coefficients are 0's and 1's (as the average is 72/251). However, larger values also appear, this time more frequently than in the r_i's. The main tools in our attack are polynomials which have a suitably large number of "large" coefficients (≥ 4 or 5, for instance).

As noted earlier, the wrap failure occurs if at least one coefficient of $\alpha = f * i + p * r * g$ differs by $q/2$ from the average. Our goal is to

- Generate pairs (m, b) so that the wrap error probability correlates with the coefficients of $p * g$;

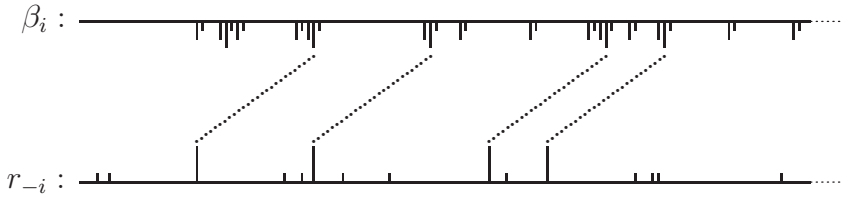


FIGURE 9.1: The four large coefficients of r match the large coefficients of β, resulting in a higher wrap error frequency.

- Increase the wrap error probability to such a level that the differences between distinct types of polynomials $p * g$ can be detected efficiently.

Our goals are achieved by generating pairs (m, b) such that the resulting blinding polynomial r has some large coefficients. In the following we denote $r = \sum r_i x^i$, $g = \sum g_i x^i$ and $\beta = p * g = \sum \beta_i x^i$. Clearly, $\beta_i \in \{0, 1, 2, 3\}$, and if we learn β_i then both g_i and g_{i-1} are revealed. For example, if $\beta_i = 3$ then $g_i = g_{i-1} = 1$. Also, the 72 indices i for which $\beta_i \geq 2$ are exactly those for which $g_i = 1$.

We increase the wrap probability by trying to make some coefficient of $\beta * r$ (and hence also α) exceptionally large. Suppose that $\beta_{u+i_j} = 3$ and that the coefficients r_{v-i_j} are large for some indices u, v and $i_1, \ldots, i_s$ (additions in subindices are reduced modulo N). Then all of the large coefficients r_{v-i_j} contribute partly to the same coefficient of $\beta * r$. More specifically, the coefficient of x^{u+v} in $\beta * r$ is at least $3 \sum_{j=1}^{s} r_{v-i_j}$. As the rest of the non-zero coefficients of r contribute to random terms in $\beta * r$, it is obvious that wrap failures become more frequent than normally.

The attack consists of four steps. In the first step we attempt to locate four 3's of β. After the second step we should have learned 15 3's of β. In the third step we spot the rest of β's coefficients that are at least 2, thus revealing g. The final step consists of computing the private key f from g and from the public key h.

In steps 1 and 2 we use blinding polynomials with four coefficients $r_i \geq 4$. For this purpose, denote by $\mathcal{M}^k_{4\times4}(i_0, \ldots, i_3)$ a set of k pairs (m, b) such that for some $u \in \{0, 1, \ldots, 250\}$ the coefficients $r_{u+i_j} \geq 4$ $(j = 0, 1, 2, 3)$, where $r = \rho(m, b)$. In step 3 we use blinding polynomials with one coefficient at least 5. A set of pairs (m, b) resulting in this kind of blinding polynomial is denoted by $\mathcal{M}_{1\times5}$.

All probabilities given below are estimates based on our implementations.

9.1 FIRST STEP

Our first goal is to locate four large coefficients of $\beta = p * g$ in one of its cyclic shifts $x^u * \beta$. For this purpose we encrypt messages in the sets $\mathcal{M}_{4\times4}^{1000}(i_0, i_1, i_2, i_3)$. Clearly, the quadruples (i_0, i_1, i_2, i_3) corresponding to four 3's of β induce a high wrap probability. In this case the large coefficients of r contribute to some term of $\beta * r$ by $4 \times 4 \times 3 = 48$, and therefore this term of α has a fair chance of exceeding the average by $q/2 = 64$, thus causing a wrap failure. Therefore the set $\mathcal{M}_{4\times4}^{1000}(i_0, i_1, i_2, i_3)$ that most frequently induces wrap failures most probably gives $\beta_{u-i_0} = \beta_{u-i_1} = \beta_{u-i_2} = \beta_{u-i_3} = 3$ (for some index u). The set-up is depicted in figure 9.1.

Luckily it is not necessary to go through all sets $\mathcal{M}_{4\times4}^{1000}(i_0, i_1, i_2, i_3)$. The average number of 3's in β is $72 \cdot \frac{71}{251} \approx 20$, and therefore a random guess of four indices has a reasonable chance (roughly 1%) of hitting four 3's in one of β's rotations. Therefore, if we randomly select, say, 1000 quadruples and of these select the quadruple which causes the most wrap failures, most probably we have found four 3's.

In our tests we made 1000 guesses for (i_0, i_1, i_2, i_3), generated 1000 encryptions for each such quadruple, and counted the number of wrap failures. The probability that this process found four 3's in β turned out to be 90%. In most of the erroneous cases we found three 3's and a 2. With 200 sets of 200 messages, the probability of success was $\frac{1}{6}$ and with a probability of 40% we found three 3's and a 2.

9.2 SECOND STEP

In the first step we learned i_0, i_1, i_2 and i_3 such that $\beta_{u-i_0} = \beta_{u-i_1} = \beta_{u-i_2} = \beta_{u-i_3} = 3$ for some u. Next we exploit these to find more large coefficients of β. More specifically, the goal is that after this step we will have found 15 3's (or possibly a combination of 15 3's and 2's) in β. Note that the probability that β has at least 15 3's is roughly 97%.

First we encrypt all of the messages in the sets $\mathcal{M}_{4\times4}^{1000}(i_0, i_1, i_2, k)$, $\mathcal{M}_{4\times4}^{1000}(i_0, i_1, i_3, k)$, $\mathcal{M}_{4\times4}^{1000}(i_0, i_2, i_3, k)$ and $\mathcal{M}_{4\times4}^{1000}(i_1, i_2, i_3, k)$ for all values of k, $0 \le k \le 250$, $k \neq i_j$. The wrap probability depends in this case on the coefficient β_{u-k}: the higher the wrap probability, the larger β_{u-k} is (see figure 9.2). We make the assumption that the 11 k's with the highest wrap probabilities give 11 large coefficients β_{u-k} (3's and possibly a few 2's among them).

This approach has one problem. It is possible that, for example,

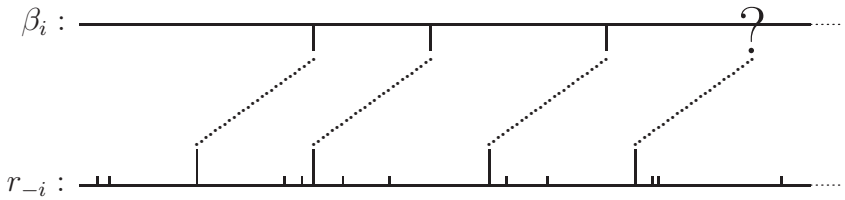


FIGURE 9.2: The four large coefficients of r match three known large coefficients of β and the position corresponding to k. The number of wrap errors at step 2 gives us information about the coefficient corresponding to k.

for some $v \neq u$ the coefficients of $x^{v-i_0}, x^{v-i_1}, x^{v-i_2}$ in β are also 3's. In this case there is a good chance that β_{u-k} is selected to be large, even though it is not. Fortunately we have a good chance of detecting such situations, as then the number of wrap errors generated by the $\mathcal{M}_{4\times4}^{1000}(i_0, i_1, i_2, k)$'s is greater than the number of wrap errors generated in the other three cases. If this is the case, we can replace $\mathcal{M}_{4\times4}^{1000}(i_0, i_1, i_2, k)$ by some $\mathcal{M}_{4\times4}^{1000}(i_0, i_1, i_4, k)$, where most probably $\beta_{u-i_4} = 3$ based on information from the other three cases.

Using the sets of 1000 messages, after finding four 3's in step 1, the sum of the 15 indices found in step 2 was in our tests always at least 41. On the other hand, if we found three 3's and a 2, then the sum of the 15 indices was at least 41 in half of our tests and at least 37 with a probability of 90%.

Using the sets of 200 messages and having found four 3's in step 1, the sum of the 15 indices was at least 41 with a probability of 70% and always at least 37. With three 3's and one 2, the sum was at least 37 with a probability of $\frac{1}{3}$.

9.3 THIRD STEP

We have now found 15 3's (or 2's) in $\beta * x^u$ for some u. Our strategy is to test the largeness of the remaining terms one at a time. We also test whether the 15 coefficients found earlier really are large.

Let β_{u-j} be the coefficient to be examined, and assume that the coefficients $\beta_{u-i_0}, \ldots, \beta_{u-i_{14}}$ are known to be large. We use blinding polynomials r with one peak: $r_v \geq 5$ for some v. Moreover, we assume that $\sum_{\ell=0}^{14} r_{v-j+i_\ell} \geq 25$. (If $j = i_\ell$ then the term r_v is excluded from the sum.) Assuming that the average of the known large coefficients β_{u-i_ℓ} is at least

2.5, then the coefficient of x^{u+v-j} in $r*\beta$ is at least $2.5\times 25+5\times\beta_{u-j}$. The consequence is that wrap failures are quite probable, and the probability is strongly influenced by β_{u-j}.

We test each index β_{u-j} by selecting and encrypting (say) 1000 pairs (m, b) such that the resulting blinding polynomial satisfies the conditions above. Then we sort the indices $u-j$ according to the number of wrap failures. The 72 highest wrap failure rates give us the 72 large β_{u-j}'s (those that equal 2 or 3). And, as already observed, these values reveal the polynomial $g * x^u$.

Our tests have shown that this step is the most accurate and can correct any errors made in the previous steps.

When the sum of the 15 indices of step 2 was 41 and sets of 1000 messages were used, this step almost always revealed β. With sets of 200 messages, we failed to spot on average four 1's in g. In the case where the sum of the 15 indices was 37 and sets of 1000 messages were used, we failed to spot on average one 1 in g.

It seems impractical to construct a sufficiently large database of pairs (m, b) for every 16-tuple of indices $u, i_0, \ldots, i_{14}$. Fortunately this is not necessary. It is sufficient to have a database of pairs (m, b) such that the resulting blinding polynomial has the required peak. If the peak value is ≥ 5, then roughly one out of 30 peak polynomials can be used to test β_u w.r.t. indices $i_0, \ldots, i_{14}$. Hence, a database of 30000 pairs (m, b) that result in a peak polynomial is large enough. For each index to be tested there are roughly 1000 suitable pairs in the database.

9.4 COMPUTING THE PRIVATE KEY

Let us finally show how to compute the private key f from the polynomial g which we now know. Denote $\varphi(x) = \frac{x^N-1}{x-1} = x^{250} + x^{249} + x^{248} + \cdots + 1$. As φ decomposes into five prime factors of degree 50, with a very high probability $\gcd(h, \varphi) = 1$ in $\mathbb{Z}_2[x]$, where $h \equiv f^{-1} * p * g \mod 128$ is the public key. Thus, we can find out the polynomials H_0, a and b such that

$$h * H_0 - a * \varphi = 1 - 2b$$

using the Extended Euclidean algorithm. We also have

$$\begin{aligned} h * H_0 &\equiv 1 - 2b \mod \varphi \\ h * H_0 * (1 + 2b) &\equiv 1 - 4b^2 \mod \varphi \\ h * H_0 * (1 + 2b) * (1 + 4b^2) &\equiv 1 - 16b^4 \mod \varphi \\ h * H_0 * (1 + 2b) * (1 + 4b^2) * (1 + 16b^4) &\equiv 1 - 256b^8 \mod \varphi. \end{aligned}$$

The polynomial $H = H_0 * (1 + 2b) * (1 + 4b^2) * (1 + 16b^4)$ is called the pseudo inverse of h in the ring $\mathbb{Z}_{128}[x]/(x^{251} - 1)$.

We have

$$p * g * H \equiv f * f^{-1} * p * g * H \equiv f * h * H \equiv f \mod \varphi, 128,$$

where f^{-1} is the inverse of f modulo $x^{251} - 1$ in $\mathbb{Z}_{128}[x]$. Therefore

$$f \equiv p * g * H + c\varphi \mod x^{251} - 1, 128$$

for some $c \in \mathbb{Z}_{128}$. From the condition

$$f(1) = p(1)g(1)H(1) + c\varphi(1)$$

we obtain $c \equiv (217 - 216 \times H(1)) \times 251^{-1} \mod 128$.

9.5 GENERATING SUITABLE BLINDING POLYNOMIALS

As mentioned above, we assume that the blinding polynomial r is constructed from three parts: $r = r_1 * r_2 + r_3$, where each $r_i(1) = 8$. The parts r_i are generated (algorithm 3.3.2.2 of [8]) by selecting 8 random indices $j_0, \ldots, j_7$ (repetitions allowed) and setting $r_i = x^{j_0} + \ldots + x^{j_7}$. In theory, the "random" indices are not random, they are determined by the message m, random data b and some specified pseudo-random number generator. However, we obtain a similar distribution of r's via random selection (assuming that the chosen PRNG is secure).

We note that there are about 6.5×10^5 different quadruples modulo rotations when $N = 251$.

In the attack we needed two types of blinding polynomials. The following probabilities are obtained by generating blinding polynomials randomly.

- About one pair (m, b) out of 10000 pairs produces a blinding polynomial $r = \rho(m, b)$ which has four coefficients of values at least four. Such polynomials are needed for about 6.5×10^5 quadruples (i_0, i_1, i_2, i_3). The construction of the whole database requires about 6.5×10^{12} message representative generations. Computing these seems to be the most demanding task of the attack. In fact, the whole database requires approximately 20 gigabytes of memory.

- Roughly one pair (m, b) out of 250 pairs produces a blinding polynomial $r = \rho(m, b)$ with one coefficient of value at least five.

It should be noted that if the parts r_i are required to be binary polynomials, it is much harder to generate r's with large enough coefficients and the construction of the databases will take a longer time. Alternatively, one could modify the attack to use somewhat smaller blinding polynomials, which again would result in a lower success probability. It would still be possible to apply a similar attack.

9.6 AN EXAMPLE

Next we show an example of the attack:

We attack a system with a private key f and a public key $h \equiv f^{-1} * p * g \mod q$. Just for reference, we list the coefficients of the polynomials f and $\beta = (2 + x) * g$ from the smallest to the largest exponent:

f: 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 2, 3, 3, 1, 0, 0, 0, 2, 1, 2, 1, 0, 0, 0, 0, 0, 0, 2, 3, 3,
5, 2, 0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 2, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0,
0, 0, 0, 0, 0, 2, 1, 0, 0, 2, 3, 3, 1, 2, 1, 2, 3, 3, 3, 3, 1, 0, 0, 2, 5, 4, 1, 0, 0, 0, 0, 0, 2, 1, 2, 3,
1, 0, 0, 2, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 2, 1, 0, 2, 1, 0, 0, 2, 1, 2, 3, 1, 0, 0, 0, 0, 2,
1, 0, 0, 0, 2, 1, 4, 2, 0, 2, 3, 1, 0, 4, 4, 1, 0, 2, 1, 0, 0, 0, 2, 1, 2, 1, 0, 0, 0, 2, 1, 2, 1, 0, 0, 0,
0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 2, 1, 0, 0, 0, 0, 0, 4, 4, 3, 1, 0, 0, 2, 1, 2, 3,
3, 1, 0, 2, 1, 0, 0, 0, 0, 2, 1, 0, 2, 1, 2, 3, 1, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0

β: 0, 0, 0, 0, 2, 1, 2, 1, 0, 2, 1, 2, 3, 3, 3, 1, 2, 1, 0, 0, 0, 2, 1, 2, 1, 0, 0, 2, 1, 2, 1, 0, 2, 1, 0, 0,
0, 2, 1, 2, 1, 0, 0, 2, 1, 0, 0, 2, 1, 0, 2, 3, 1, 0, 0, 2, 3, 3, 3, 1, 0, 0, 2, 1, 2, 3, 1, 0, 0, 0, 0, 0,
0, 2, 1, 0, 0, 2, 3, 1, 2, 3, 1, 2, 1, 0, 2, 1, 0, 0, 0, 2, 1, 0, 2, 1, 0, 0, 2, 1, 0, 2, 1, 2, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 2, 1, 2, 1, 0, 0, 2, 1,
2, 3, 3, 1, 0, 0, 0, 0, 2, 1, 2, 3, 3, 3, 1, 0, 0, 0, 0, 0, 2, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 3, 1, 0,
0, 0, 0, 2, 3, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 2, 3, 1, 0, 2, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 2,
1, 2, 1, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 0, 2, 1, 0, 2, 1, 0, 0, 0, 0, 0, 0, 2, 1

quadruple	wrap errors
21,159,204,205	11
37,53,60,224	10
44,67,88,166	10
68,75,112,221	10
9,118,226,236*	10
1,16,118,172	9
9,53,197,240*	9

TABLE 9.1: Step 1. List of best choices for the quadruple corresponding to four 3's. Quadruples marked with a star do not correspond to four 3's in β and choosing one of them would cause some problems for the attack.

Step 1: We picked 1000 random quadruples and sets $\mathcal{M}_{4\times4}^{1000}(i_0, i_1, i_2, i_3)$. After we encrypted and decrypted all one million messages, the quadruples that generated the most wrap errors were listed in table 9.1. We selected the best quadruple $(21, 159, 204, 205)$, which induced 11 wrap errors. We made a guess that for some index u it is true that $\beta_{u-21} = \beta_{u-159} = \beta_{u-204} = \beta_{u-205} = 3$. There is no way for the real attacker to know whether this is true, but we can see from the list of coefficients of β that it is for $u = 217$.

Step 2: For indices $k \in \{0, 1, \ldots, 80\}$, we encrypted messages in the sets $\mathcal{M}_{4\times4}^{1000}(21, 159, 204, k)$, $\mathcal{M}_{4\times4}^{1000}(21, 159, 205, k)$, $\mathcal{M}_{4\times4}^{1000}(21, 204, 205, k)$ and $\mathcal{M}_{4\times4}^{1000}(159, 204, 205, k)$ and observed the number of wrap errors. Results can be seen in table 9.2. We have included the coefficients β_{217-k} in the last column to demonstrate their correlation with the number of wrap errors induced. We added the 11 k's with largest wrap error rates to the four indices picked in step 1. The 15 indices were then

$$17, 18, 21, 22, 33, 40, 41, 60, 61, 62, 71, 72, 159, 204 \text{ and } 205.$$

Note that we only examined about one third of the indices, and therefore $\beta_{217-k} < 3$ for some of the k's which were picked. But this does not matter, as all large coefficients of $\beta * x^{217}$ will be found in step 3. Naturally, more reliable results are obtained if we go through all indices.

Step 3: The next task was to find 1000 suitable pairs $(m, b) \in \mathcal{M}_{1\times5}$ for each index $k \in \{0, 1, \ldots, 250\}$ (as described earlier). Again, we observed the number of wrap errors encountered (table 9.3). We assumed that the

k	wrap errors	
0	4+2+2+7=15	2
1	1+1+7+3=12	1
2	2+2+4+7=15	2
3	0+1+3+0=4	0
4	0+0+1+3=4	0
5	1+1+1+1=4	0
6	1+1+1+3=6	1
7	6+0+2+7=15	2
8	0+1+2+7=10	0
9	1+1+1+2=5	0
10	0+1+2+2=5	0
11	2+0+0+2=4	0
12	2+0+0+2=4	0
13	0+0+2+2=4	0
14	0+1+2+1=4	0
15	0+0+4+4=8	0
16	2+5+5+2=14	1
17	8+5+6+6=25	3
18	2+7+6+6=21	2
19	1+1+3+5=10	0
20	2+4+1+9=16	1
21	n/a	3
22	8+2+13+4=27	2
23	2+2+5+1=10	0
24	2+0+0+3=5	0
25	2+2+1+1=6	0
26	1+0+4+2=7	0

k	wrap errors	
27	0+2+2+0=4	0
28	2+3+4+3=12	0
29	1+0+2+4=7	0
30	1+0+3+3=7	1
31	4+4+7+4=19	2
32	1+2+3+6=12	1
33	6+8+6+7=27	3
34	2+3+9+1=15	2
35	1+1+3+1=6	0
36	0+0+4+2=6	0
37	0+0+1+1=2	0
38	0+1+1+4=6	0
39	0+4+2+6=12	1
40	10+5+10+15=40	3
41	3+3+4+10=20	2
42	1+2+7+9=19	0
43	0+0+4+3=7	0
44	1+0+2+6=9	0
45	1+0+2+2=5	0
46	0+2+0+1=3	0
47	0+1+1+3=5	0
48	0+2+0+4=6	0
49	1+1+2+2=6	0
50	2+2+3+4=11	1
51	7+1+3+5=16	2
52	2+0+10+2=14	1
53	1+1+7+5=14	2

k	wrap errors	
54	0+0+2+0=2	0
55	0+0+0+3=3	0
56	0+0+1+5=6	0
57	1+1+2+0=4	0
58	2+1+2+1=6	0
59	1+4+3+1=9	1
60	4+7+12+7=30	3
61	8+8+11+5=32	3
62	4+9+21+14=48	3
63	1+3+12+1=17	2
64	1+1+3+0=5	1
65	0+2+5+5=12	2
66	2+0+3+4=9	0
67	0+1+0+7=8	0
68	1+2+3+0=6	0
69	0+1+0+0=1	0
70	0+3+6+1=10	1
71	6+6+4+8=24	3
72	6+9+7+10=32	3
73	4+5+9+2=20	2
74	1+2+3+1=7	1
75	0+4+5+6=15	2
76	3+0+9+1=13	0
77	0+1+0+2=3	0
78	5+3+4+5=17	1
79	1+5+3+2=11	2
80	3+3+3+3=12	1

TABLE 9.2: Step 2. Wrap error counts for the first 81 k's. In the last column there is the corresponding coefficient of x^{217-k} in β.

72 largest wrap counts – anything over 80 here – corresponded to 2's and 3's in some rotation of β. A correct assumption here would lead to the discovery of a rotation of the polynomial g. In our case the guess was indeed true: we discovered $g * x^{217}$. However, the attacker must proceed to the next step to check the validity of the obtained g.

Step 4: Let g' be the polynomial found in step 3. Our final task was to compute f' such that $f'^{-1} * p * g' \equiv h \mod q$, where h is the public key. Applying the method described earlier we obtained $f' = f * x^{217}$, and the key was broken.

In this example we did not use actual messages to generate useful blinding polynomials. Instead, for steps 1 and 2 we selected random

k	errors		k	errors		k	errors		k	errors		k	errors		k	errors	
0	105	2	42	48	0	84	91	2	126	93	2	168	30	0	210	42	1
1	46	1	43	40	0	85	48	0	127	13	0	169	38	1	211	93	2
2	85	2	44	24	0	86	41	0	128	11	0	170	118	2	212	49	1
3	19	0	45	13	0	87	49	0	129	18	0	171	26	0	213	95	2
4	16	0	46	18	0	88	23	0	130	60	1	172	25	0	214	34	0
5	27	0	47	17	0	89	22	0	131	97	2	173	68	1	215	29	0
6	46	1	48	36	0	90	21	0	132	22	0	174	116	2	216	23	0
7	101	2	49	20	0	91	17	0	133	54	1	175	30	0	217	23	0
8	43	0	50	44	1	92	15	0	134	105	2	176	22	0	218	42	1
9	9	0	51	125	2	93	26	0	135	55	1	177	47	1	219	81	2
10	16	0	52	62	1	94	13	0	136	200	3	178	97	2	220	29	0
11	14	0	53	105	2	95	44	1	137	122	2	179	46	1	221	19	0
12	19	0	54	22	0	96	101	2	138	52	1	180	94	2	222	14	0
13	29	0	55	14	0	97	30	0	139	197	3	181	22	0	223	13	0
14	15	0	56	25	0	98	21	0	140	137	2	182	23	0	224	22	0
15	19	0	57	19	0	99	11	0	141	26	0	183	17	0	225	29	0
16	72	1	58	24	0	100	43	1	142	16	0	184	53	1	226	45	1
17	192	3	59	53	1	101	108	2	143	50	1	185	116	2	227	92	2
18	140	2	60	210	3	102	27	0	144	110	2	186	37	0	228	22	0
19	39	0	61	222	3	103	16	0	145	18	0	187	50	1	229	48	1
20	45	1	62	228	3	104	19	0	146	14	0	188	98	2	230	105	2
21	208	3	63	116	2	105	23	0	147	19	0	189	52	1	231	40	0
22	131	2	64	38	1	106	21	0	148	14	0	190	121	2	232	24	0
23	25	0	65	100	2	107	20	0	149	27	0	191	25	0	233	23	0
24	13	0	66	23	0	108	19	0	150	70	1	192	19	0	235	13	0
25	23	0	67	16	0	109	21	0	151	197	3	193	72	1	235	50	1
26	21	0	68	18	0	110	16	0	152	117	2	194	121	2	236	91	2
27	18	0	69	22	0	111	8	0	153	51	1	195	67	1	237	28	0
28	20	0	70	47	1	112	24	0	154	108	2	196	109	2	238	12	0
29	20	0	71	201	3	113	44	1	155	40	0	197	32	0	239	19	0
30	44	1	72	202	3	114	94	2	156	19	0	198	19	0	240	18	0
31	93	2	73	116	2	115	60	1	157	43	1	199	16	0	241	11	0
32	59	1	74	46	1	116	101	2	158	203	3	200	33	1	242	14	0
33	216	3	75	107	2	117	25	0	159	199	3	201	88	2	243	18	0
34	105	2	76	30	0	118	41	1	160	214	3	202	55	1	244	18	0
35	23	0	77	18	0	119	117	2	161	205	3	203	208	3	245	10	0
36	23	0	78	48	1	120	29	0	162	113	2	204	221	3	246	19	0
37	18	0	79	101	2	121	16	0	163	17	0	205	227	3	247	49	1
38	19	0	80	45	1	122	42	1	164	22	0	206	136	2	248	95	2
39	52	1	81	91	2	123	104	2	165	63	1	207	40	1	249	27	0
40	217	3	82	26	0	124	26	0	166	223	3	208	115	2	250	56	1
41	129	2	83	53	1	125	41	1	167	116	2	209	28	0			

TABLE 9.3: Step 3. Wrap error counts for all k's. In the last column there is the coefficient of x^{217-k} in β.

binary polynomials with weight 56 and added 4 to four coefficients. For step 3 we selected random binary polynomials with weight 45, added 5 to one coefficient and distributed the rest of the weight, 22, randomly between 15 coefficients.

9.7 A METHOD TO CREATE MORE WRAP ERRORS

The attack which we have described would be more efficient if wrap failures occurred more frequently. And, in fact, we can make this happen. As noted before, the generation function of the message representative i from the message m and randomness b is invertible. Therefore, we can select messages m such that, for example, $i(1) > N/2 + q/2$. Then the average coefficient of α, i.e. A, is computed to be too small (by 110 if $N = 251$) and wrap errors are inevitable. The decryptor is still able to correct the error if (s)he tries to correct A as stated in the standard. The number of corrections needed depends on the largest coefficient of α.

Our attack exploited the connection between the largest coefficient of α and a guess on the polynomial g. Then, a large coefficient was detected by wrap errors, which occurred quite rarely. Using this new approach we get information on the largest coefficient in every decryption process, provided that we can find out how many changes were made to A before the decryption succeeded. This information can be obtained by measuring the time it takes to decrypt a message.

We expect the modification sketched above to be significantly more efficient than the original one.

Although not specified in the standards, NTRU recommends that the polynomial g does not have any two consecutive coefficients equal to 1 [18]. Therefore the coefficients of $p * g$ are all smaller than 3, which makes our basic attack less effective. However, it seems that the modified approach would still work.

The starting point for this modification of the attack was that it is easy to find messages with representatives of large weight. Several other attacks are based on the same property, see [32, 14, 35]. These attacks do not require blinding polynomials with large coefficients. Indeed, even in the binary case, there are serious security implications if wrap or gap errors occur too frequently.

9.8 REMARKS AND CONCLUSION

The starting assumption in our attack is that the attacker can detect wrap failures. An obvious method would be to measure the time taken to decrypt a message. To make the attack impossible, one could make the decryption algorithm constant time, as if wrap failures occurred every time. A more efficient way would be to simulate wrap failures every now and then. Also, the decryption machinery could count the number of wrap failures it faces. If they occur too frequently, the key should be changed.

The counter-measure we would like to recommend is to always require that the blinding polynomial r is binary. Specifically r should not be constructed from two or more smaller parts.

The generation of the required databases $\mathcal{M}_{4\times 4}$ and $\mathcal{M}_{1\times 5}$ can (and must) be done off-line. The workload of generating these sets could be divided over the whole internet, for example. The same library can be used to break any NTRU key.

Our attack is by no means optimized; most probably similar ideas can be further developed to extract g more efficiently. For example, one could use blinding polynomials with different patterns or even larger coefficients. Also we have ignored that the coefficients of β obey a certain pattern. For example, a 3 is always preceded by another 3 or a 2. Finally, it is not necessary to discover g exactly. If we find a close enough approximation, we can apply some lattice reduction methods to extract f (and g).

It should also be mentioned that although the attack is designed against a particular set of parameters, it can be modified to beat efficiently any parameter set, as long as one can generate blinding polynomials with large coefficients. The complexity seems to be linear with respect to N, the degree of the polynomials. However, the number of quadruples increases more rapidly, so the construction of the databases becomes more troublesome.

CHAPTER 10

NTRUSign

As with the NTRUEncrypt public key cryptosystem, there are several versions of the NTRU signature scheme. The current version is called NTRUSign. Before this there was a system called NSS, NTRU Signature Scheme. NTRUSign is defined in the same EESS standard as NTRUEncrypt [9]. In the following we outline the characteristics of NTRUSign.

The lattices of NTRUSign are very similar to those of NTRUEncrypt and the underlying problem is the closest vector problem. All the operations are performed in the ring

$$\mathbf{R} = \mathbb{Z}[x]/(x^N - 1),$$

exactly as in NTRUEncrypt.

We define the *centered norm* of vector $v = (v_1, v_2, \ldots, v_n)$ as

$$||v||_c^2 = \sum_{i=1}^{n} \left(v_i - \frac{1}{n} \sum_{j=1}^{n} v_j \right)^2 = \sum_{i=1}^{n} v_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} v_i \right)^2.$$

REMARK 10.1 If we project the vector into the space orthogonal to $\begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}$ and take the Euclidean norm we get the centered norm of the original vector. It turns out that the centered norm is a better measure than the Euclidean norm. Moreover, the attacker can always work with the centered norms. We have

$$\frac{\mathbb{Z}[x]}{(x^n - 1)} \simeq \mathbb{Z} \times \frac{\mathbb{Z}[x]}{(x^{n-1} + x^{n-2} + \cdots + x + 1)}.$$

REMARK 10.2 The centered norm is quasi-multiplicative, that is,

$$||v * u||_c \approx ||v||_c \cdot ||u||_c.$$

Let us denote $u = \sum_{i=0}^{n-1}(c+u_i)x^i$ and $v = \sum_{i=0}^{n-1}(d+v_i)x^i$, where $\sum_{i=0}^{n-1} u_i = \sum_{i=0}^{n-1} v_i = 0$, and let $u * v = w = \sum_{i=0}^{n-1} w_i x^i$. The average coefficient of w is $\frac{1}{n}w(1) = \frac{1}{n}u(1)v(1) = ncd$ and

$$\begin{aligned} w_i =& \sum_{j=0}^{n-1}(c+u_j)(d+v_{i-j}) \\ =& \sum_{j=0}^{n-1} cd + \sum_{j=0}^{n-1} cv_{i-j} + \sum_{j=0}^{n-1} du_j + \sum_{j=0}^{n-1} u_j v_{i-j} \\ =& ncd + \sum_{j=0}^{n-1} u_j v_{i-j}. \end{aligned}$$

Here we reduced the indices modulo n when needed. Thus

$$\begin{aligned} ||w||_c =& \sum_{i=0}^{n-1}\left(\sum_{j=0}^{n-1} u_j v_{i-j}\right)^2 \\ =& \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(u_j v_{i-j})^2 + 2\sum_{i=0}^{n-1}\sum_{\substack{0\le j,k<n \\ j\neq k}} u_j u_k v_{i-j} v_{i-k} \\ =& ||u||_c \cdot ||v||_c + 2s, \end{aligned}$$

where the terms of s mostly cancel each other out, if u and v are random enough.

REMARK 10.3 We can also show that

$$||v+u||_c^2 \approx ||v||_c^2 + ||u||_c^2.$$

Let $v = (v_1, v_2, \ldots, v_n)$ and $u = (u_1, u_2, \ldots, u_n)$. We have

$$\begin{aligned} ||v+u||_c^2 =& \sum_{i=1}^{n}\left(v_i + u_i - \frac{1}{n}\sum_{j=1}^{n}(v_j+u_j)\right)^2 \\ =& \sum_{i=1}^{n}\left(v_i - \frac{1}{n}\sum_{j=1}^{n} v_j\right)^2 + \sum_{i=1}^{n}\left(u_i - \frac{1}{n}\sum_{j=1}^{n} u_j\right)^2 \\ & + 2\sum_{i=1}^{n}\left(\left(v_i - \frac{1}{n}\sum_{j=1}^{n} v_j\right)\left(u_i - \frac{1}{n}\sum_{j=1}^{n} u_j\right)\right) \\ =& ||v||_c^2 + ||u||_c^2 + 2s \end{aligned}$$

for some s. On the other hand

$$\sum_{i=1}^{n}\left(v_i - \frac{1}{n}\sum_{j=1}^{n} v_j\right) = \sum_{i=1}^{n} v_i - n\frac{1}{n}\sum_{j=1}^{n} v_j = 0.$$

If the vectors are random enough, the quantity s is very close to 0.

We are interested in the centered norms of $v \mod q$. There are several ways to compute this. One way is to put the coefficients into the interval $[-\frac{q}{2}+t, \frac{q}{2}+t]$ for all $t = 0, \ldots, q-1$ and to check when the centered norm is minimal. Another way would be to do as with NTRUEncrypt: Let us assume that q is even. Let $k \equiv n^{-1}v(1) - \frac{q}{2} \mod q$ and $-q < k \leq 0$. Now let $\bar{v} \equiv v \mod q$, where the coefficients of $\bar{v}$ lie in the interval $[k, k+q-1]$. The centered norm of $v \mod q$ is $||\bar{v}||$.

In the following, we are mainly considering the case with parameter values $N = 251$, $q = 128$, $d_f = 73$, $d_g = 71$ and NormBound= 310.

10.1 KEY GENERATION

Let f be a polynomial in $\mathbf{R}$ with d_f randomly selected coefficients set to 1 and the rest set to 0. Let f be such that there exists an inverse f^{-1} such that $f * f^{-1} \equiv 1$ modulo q. Let g be a polynomial in $\mathbf{R}$ with d_g arbitrarily selected coefficients set to 1 and the rest set to 0 and let $h \equiv f^{-1} * g \mod q$.

First we find polynomials F_1, $G_1 \in \mathbf{R}$ such that

$$f * G_1 - g * F_1 = 1.$$

First note that there are polynomials $u, v, k_1, k_2 \in \mathbb{Z}[x]$ such that

$$f * v + k_1 * (x^N - 1) = R_f,$$
$$g * u + k_2 * (x^N - 1) = R_g,$$

in $\mathbb{Z}[x]$ where R_f and R_g are integers. If R_f and R_g are coprimes we can apply the Extended Euclidean algorithm to obtain integers α and β satisfying

$$\alpha R_f + \beta R_g = 1.$$

Putting above three equations together we get

$$(\alpha v) * f + (\beta u) * g \equiv 1 \mod x^N - 1.$$

Now we have found the polynomials $F_1 = -\beta u$ and $G_1 = \alpha v$.

We denote $F' = qF_1$ and $G' = qG_1$. Let

$$B_{fg} = \begin{pmatrix} \overrightarrow{f} & \overrightarrow{g} \\ \overrightarrow{F}' & \overrightarrow{G}' \end{pmatrix} = \left(\begin{array}{cccc|cccc} f_0 & f_1 & \cdots & f_{N-1} & g_0 & g_1 & \cdots & g_{N-1} \\ f_{N-1} & f_0 & \cdots & f_{N-2} & g_{N-1} & g_0 & \cdots & g_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ f_1 & f_2 & \cdots & f_0 & g_1 & g_2 & \cdots & g_0 \\ \hline F'_0 & F'_1 & \cdots & F'_{N-1} & G'_0 & G'_1 & \cdots & G'_{N-1} \\ F'_{N-1} & F_0 & \cdots & F'_{N-2} & G'_{N-1} & G'_0 & \cdots & G'_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ F'_1 & F'_2 & \cdots & F'_0 & G'_1 & G'_2 & \cdots & G'_0 \end{array}\right)$$

and

$$B_h = \begin{pmatrix} I & \overrightarrow{g} \\ 0 & qI \end{pmatrix} = \left(\begin{array}{ccc|cccc} 1 & & & h_0 & h_1 & \cdots & h_{N-1} \\ & 1 & 0 & h_{N-1} & h_0 & \cdots & h_{N-2} \\ & \ddots & & \vdots & \vdots & \ddots & \vdots \\ & 0 & 1 & h_1 & h_2 & \cdots & h_0 \\ \hline & & & q & & & \\ & & & & q & & 0 \\ & 0 & & & & \ddots & \\ & & & & 0 & & q \end{array}\right).$$

THEOREM 17 *Matrices B_{fg} and B_h generate the same lattices, i.e.*

$$L(B_{fg}) = L(B_h).$$

Proof. Using previous notations let

$$U = \begin{pmatrix} \overrightarrow{U}_1 & \overrightarrow{U}_2 \\ \overrightarrow{U}_3 & \overrightarrow{U}_4 \end{pmatrix}$$

where

$$\begin{aligned} U_1 &= G_1 - F_1 * h, \\ U_2 &= \frac{-g + f * h}{q}, \\ U_3 &= -qF_1 \qquad \text{and} \\ U_4 &= f. \end{aligned}$$

We have

$$\begin{aligned}
&U_1 * f + U_2 * F' = G_1 * f - F_1 * h * f - g * F_1 + f * h * F_1 = 1,\\
&U_1 * g + U_2 * G' = G_1 * g - F_1 * h * g - g * G_1 + f * h * G_1 = h,\\
&U_3 * f + U_4 * F' = -qF_1 * f + f * gF_1 = 0 \qquad \text{and}\\
&U_3 * g + U_4 * G' = -qF_1 * g + f * gG_1 = g.
\end{aligned}$$

Thus,

$$UB_{fg} = B_h.$$

Furthermore,

$$\det(U) = (G_1 - F_1 * h) * f + (-g + f * h) * F_1 = 1.$$

These equations together imply that

$$L(B_{fg}) = L(B_h).$$

■

The private signing key will consist of a small basis for the lattice B_{fg}. However, the polynomials F' and G' are not small enough for this purpose. But a slight modification of them will suffice.

We denote by $[f]$ the polynomial where each coefficient of f is rounded to the closest integer. It is easy to see that if

$$k = \left[\frac{f(x^{-1}) * F'(x) + g(x^{-1}) * G'(x)}{f(x^{-1}) * f(x) + g(x^{-1}) * g(x)}\right]$$

and

$$F = F' - k * f, \qquad G = G' - k * f$$

then the matrix

$$\begin{pmatrix} \overrightarrow{f} & \overrightarrow{g} \\ \overrightarrow{F} & \overrightarrow{G} \end{pmatrix}$$

generates the lattice $L(B_{fg})$ and

$$||F||_c \approx ||f||_c\sqrt{\frac{N}{12}} \qquad ||G||_c \approx ||g||_c\sqrt{\frac{N}{12}}.$$

We justify the approximations later.

The private signing key of this signature scheme will be (f, g, F, G). The public verification key is h. These keys generate two different bases for the same matrix.

10.2 SIGNING

The signature of a message m will be a lattice vector close to $\begin{pmatrix} 0 & i \end{pmatrix}$, where i is the message representative of m. Here "close to" means that the distance is at most some predefined limit, namely NormBound.

The process of signing m starts by computing $i \in \mathbb{Z}_q[x]/(x^N - 1)$ using some selected hash function. This method is public as it needs to be done by the verifiers of the signature as well.

Let

$$B = \left[\frac{-F * i}{q}\right] \quad \text{and} \quad b = \left[\frac{f * i}{q}\right].$$

The signature s of the message is

$$s \equiv f * B + F * b \mod q.$$

10.3 VERIFICATION

Suppose that we want to verify that s is the signature of message m, and that we know the public verification key of the (claimed) signer, h. The first task is to compute

$$t \equiv h * s \mod q$$

where $\begin{pmatrix} s & t \end{pmatrix}$ is the lattice point. It only remains to be checked that the centered norm of

$$\begin{pmatrix} s & t \end{pmatrix} - \begin{pmatrix} 0 & i \end{pmatrix}$$

is at most the size of the NormBound constant.

10.4 WHY DOES THE VERIFICATION WORK

The signer wants to find a lattice point near the vector

$$J = \begin{pmatrix} 0 & i \end{pmatrix}.$$

(S)he can see the secret short basis of the lattice directly from (f, g, F, G). If the secret basis is short it is also almost orthogonal and the lattice point

$$\begin{aligned}\begin{pmatrix} s & t \end{pmatrix} &= \begin{pmatrix} B & b \end{pmatrix} \begin{pmatrix} f & g \\ F & G \end{pmatrix} \\ &= \left[\begin{pmatrix} 0 & i \end{pmatrix} \frac{1}{q} \begin{pmatrix} G & -g \\ -F & f \end{pmatrix} \right] \begin{pmatrix} f & g \\ F & G \end{pmatrix} \\ &= \left[\begin{pmatrix} 0 & i \end{pmatrix} \begin{pmatrix} f & g \\ F & G \end{pmatrix}^{-1} \right] \begin{pmatrix} f & g \\ F & G \end{pmatrix}\end{aligned}$$

is near J.

Next we approximate the centered norm of F and G. The polynomials f and g were chosen to satisfy $||f||_c \approx ||g||_c \approx c\sqrt{N}$. We have

$$\begin{aligned}F(x) =& F'(x) - f(x) * \left[\frac{f(x^{-1}) * F'(x) + g(x^{-1}) * G'(x)}{f(x^{-1}) * f(x) + g(x^{-1}) * g(x)} \right] \\ =& F'(x) - f(x) * \frac{f(x^{-1}) * F'(x) + g(x^{-1}) * G'(x)}{f(x^{-1}) * f(x) + g(x^{-1}) * g(x)} + f(x) * A(x) \\ =& F'(x) - \frac{f(x^{-1}) * f(x) * F'(x) + g(x^{-1}) * f(x) * G'(x)}{f(x^{-1}) * f(x) + g(x^{-1}) * g(x)} \\ &+ f(x) * A(x),\end{aligned}$$

where the coefficients of A are in the interval $[-\frac{1}{2}, \frac{1}{2}]$. We substitute $f * G' = g * F' + q$ into the equation above and get

$$F(x) = \frac{q g(x^{-1})}{f(x^{-1}) * f(x) + g(x^{-1}) * g(x)} + f(x) * A(x).$$

Because $||f(x)||_c = ||f(x^{-1})||_c \approx ||g(x)||_c = ||g(x^{-1})||_c$, Remark 10.3 gives us

$$\begin{aligned}||f(x^{-1}) * f(x) + g(x^{-1}) * g(x)||_c \approx& \sqrt{||f(x^{-1}) * f(x)||_c^2 + ||g(x^{-1}) * g(x)||_c^2} \\ \approx& \sqrt{2}||g(x^{-1})||_c^2.\end{aligned}$$

In addition, if the coefficients of A are uniformly distributed in the interval $[-\frac{1}{2}, \frac{1}{2}]$, the centered norm of A is approximately $\sqrt{\frac{N}{12}}$. This is due to the fact that the square of the average coefficient is

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} x^2 dx = 2 \Big/_0^{\frac{1}{2}} \frac{x^3}{3} = \frac{1}{12}.$$

Because of its quasi-multiplicativity, the centered norm of $F(x)$ has an upper limit which is approximately

$$\frac{q||g(x^{-1})||_c}{\sqrt{2}||g(x^{-1})||_c^2} + ||f(x)||_c||A(x)||_c \approx \frac{q}{c\sqrt{2N}} + \frac{cN}{\sqrt{12}} \approx \frac{cN}{\sqrt{12}}.$$

The same approximation holds for G.

We have shown that $||f||_c \approx ||g||_c \approx c\sqrt{N}$ and $||F||_c \approx ||G||_c \approx \frac{cN}{\sqrt{12}}$. We have

$$\begin{pmatrix} s & t \end{pmatrix} - \begin{pmatrix} 0 & i \end{pmatrix} = \begin{pmatrix} A & a \end{pmatrix} \begin{pmatrix} f & g \\ F & G \end{pmatrix},$$

where the coefficients of a and A are in the interval $[-\frac{1}{2}, \frac{1}{2}]$. We assume that these coefficients are uniformly distributed on this interval. The centered norms of a and A are approximately $\sqrt{\frac{N}{12}}$. Because of the quasi-multiplicativity we have

$$\begin{aligned} || \begin{pmatrix} s & t \end{pmatrix} - \begin{pmatrix} 0 & i \end{pmatrix} ||_c^2 &= ||A * f + a * F||_c^2 + ||A * g + a * G||_c^2 \\ &\approx ||A * f||_c^2 + ||a * F||_c^2 + ||A * g||_c^2 + ||a * G||_c^2 \\ &\approx 2\left(\frac{N}{12}c^2 N + \frac{N}{12}\frac{c^2N^2}{12}\right) \\ &\approx \frac{c^2N^2}{6} + \frac{c^2N^3}{72}. \end{aligned}$$

With our parameter set the square root of this quantity equals 216 and $c = 0.45$. Thus we have a hefty safety margin, as the parameter NormBound equals 310.

10.5 AN EXAMPLE

Figure 10.1 describes a lattice with a short basis $(4, 1), (-1, 2)$. Using this basis we can calculate a lattice point (s, t) close to $(0, 5)$:

$$\begin{pmatrix} 4 & 1 \\ -1 & 2 \end{pmatrix}^{-1} = \begin{pmatrix} \frac{2}{9} & -\frac{1}{9} \\ \frac{1}{9} & \frac{4}{9} \end{pmatrix},$$

$$\begin{pmatrix} 0 & 5 \end{pmatrix} \begin{pmatrix} \frac{2}{9} & -\frac{1}{9} \\ \frac{1}{9} & \frac{4}{9} \end{pmatrix} = \begin{pmatrix} \frac{5}{9} & \frac{20}{9} \end{pmatrix} \approx \begin{pmatrix} 1 & 2 \end{pmatrix} \qquad \text{and}$$

$$\begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 4 & 1 \\ -1 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 5 \end{pmatrix}.$$

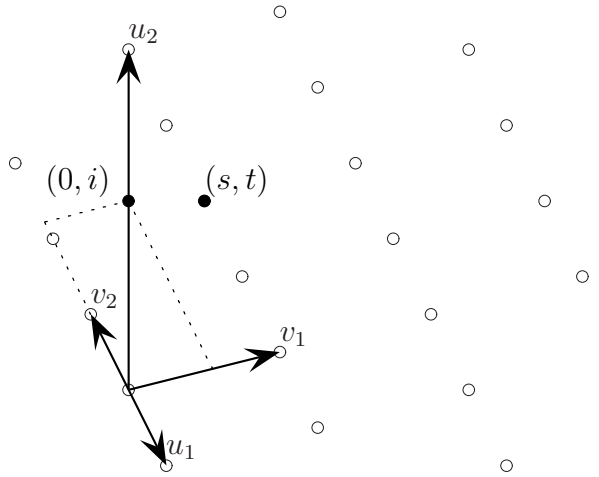


FIGURE 10.1: A lattice with two bases ($v_1 = (4, 1), v_2 = (-1, 2)$ and $u_1 = (1, -2), u_2 = (0, 9)$), the message representative $i = 5$ and the signature $s = 2$.

With the same method and a longer basis $(1, -2), (0, 9)$ we can only find a point which is further away from $(0, 5)$:

$$\begin{pmatrix} 0 & 5 \end{pmatrix} \begin{pmatrix} 1 & \frac{2}{9} \\ 0 & \frac{1}{9} \end{pmatrix} = \begin{pmatrix} 0 & \frac{5}{9} \end{pmatrix} \approx \begin{pmatrix} 0 & 1 \end{pmatrix},$$
$$\begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -2 \\ 0 & 9 \end{pmatrix} = \begin{pmatrix} 0 & 9 \end{pmatrix}.$$

However, anybody with this kind of basis can verify that (s, t) belongs to the lattice:

$$s * h = 2 * -2 \equiv 5 = t \mod 9.$$

10.6 AN ATTACK

The idea of the attack discussed in this section is that two signatures that are very close to each other give us more information about the private key than the public key alone.

Let us assume that we have acquired two signatures s and s' generated with the same private key. Let L be the lattice generated by this key. The vectors (s, t) and (s', t') belong to the lattice L when $t \equiv s*h \mod q$ and $t' \equiv s' * h \mod q$. Therefore their difference also belongs to the lattice. With some luck we have $s - s' = \pm x^k * f$, and the private key is revealed.

We performed some experiments to approximate the probability of this happening. If the distance between the message representatives i and i' was at least 5, the probability was far too small to have any practical relevance. But in the extreme case, when i and i' differed by only one, the private key was found with probability 1%.

If we were to find two messages with adjacent message representatives we could break any private key with probability 1%. We would only need to persuade our victim to sign those two messages.

It is highly unlikely to be able to find messages whose representatives are close enough. However, we can improve the described method.

Our first improvement is quite trivial. It is enough to find representatives with some rotations close to each other.

A better improvement is achieved by noting that the polynomial f has smaller coefficients than the polynomial F. When signing two message representatives i and i' we calculate

$$B = \left[\frac{-F * i}{q}\right], \qquad b = \left[\frac{f * i}{q}\right]$$

and

$$B' = \left[\frac{-F * i'}{q}\right], \qquad b' = \left[\frac{f * i'}{q}\right].$$

If the representatives are relatively close to each other, the polynomials B and B' differ from one another but the polynomials b and b' are the same. Then the difference of the signatures is $s - s' \equiv f * (B - B') \mod q$. Both f and $B - B'$ are polynomials with small coefficients. Thus with high probability we have $s - s' = f * (B - B')$. If an attacker succeeds in acquiring the signatures s and s' of the selected i and i', he can easily calculate the private key f.

We have estimated the probability that b equals b'. If the distance of i and i' (or a rotation of i') is 10, then the probability is about 1%.

Still, it is impossible to say if such representatives exist. However, if such a pair were to exist it could be used against all the private keys.

CHAPTER 11

Concluding remarks

In this thesis we have considered the NTRU public key systems. Main emphasis has been on the security of the encrypytion system.

The most severe attacks against NTRU encryption systems have been based on the occurrences of wrap errors. One of these attacks is described in Chapter 9. Consequently, the system has been modified. In the current version of the system the parameters have been selected in such a way that the probability of wrap errors is negligible. Therefore attacks of this kind are no longer a threat.

Any public key cryptosystem can be attacked in two ways: either one tries to learn the private key given the public key, or one tries to learn the plaintext given the cryptotext. In the NTRU case these problems can be reduced to the problem of finding small vectors in the NTRU-lattice. No efficient algorithm for this problem is known. Better methods are continuously being developed, see [32, 39, 40], but the advances have not been significant. A small increase in the size of the parameters seems to be enough to cancel out the effects of the little faster reduction methods.

Bibliography

[1] M. Ajtai, Generating Hard instances of Lattice Problems. Electronic Colloquium on Computational Complexity, TR96-007, 1996, http://www.eccc.unitrier.de/eccc/

[2] I. Blake, G. Seroussi and N. Smart, Elliptic Curves in Cryptography. Cambridge University Press, Cambridge, 1999.

[3] D. Bleichenbacher, Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS#1. Advances in Cryptology - Crypto '98, Springer-Verlag, 1992.

[4] J. W. S. Cassels, An Introduction to the Geometry of Numbers. Springer-Verlag, Berlin, 1971.

[5] S. A. Cook, The complexity of theorem proving procedures. Proc. 3rd Ann. ACM Symp. on Theory of Computing, Association for Computing machinery, New York, 151-158, 1971.

[6] W. Diffie and M. Hellman, New directions in cryptography. IEEE Transactions on Information Theory IT-22, 644-654, 1976.

[7] C. Dwork, Lattices and Their Application to Cryptography. Lecture Notes, Stanford University, 1998.

[8] Efficient Embedded Security Standard (EESS) #1: Draft 4. Consortium for Efficient Embedded Security, March, 2002.

[9] Efficient Embedded Security Standard (EESS) version 2.0. Consortium for Efficient Embedded Security, June, 2003.

[10] T. El Gamal, A public key cryptosystem and signature scheme based on discrete logarithms. IEEE Transactions on Information Theory IT-31, 496-473,1976.

[11] C. Gentry, Key Recovery and Message Attacks on NTRU-Composite. Advances in Cryptology - Eurocrypt 2001, LNCScience 2045, Springer-Verlag, 2001.

[12] O. Goldreich. Foundations of Cryptography: Volume 1, Basic Tools. Cambridge University Press, 2001.

[13] G. H. Golub and C. F. van Loan, Matrix computations. The Johns Hopkins University Press, Baltimore, 1989.

[14] D. Han, J. Han, D. Kwon and J. Hong, A New Chosen Ciphertext Attack against NTRU. Unpublished manuscript.

[15] J. Hoffstein, J. Pipher and J. H. Silverman, NTRU: A Ring-Based Public Key Cryptosystem. Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, J.P. Buhler (ed.), LNCS 1423, Springer-Verlag, Berlin, 267-288, 1998.

[16] J. Hoffstein, J. H. Silverman and W. Whyte, Estimated Breaking Times for NTRU Lattices. Technical Report #12, available at www.ntru.com.

[17] N. Howgrave-Graham, P. Q. Nguyen, D. Pointcheval, J. Proos, J. H. Silverman, A. Singer and W. Whyte, The Impact of Decryption Failures on the Security of NTRU Encryption. Advances in Cryptology - CRYPTO 2003, LNCS 2729, Springer-Verlag, Berlin, 226-246, 2003.

[18] N. Howgrave-Graham, P. Q. Nguyen, D. Pointcheval, A. Singer and W. Whyte, Padding Schemes and Decryption Failures in NTRUEncrypt. Unpublished manuscript.

[19] N. Howgrave-Graham, J. H. Silverman and W. Whyte, A Meet-In-The-Middle Attack on an NTRU Private Key. Technical Report #4, Version 2, available at www.ntru.com.

[20] N. Howgrave-Graham, J. H. Silverman and W. Whyte, Choosing Parameter Sets for NTRUEncrypt with NAEP and SVES. Unpublished manuscript, available at www.ntru.com.

[21] IEEE P1363.1. Standard Specification for Public-Key Cryptographic Techniques Based on Hard Problems over Lattices.

[22] É. Jaulmes and A. Joux, A Chosen-Ciphertext Attack against NTRU. Advances in Cryptology - Crypto 2000, LNCS 1880, Springer-Verlag, Berlin, 20-35, 2000.

[23] R. Kannan, Minkowski's convex body theorem and integer programming. Mathematics of Operations Research, Vol 12 No. 3, 415-440, 1987.

[24] N. N. Lebedev, Special functions and their applications. Dover Publications, 1972.

[25] A. K. Lenstra, H. W. Lenstra, and L. Lovasz, Factoring Polynomials with Rational Coefficients. Math. Ann. 261, 515-534, 1982.

[26] R. Lidl and G. Pilz, Applied Abstract Algebra. Springer-Verlag, Berlin, 1984.

[27] A. May, Cryptanalysis of NTRU. Unpublished preprint, 1999. Available at http://www.informatik.uni-frankfurt.de/~alex/ntru.ps

[28] A. May and J. H. Silverman, Dimension reduction methods for convolution modular lattices. Conference on Lattices and Cryptography (CaLC 2001), LNCS 2146, Springer-Verlag, 111-127.

[29] A. Menezes, P. van Oorschot and S. Vanstone, Handbook of applied cryptography. CRC Press, 1997.

[30] R. Merkle and M. Hellman, Hiding information and signatures in trapdoor knapsacks. IEEE Transactions in Information Theory IT-24, 525-530, 1978.

[31] T. Meskanen and A. Renvall, A Wrap Error Attack against NTRUEncrypt. To appear in Discrete applied mathematics.

[32] P. Q. Nguyen and D. Pointcheval, Analysis and Improvements of NTRU Encryption Paddings. Advances in Cryptology - CRYPTO 2002, Lecture Notes in Computer Science 2442, Springer-Verlag, Berlin, 210-225, 2002.

[33] P. Q. Nguyen and D. Stehlé, Floating-Point LLL Revisited. Proceedings of Eurocrypt '05. Available from http://www.di.ens.fr/~pnguyen/pub.html

[34] www.ntru.com.

[35] J. Proos, Imperfect Decryption and an Attack on the NTRU Encryption Scheme. Technical report, available at http://www.cacr.math.uwaterloo.ca/tech_reports.html.

[36] R. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21, 120-126, 1978.

[37] C. P. Schnorr, A hierarchy of polynomial time lattice basis reduction algorithms. Theoretical Computer Science, 53, 201-224, 1987.

[38] C. P. Schnorr, Block Korkin-Zolotarev Bases and Successive Minima. Technical report, 1996, www.mi.informatik.uni-frankfurt.de/research/papers/schnorr.block_korkine_zolotarev.1994.ps.gz

[39] C. P. Schnorr, Lattice Reduction by Random Sampling and Birthday Methods, STACS 2003, Computer Science 2607, 145-156, Springer-Verlag, Berlin, 2003.

[40] C. P. Schnorr, Fast LLL-Type Lattice Reduction. Available at http://www.mi.informatik.uni-frankfurt.de/research/papers.html

[41] C. P. Schnorr and H. H. Hörner, Attacking the Chor–Rivest Cryptosystem by Improved Lattice Reduction. Eurocrypt'95, LNCS 921 (1995), Springer Berlin, 1-12.

[42] C. P. Schnorr and M. Euchner, Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. Math. Programming 66, 181-199, 1994.

[43] J. H. Silverman and W. Whyte, Estimating Decryption Failure Probabilities for NTRUEncrypt. Technical Report #18, available at www.ntru.com.

Appendix A

The PARI/GP code we used to generate NTRU-like lattices:

```
{
    n=115;                          /* Set the parameters */
    d=32;
    q=109;

    setrand(71104);
    lb=matrix(n*2,n*2);          /* Set matrix dimensions */
    f=Mod(1+2*bigtau(d,0,n),x^n-1); /* Generate the keys */
    g=Mod(bigtau(d,0,n),x^n-1);
    h=2*g*inverse(f,q,n);
    h=redu(h,q,n);

    i=1;                           /* Generate the matrix */
    lb[1,1]=1;
    while(i<=n,
        lb[n+i,1]=polcoeff(h,i-1);
        i++;
    );
    j=2;
    while(j<=n,
        i=2;
        while(i<=2*n,
            lb[i,j]=lb[i-1,j-1];
            i++;
        );
        lb[1+n,j]=lb[2*n,j-1];
        j++;
    );
    i=n+1;
```

```
    while(i<=2*n,
        lb[i,i]=q;
        i++;
    );
    firstline=lb[,1];              /* Tricks one and two */
    lb=lb*4;                       /* and the embedding  */
    i=1;
    while(i<=n,
        lb[i,1]=-1;
        i++;
    );
    while(i<=2*n,
        lb[i,1]=2*firstline[i]-1;
        if(lb[i,1]%4==1,lb[i,1]-=q*2,);
        if(lb[i,1]%4==-3,lb[i,1]+=q*2,);
        i++;
    );
    i=1;
    write1("ofb", "[");            /* Output the lattice */
    while(i<=2*n,write("ofb", lb[,i]~);i++);
    write("ofb", "]");
    write("fgh",f);
    write("fgh",g);
    write("fgh",h);
}
```

The C++ code we used to find the shortest vectors of NTRU-like lattices:

```
#include "NTL/LLL.h"
#include "stdio.h"

static long EndCondition(const vec_ZZ& z)
{                                // Check whether we have
     long i,n;                   // found the result

     n=z.length();
     for(i=1;i<=n;i++)
          if (z(i)>3 || z(i)<-3) return 0;
     cerr << z;
     cout << z;
     return 1;
}

main()
{
     int             i,j,n2,beta;
     mat_ZZ          L,LL;
     double          t;

     cin >> L;                   // The lattice
     cin >> df;                  // The weight of the result
     cin >> beta;                // Starting block-size

     n2=L.NumRows();             // The dimension of L
     for(i=1;i<=n2/2;i++)        // Trick 3B
          for(j=1;j<=n2;j++)
               L(i,j)+=1;
     for(i=1;i<=n2;i++)
          L(1,i)-=df+1;
     swap(L(1),L(n2));
     for(i=2;i<n2;i++) {         // Randomize order of rows
          j=RandomBnd(n2-i)+i+1;
          swap(L(i),L(j));
     }
     for(i=beta;i<70;i++) {
          cerr << i << "\n";
```

```
            cout << i << "\n";
            LL=L;
            t=GetTime();          // Start timer
            G_BKZ_FP(LL,0.99,i,0,EndCondition,0); // Call BKZ
            cerr << (GetTime()-t) << "\n";
            cout << (GetTime()-t) << "\n";
            cerr << LL(1) << "\n";
            cout << LL(1) << "\n";
        }
}
```

Turku Centre for Computer Science
TUCS Dissertations

25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming : C++ is More Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Marked Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Franck Tétard**, Managers, Fragmentation of Working Time, and Information Systems
41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, Z_4-Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity - A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen**, On the NTRU Cryptosystem

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku

- Department of Information Technology
- Department of Mathematics

Åbo Akademi University

- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration

- Institute of Information Systems Sciences

ISBN 952-12-1570-4
ISSN 1239-1883