NTNU
Innovation and Creativity

Security Analysis of the NTRUEncrypt Public Key Encryption Scheme

Halvor Sakshaug

Master of Science in Physics and Mathematics
Submission date: June 2007
Supervisor: Alexei Roudakov, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Problem Description

The NTRUEncrypt public key cryptosystem is a relatively new system. The work should give a survey of the existing analysis of the system, especially lattice-based attacks.

Assignment given: 17. January 2007
Supervisor: Alexei Roudakov, MATH

Abstract

The public key cryptosystem NTRUEncrypt is analyzed with a main focus on lattice based attacks. We give a brief overview of NTRUEncrypt and the padding scheme NAEP. We propose NTRU-KEM, a key encapsulation method using NTRU, and prove it secure. We briefly cover some non-lattice based attacks but most attention is given to lattice attacks on NTRUEncrypt. Different lattice reduction techniques, alterations to the NTRUEncrypt lattice and breaking times for optimized lattices are studied.

Preface

I would like to thank Kristian Gjøsteen for excellent supervision of this project, Per Kristian Hove for quick and helpful assistance in C++ implementations and the fellow students Børge Nordli, Asgeir Steine, Jørgen Avdal and Øystein Thuen for proofreading, discussions and answers.

Contents

1 Introduction

The NTRUEncrypt public key cryptosystem was first presented at Crypto '96 by NTRU Cryptosystems Inc and is now included in the IEEE P1363 standard. NTRUEncrypt offers high speed key creation, encryption and decryption and can easily be implemented on constrained devices. It has gained interest and customers in the electronics industry.

NTRUEncrypt is a probabilistic, lattice based cryptosystem. The hard problem of NTRUEncrypt is based on finding very short vectors in a lattice of high dimension. NTRUEncrypt has the unusual feature of decryption failures, namely that validly created ciphertexts may fail to decrypt correctly. New parameter sets for NTRUEncrypt have been presented on several occasions to increase security and efficiency. With the latest recommended parameter sets from NTRU Cryptosystems decryption failuers are extremely rare or do not occur at all.

We present the NTRUEncrypt cryptosystem, its evolution, paddings and various attacks. We propose an NTRU-KEM and analyze its security in section 4. Our main focus has been on lattice attacks on NTRUEncrypt studying general lattice reduction methods in section 2 and methods specific for the NTRUEncrypt lattice and adjustments to the lattice itself in section 6. We have also implemented lattice reduction of NTRUEncrypt lattices testing various adjustments to the NTRUEncrypt lattice and run a series of experiments to project breaking times for NTRUEncryt. Results are given in section 7.

2 Lattices and lattice reduction

2.1 Lattices

A *lattice* is a discrete, additive subgroup of $\mathbf{R}^n$. We may also restrict to additive subgroups of $\mathbf{Z}^n$ called *integer lattices*. An equivalent definition of lattice is all integral linear combinations of a set of linearly independent vectors,

$$L = \{\sum_{i=1}^{m} a_i \mathbf{b}_i \mid a_i \in \mathbf{Z}, \mathbf{b}_i \in \mathbf{R}^n\}$$

where $\{\mathbf{b}_i\}$ is a set of linearly independent vectors in the vector space $\mathbf{R}^n$. A set of linearly independent vectors generating the lattice is called a *basis*, and there are an infinite number of different bases for a lattice. All bases of a lattice share the same number of elements, called the *rank* of the lattice. If $m = n$, the lattice is said to have full rank.

We will often represent the lattice as a matrix with the basis vectors as the columns of the matrix. Operations on the lattice are exchanging of vectors and adding a multiple of one vector to another. This can be done

by multiplying by an $n \times n$ identity matrix with two columns exchanged or with a non-zero off-diagonal element respectively. Such matrices for elementary column operations clearly have determinant ± 1. These matrices and products of them are called *unimodular matrices*.

The *determinant of a lattice* is denoted $\det(L)$ and also called the *volume* of the lattice. It is given as $\det(L) = \sqrt{|L^T L|}$, with the lattice written as a matrix and $|\cdot|$ the regular matrix determinant. For the special case of full rank lattices the lattice determinant equals the matrix determinant. It remains unchanged in lattice operations as multiplying by a unimodular matrix has determinant ± 1. The determinant can also be computed as the product of the lengths of an orthogonalized system of vectors, hence the name volume for the m-dimensional parallelepiped spanned by the basis.

For norms of vectors, $\|\mathbf{b}_i\|$, the Euclidian length is used. Finding the shortest vector in the lattice, or the vector closest to a given point not in the lattice are central problems in the study of lattices. These problems are called shortest vector problem (SVP) and closest vector point (CVP) respectively, and defined as follows,

$$\text{Shortest vector problem: } \mathrm{SVP}(L) = \{\mathbf{x} \mid \mathbf{x} \neq \mathbf{0}, \forall \mathbf{y} \in L : |\mathbf{x}| \leq |\mathbf{y}|\}$$

$$\text{Closest vector point: } \mathrm{CVP}(L, \mathbf{x}_0) = \{\mathbf{x} \mid \forall \mathbf{y} \in L : |\mathbf{x} - \mathbf{x}_0| \leq |\mathbf{y} - \mathbf{x}_0|\}.$$

The shortest vector of a lattice is called the *first minimum* and denoted by $\lambda_1(L)$ or simply $\lambda(L)$. We can define successive minima of a lattice by $\lambda_i = \lambda_i(L)$ where λ_i is the radius of the smallest ball centered at the origin containing i linearly independent vectors.

For the further treatment of lattices we will introduce some more notation. Let $\mathbf{b}_1^*, ..., \mathbf{b}_m^*$ denote an orthogonalized system of lattice vectors orthogonalized on increasing indices as in the Gram-Schimdt procedure. The projection of a lattice vector onto another is given by $\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}$ for $1 \leq j \leq i \leq m$. We will also use $\lceil x \rfloor = \lfloor x + 0.5 \rfloor$ as rounding to the nearest integer.

2.2 Reduced lattice basis

We are interested in lattice bases with certain good properties. Primarily we seek a basis that has one or more short vectors. Finding the shortest vector in a lattice of arbitrary rank is infeasible as solving SVP under randomized reductions is NP-hard [1, 13], and it has also been shown that CVP is equally hard [28]. To obtain a feasible bound on the computational time, we accept reasonably short vectors by weaker bounds on the length.

Lattice basis reduction algorithms finds a lattice basis with short vectors that are nearly orthogonal and sorted on length. We call such lattice bases *reduced*.

There are different definitions of a reduced lattice basis and ways of obtaining such. Primarily there are different orderings of the vectors on which the quality of reduction and efficiency of the reduction method depends. A strict ordering of vectors by their length leads to a runnning time exponential in the dimension of the lattice. More lenient restrictions lead to running times exponential in a factor lower than the dimension of the lattice and for some orderings a polynomial running time. More efficient algorithms give lower quality of reduction.

We will examine the upper bounds of vector lengths and running time of some definitions of reduced lattice bases and corresponding methods for obtaining them. We do not give all details here but refer to [21].

A general criterion for all definitions of reduced lattice basis is that the basis is *size reduced* meaning that

$$|\mu_{i,j}| \leq 0.5$$

for all $1 \leq j < i \leq m$. Size reduction of a lattice decreases vector lengths as well as ensuring that the vectors are close to orthogonal. It is performed by setting

$$\mathbf{b}_i \leftarrow \mathbf{b}_i - \lceil \mu_{i,j} \rfloor \mathbf{b}_j.$$

2.3 LLL reduced lattice basis and the LLL algorithm

We first present the LLL lattice basis reduction algorithm by Lenstra, Lenstra and Lovász [14], which is an efficient algorithm for finding a reduced basis, and the method also gives name to the following definition.

Definition 1. *A lattice basis is* Lovász reduced*, if for a* δ*,* $\frac{1}{4} < \delta < 1$*, and all* i*,* $2 \leq i \leq m$ *the vectors satisfy the ordering*

$$\delta \|\mathbf{b}_{i-1}^*\|^2 \leq \|\mathbf{b}_i^* - \mu_{i,i-1}\mathbf{b}_{i-1}^*\|^2. \tag{1}$$

A lattice basis is LLL reduced *with a given* δ *if it is size reduced and Lovász reduced.*

Proposition 1. *The length of vectors in an LLL reduced lattice bases satisfies*

$$\|\mathbf{b}_j^*\|^2 \leq 2^{i-j}\|\mathbf{b}_i^*\|^2, \text{ and}$$
$$\|\mathbf{b}_1\| \leq 2^{\frac{n-1}{4}} \det(L)^{\frac{1}{n}}.$$

Proposition 2. *An LLL reduced lattice basis can be obtained by at most* $\mathcal{O}(m^3 n \log(B))$ *operations on integers that are* $\mathcal{O}(m \log(B))$ *bits long, where* $B = \max_i \|\mathbf{b}_i\|^2$ *in the initial lattice.*

The two main operations of the LLL algorithm is the size reduction and swap of consequtive vectors when (1) is not satisfied. Starting with $i = 2$

it builds a subset of vectors that are size reduced and Lovász reduced by increasing the subset when (1) is satisfied and shrinks the subset when it is not. At $i = m$ the algorithm terminates as it will be size reduced and all pairs are ordered.

The LLL algorithm is described in Algorithm 1. This variant focuses on simplicity and ignores many obvious improvements. See [14] for a full presentation of the LLL algorithm.

Due to its effectiveness, the LLL algorithm with different improvements has become a standard method for lattice reduction. LLL also serves as the basic reduction procedure in other lattice reduction methods.

Algorithm 1 LLL algorithm

Input: Lattice basis $\{\mathbf{b}_1, ..., \mathbf{b}_n\} \subset \mathbf{Z}^n$
Start
[Initial step] Compute orthogonalized vectors, $\mathbf{b}_1^*, ..., \mathbf{b}_n^*$
[Reduction step]
for i from 2 to n **do**
  for j from $i-1$ to 1 **do**
    $\mathbf{b}_i \leftarrow \mathbf{b}_i - \lceil \mu_{i,j} \rfloor \mathbf{b}_j$
[Swap step]
for i from 2 to n **do**
  if $\delta \|\mathbf{b}_{k-1}^*\|^2 > \|\mu_{k,k-1}\mathbf{b}_{k-1}^* + \mathbf{b}_k^*\|^2$ **then**
    $\mathbf{b}_{k-1} \leftrightarrow \mathbf{b}_k$
    goto Start
Output $\mathbf{b}_1, ..., \mathbf{b}_n$

2.4 The LLL floating point algorithm

The original LLL algorithm of Lenstra, Lenstra and Lovász operate on integer lattices and also uses integers for representing the entries of rational numbers originating in Gram-Schmidt orthogonalization. To speed up the algorithm it is desireable to use floating point arithmetics for all non-integral numbers, while keeping the lattice entries integral. This does however affect stability, and due to round off errors the algorithm may fail to terminate, or produce a basis which is not LLL reduced.

Several floating point versions of LLL with reduced running time have been proposed. Different floating point precision, different orthogonalization methods and special considerations when floats become very large or very small are the main ideas of these proposed algorithms. Some reduction of worst case running time is achieved, especially at low precision. To retain stability in high dimension, a certain amount of precision is needed, as well as more costly orthogonalization methods.

Choosing the best LLL procedure to use for a given lattice is hard. LLL

reduction generally behaves much better than the given worst case bounds both in running time and lengths of vectors. Using methods extending LLL such as BKZ or "deep insertions", where not only consequtive vectors are swapped, allow trade-offs between quality of reduction and running time. Practical performance on individual matrices may also differ greatly.

2.5 Korkin-Zolotarev reduced lattice basis

The most natural definition of reduced lattice basis was introduced by Korkin and Zolotarev [12] and has a strict ordering of the vectors with respect to their lengths. Let $\pi_i(\mathbf{b}_l)$ be the projection of $\mathbf{b}_l$ orthogonal to $\mathbf{b}_1, ..., \mathbf{b}_{i-1}$. Let L_i be the lattice of rank $m-i+1$ with basis $\pi_i(\mathbf{b}_i), \pi_i(\mathbf{b}_{i+1}), ..., \pi_i(\mathbf{b}_m)$. Note that in L_i we have $\pi_i(\mathbf{b}_i) = \mathbf{b}_i^*$ and $\pi_i(\mathbf{b}_l) = \mathbf{b}_l^* + \sum_{k=i}^{l-1} \mu_{l,k}\mathbf{b}_k^*$.

Definition 2. *A* Korkin-Zolotarev reduced *basis is size reduced bases that satisfies*

$$\|\mathbf{b}_i^*\| = \lambda_1(L_i) \text{ for all } i.$$

As the basis is size reduced, we have $\mu_{i,j}^2 \leq \frac{1}{4}$ for all $j < i$ and for the upper bound of vector lengths we have

$$\|\mathbf{b}_i\|^2 \leq \|\mathbf{b}_i^*\|^2 + \frac{1}{4}\sum_{j=1}^{i-1}\|\mathbf{b}_j^*\|^2 \leq \lambda_i(L)^2 + \frac{1}{4}\sum_{j=1}^{i-1}\lambda_j(L)^2 \leq \frac{i+3}{4}\lambda_i(L)^2.$$

For $j \leq i$ we have that

$$\|\mathbf{b}_j^*\|^2 = \lambda_1(L_j)^2 \leq \|\mathbf{b}_i\|^2$$

and

$$\|\mathbf{b}_j\|^2 \leq \|\mathbf{b}_j^*\|^2 + \frac{1}{4}\sum_{k=1}^{j-1}\|\mathbf{b}_k^*\|^2 \leq \frac{j+3}{4}\|\mathbf{b}_i\|^2,$$

so consequently

$$\lambda_i(L)^2 \leq \frac{i+3}{4}\|\mathbf{b}_i\|^2,$$

which also gives us a lower bound for $\|\mathbf{b}_i\|$ resulting in the following proposition

Proposition 3. *The length of the vectors in a KZ reduced basis satisfies*

$$\frac{4}{i+3} \leq \frac{\|\mathbf{b}_i\|^2}{\lambda_i^2} \leq \frac{i+3}{4} \quad \text{for } i = 1, ..., m. \tag{2}$$

Given a Korkin-Zolotarev reduced lattice basis we would like to bound the lengths of the orthogonal projections, namely to find the bound of the constant α_k in $\|\mathbf{b}_i^*\|^2 \leq \alpha_k\|\mathbf{b}_{i+j}^*\|^2$, for $j \leq k-1$. Given that two vectors $\mathbf{b}_i, \mathbf{b}_{i+j}$ have the same length in L_i, how much shorter can the orthogonal

projection of one to the other be? With size reduced vectors $|\mu_{i+j,i}| \leq \frac{1}{2}$ we can find the minimal length of $\|\mathbf{b}_{i+1}^*\|^2$. For this set $j = 1, k = 2$ to imagine two vectors of equal length in L_i, $\|\pi_i(\mathbf{b}_i)\| = \|\pi_i(\mathbf{b}_{i+1})\|$ and consider them spanning an equilateral triangle. As $|\mu_{i,j}| \leq \frac{1}{2}$, this construction maximizes the value of α_2. The height of this triangle is $\sqrt{\frac{3}{4}}$, and hence $\|\mathbf{b}_{i+1}^*\|^2 \geq \frac{3}{4}\|\pi_{i+1}(\mathbf{b}_{i+1})\|^2$. Combining with the equalities above we get $\alpha_2 = \frac{4}{3}$ and thus,

$$\|\mathbf{b}_i^*\|^2 \leq \frac{4}{3}\|\mathbf{b}_{i+1}^*\|^2.$$

Similarly from the construction of a tetrahedron with edges of unit length we get the height $\sqrt{\frac{2}{3}}$ resulting in $\alpha_3 = \frac{3}{2}$ for $k = 3$.

The values of α_k can be bounded above by the Hermite constants γ_k which are defined as

$$\gamma_k = \sup\{\lambda_1(L)^2 \det(L)^{\frac{-2}{k}} \mid L \text{ a lattice of rank k}\}. \tag{3}$$

Exact values of the constants are only known for $k \leq 8$ (see [3]), and we will be using both α_k and γ_k without further specifications. With $j \leq k-1, i+j \leq n$, we get

$$\|\mathbf{b}_i^*\|^2 \leq \alpha_k \|\mathbf{b}_{i+j}^*\|^2.$$

Inductive application of this bound in steps of $k-1$ vectors with $j \leq l(k-1)$ yields

$$\|\mathbf{b}_i^*\|^2 \leq \alpha_k^l \|\mathbf{b}_{i+j}^*\|^2. \tag{4}$$

Proposition 4. *A Korkin-Zolotarev reduced basis can be obtained by at most* $\sqrt{n}^{n+o(n)} + \mathcal{O}(n^4 \log(B))$ *arithmetic steps on integers no longer than* $\mathcal{O}(n \log(B))$ *bits.*

To find a Korkin-Zolotarev reduced basis we choose the shortest vector in L, then find the shortest vector in L_2, which is the shortest vector in $\text{span}(\mathbf{b}_1)^\perp \subset L$. Then we proceed to the shortest vector in $L_3, ..., L_{m-1}$. Size reductions must also be carried out, potentially requiring vectors to be sorted again as lengths have decreased.

2.6 Block Korkin-Zolotarev reduced lattice basis

Schnorr develops a hierarchy of lattice reduction methods stretching from LLL to Korkin-Zolotarev reduction in [19]. He introduces *block Korkin-Zolotarev reduced* lattice basis, and a corresponding method that applies Korkin-Zolotarev reduction to blocks of the lattice basis in a trade-off between algorithmic speed and bounds for the length of lattice vectors. The resulting bases are locally Korkin-Zolotarev reduced, meaning that for block size $1 < \beta < m$, all blocks

$$\pi_i(\mathbf{b}_i), \pi_i(\mathbf{b}_{i+1}), ..., \pi_i(\mathbf{b}_{i+\beta-1})$$

are Korkin-Zolotarev bases for $i = 1, ..., m - \beta + 1$. This implies that $\mathbf{b}_i^* = \pi_i(\mathbf{b}_i)$ is the shortest non-zero vector in the block $\pi_i(L(\mathbf{b}_i, ..., \mathbf{b}_{\min(i+\beta-1,m)}))$, and we call this a β-BKZ basis. With blocks of size β, we step $\beta - 1$ vectors at a time, resulting in the following bounds from (2) and (4),

$$\gamma_\beta^{-2^{\frac{i-1}{\beta-1}}} \leq \|\mathbf{b}_i^*\|^2 \lambda_i(L)^{-2} \leq \gamma_\beta^{2^{\frac{m-i}{\beta-1}}} \quad \text{for } i = 1, ..., m$$

and

$$\frac{4}{i+3}\gamma_\beta^{\frac{i-1}{\beta-1}} \leq \|\mathbf{b}_i\|^2 \lambda_i(L)^{-2} \leq \gamma_\beta^{2^{\frac{m-1}{\beta-1}}} \frac{i+3}{4} \quad \text{for } i = 1, ..., m.$$

Another relaxation of the requirements which allows proving polynomial running time is to loosen the strict ordering of vectors to produce a *semi block Korkin-Zolotarev reduced* lattice basis. The LLL algorithm is a semi BKZ method with $\beta = 2$, that would have been a BKZ and not a semi BKZ method if the factor $\frac{1}{4} < \delta < 1$ in (1) had been replaced by $\delta = 1$. The more generalized algorithm with larger block size will also obtain a running time polynomial in the dimension of the lattice. This will be discussed at the end of this section.

The BKZ reduction algorithm initially uses LLL reduction to find a short basis. Then it alternates between exhaustive search for short vectors in a block and further LLL reduction. In this process the number of vectors satisfying the BKZ condition is counted (variable j in Algorithm 2), and when this number reaches $\dim(\mathbf{b}_1, ..., \mathbf{b}_m) - 1$, a BKZ reduced basis is output.

The underlying idea for the exhaustive search method ENUM is that in the block $\mathbf{b}_j, ..., \mathbf{b}_k$ all lattice vectors $\mathbf{v} = \sum_{s=j}^{k} v_s \mathbf{b}_s$ shorter than $\mathbf{b}_j$ satisfy

$$\|\mathbf{v}\|^2 = \sum_{s=j}^{k} \left(\sum_{i=s}^{k} u_i \mu_{i,s} \right)^2 \|\mathbf{b}_s^*\| < \|\mathbf{b}_j\|^2$$

analogous to the inequality $\|\mathbf{b}_i\| \leq \sum_{j=1}^{i} \mu_{i,j}^2 \|\mathbf{b}_j^*\|^2$ with $\mu_{i,i} = 1$. We want to find $(u_j, ..., u_k)$ minimizing $\mathbf{v}$ and for this we find the maximal range of each u_i. Starting with index k and $\mu_{k,k} = 1$ we have

$$|u_k| < \frac{\|\mathbf{b}_1\|}{\|\mathbf{b}_k^*\|}.$$

and for lower indices

$$\left(\sum_{i=j}^{k} u_i \mu_{i,j} \right)^2 \|\mathbf{b}_j^*\| < \|\mathbf{b}_1\|^2 - \sum_{v=j+1}^{k} \left(\sum_{s=v}^{k} u_s \mu_{s,v} \right)^2 \|\mathbf{b}_v^*\|^2 \quad \text{for } j = 1, ..., n-1$$

and then $\| \sum_{s=j}^{k} u_s \mu_{s,j} \| < \frac{\|\mathbf{b}_1\|}{\|\mathbf{b}_j^*\|}$ bounding the number of possible values of u_j given $u_{j+1}, ..., u_k$ to at most $\lfloor 2\frac{\|\mathbf{b}_1\|}{\|\mathbf{b}_j^*\|} \rfloor + 1$. With the restriction that we

Algorithm 2 BKZ algorithm

Input: Lattice basis $\mathbf{b}_1, ..., \mathbf{b}_m$, $\frac{1}{2} < \delta < 1, 2 < \beta < m$
Perform LLL$(\mathbf{b}_1, ..., \mathbf{b}_m, \delta)$ and set $z = 0, j = 0$
while $z < m - 1$ **do**
  $j \leftarrow j + 1$, $k \leftarrow \min(j + \beta - 1, m)$
  if $j = m$ **then**
    $j \leftarrow 1, k \leftarrow \beta$
  ENUM(j,k):
    find the vector $(u_j, ..., u_k) \in \mathbf{Z}^{k-j+1} - \mathbf{0}^{k-j+1}$ minimizing $c_j(u_j, ..., u_k) = \sum_{s=j}^{k} \left(\sum_{i=s}^{k} u_i \mu_{i,s}\right)^2 \mathbf{b}_s$, return the minimal value $\bar{c}_j$ and $\mathbf{b}_j^{new} \leftarrow \sum_{s=j}^{k} u_s \mathbf{b}_s$
  $h \leftarrow \min(k + 1, m)$
  if $\delta \|\mathbf{b}_j\| > \bar{c}_j$ **then**
    Perform LLL$(\mathbf{b}_1, ..., \mathbf{b}_{j-1}, \mathbf{b}_j^{new}, \mathbf{b}_j, ..., \mathbf{b}_h, \delta)$, $z \leftarrow 0$
  else
    $z \leftarrow z + 1$
    Perform LLL$(\mathbf{b}_1, ..., \mathbf{b}_h, 0.99)$ at stage $h - 1$
Output: $\mathbf{b}_1, ..., \mathbf{b}_m$, a basis β-reduced with δ.

only search for a shorter vector when $\|\mathbf{b}_j\|^2 \leq 2\|\mathbf{b}_{j+1}^*\|^2$ we get that the number of possible choices for $u_j, ..., u_k$ is

$$\begin{aligned}\prod_{s=2}^{k}(\lfloor 2\|\mathbf{b}_j\|/\|\mathbf{b}_s^*\|\rfloor + 1) &\leq 3^{k-1} \prod_{s=j}^{k} \frac{\|\mathbf{b}_1\|}{\|\mathbf{b}_s^*\|} \\ &\leq 3^{n-1}(\sqrt{2}\lambda(L_2))^{k-1} d(L_2)^{-1} \\ &\leq (18\gamma_{k-1})^{\frac{k-1}{2}},\end{aligned}$$

using the definition of the Hermite constant γ_{k-1}. If a shorter vector $\mathbf{v}$ is found it is added to the set of vectors as $\mathbf{b}_j^{\text{new}}$, which is then linearly dependent. Reduction will produce a zero vector which is removed to get a linearly independent set of m vectors as before.

With the relation $\lim_n \sup \gamma_k/k \leq (e\pi)^{-1}$ [19] we get that $(18\gamma_{k-1})^{\frac{k-1}{2}} = \sqrt{k}^{k+o(k)}$ bounding the running time for the ENUM procedure.

Time analysis of the LLL algorithm shows that with $\delta < 1$, the number of vector swaps is limited to $n^2 \log(B)$. For every exchange of vectors in BKZ, we perform up to n^2 operations in the LLL procedure and/or $\sqrt{n}^{k+o(k)}$ operations in the ENUM procedure. The running time of the BKZ algorithm is then $\mathcal{O}((n^2 + \sqrt{k}^{k+o(k)})n^2 \log(B))$.

2.7 Random sampling reduction

The ENUM method in BKZ reduction does exhaustive search in low dimension. Schnorr's random sampling method [20] uses a random sampling in high dimension to find a shorter lattice vector, which surprisingly has better performance than the low dimensional exhaustive search.

In random sampling we are looking for short vectors $\pi_j(\mathbf{b}) = \sum_{i=j}^{n} \mu_{i,j}\mathbf{b}_j^*$ in $\pi_j(L(\mathbf{b}_j, ..., \mathbf{b}_n))$. To find $\|\pi_j(\mathbf{b})\|^2 = \sum_{i=j}^{n} \mu_{i,j}^2 \|\mathbf{b}_i^*\|^2$ we need to find small coefficients of $\mu_{j+1,j}, ..., \mu_{n,j}$. Schnorr notes that among the orthogonalized vectors $\mathbf{b}_j^*, ..., \mathbf{b}_n^*$, the vectors at higher indices are shorter than the vectors at lower indices, giving small $\mu_{i,j}$ at low i a higher impact on the length of $\|\pi_j(\mathbf{b})\|^2$.

Let the single indexed $\mu_j = \frac{\langle \mathbf{b}, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}$ be the projection of the sampled $\mathbf{b}$ along $\mathbf{b}_j$. For $t_i \in \mathbf{Z}$ and $1 \leq u < n$ we are sampling a vector $\mathbf{b} = \sum_{i=1}^{n} t_i \mathbf{b}_i = \sum_{i=1}^{n} \mu_i \mathbf{b}_i^*$ such that

$$|\mu_i| \leq \begin{cases} \frac{1}{2} \text{ for } i < n-u \\ 1 \text{ for } n-u \leq i < n \end{cases}, \quad \mu_n = 1.$$

This leaves no choice for $i < (n-u)$ in a size reduced basis, but for $(n-u) \leq i < n$ we have two (and sometimes three) choices, leading to 2^u possible lattice vector outputs of the method. The method is given in Algorithm 3 which, counting the basic operation, clearly has a running time $\mathcal{O}(n^2)$.

Algorithm 3 Sampling algorithm

Input: Lattice basis $\{\mathbf{b}_1, ..., \mathbf{b}_n\}$, coefficients $\mu_{i,j}$, u
$\mathbf{b} \leftarrow \mathbf{b}_n$
for j from 1 to $n-1$ **do**
  $\mu_j \leftarrow \mu_{n,j}$
for i from $n-1$ to 1 **do**
  if $i < n-u$ **then**
    Select $\mu \in \mathbf{Z}$ such that $|\mu_i - \mu| \leq \frac{1}{2}$
  else
    Select $\mu \in \mathbf{Z}$ such that $|\mu_i - \mu| \leq 1$
  $\mathbf{b} \leftarrow \mathbf{b} - \mu\mathbf{b}_i$
  for j from 1 to i **do**
    $\mu_j \leftarrow \mu_j - \mu\mu_{i,j}$
Output: $\mathbf{b}, \mu_1, ..., \mu_n$

As the coefficients are updated $(n-i)$ times, the distribution of μ_i will be nearly uniform for small i. We assume in the *randomness assumptions* (RA) that μ_i is uniformly distributed in the intervals $[-\frac{1}{2}, \frac{1}{2}]$ for $i < n-1$ and $[-1, 1]$ for $n-u \leq i < n$. In addition we might assume that the coefficients μ_i, μ_i' of distict vectors $\mathbf{b}, \mathbf{b}'$ are independent for distinct i.

We also need the *geometric series assumption* (GSA) which is satisfied if $\frac{\|\mathbf{b}_i^*\|^2}{\|\mathbf{b}_1\|^2} = q^{i-1}$ for $i = 1, ..., n$, a geometric series with quotient q, $\frac{3}{4} \leq q < 1$. In practice we can only approximate a geometric series, and do this by requiring that

$$\sum_{i=1}^{n} \left(\frac{\|\mathbf{b}_i^*\|^2}{\|\mathbf{b}_1\|^2} - q^{i-1} \right) \mu_i^2 < 0.1$$

for random and independent $\mu_i \in [-\frac{1}{2}, \frac{1}{2}]$.

If the GSA and RA hold, we can estimate the probability of finding a shorter vector by random sampling. As the first vectors have the greatest impact on the length of the sampled vector, we want to keep the projection onto these vectors small. Specifically we are seeking projections of the first $k < n - u$ vectors such that

$$|\mu_i|^2 \leq \frac{1}{4} q^{k-i}. \tag{5}$$

Given that RA holds, the projections are uniformly distributed in the interval $[-\frac{1}{2}, \frac{1}{2}]$ and the probability that (5) holds for k vectors is

$$\prod_{i=1}^{k} q^{\frac{k-i}{2}} = q^{\binom{k}{2}/2} = q^{\frac{k(k-1)}{4}}. \tag{6}$$

Schnorr claims in his article that bases reduced with an LLL-type reduction algorithm such as BKZ closely approximate GSA and that the sampling algorithm (Algorithm 3) gives an approximation to RA [20]. He also states that lattice reduction gets harder as the basis approximates a geometric series making GSA the worst case for lattice reduction.

To advance the reduction at this point, the sampling algorithm is used to sample a number of potentially shorter vectors. We require new short vectors to satisfy $\|\mathbf{b}\|^2 < 0.9\|\mathbf{b}_j\|^2$. To estimate the quotient q and average number of necessary samples, we set $\gamma_k \leq \frac{k}{6}$ for $k \geq 24$, which leads to $q \geq (\frac{6}{k})^{\frac{1}{k}}$. From this and (6) we see that the probability of finding projections $u_1, ..., u_k$ that satisfies (5) is $(\frac{6}{k})^{\frac{k-1}{4}}$, so we would like to sample at least $(\frac{k}{6})^{\frac{k-1}{4}}$ vectors. Choosing u to be the minimal integer such that $2^u \geq 2(\frac{k}{6})^{\frac{k-1}{4}}$ we will then need to sample on average $(\frac{k}{6})^{\frac{k}{4}}$ vectors and can do so in $\mathcal{O}(n^2(\frac{k}{6})^{\frac{k}{4}})$ average time. The method can in this manner find vectors that are as short as $(\frac{k}{6})^{\frac{n}{2k}}$ times the minimal length of a vector in the lattice. If the shortest vector is longer than this bound, the random sampling algorithm should find a shorter (by a factor $\sqrt{0.9}$) vector on average by the time given above.

When a shorter vector $\mathbf{b}$ is found by the sampling algorithm, we insert it into the lattice basis $\mathbf{b}_1, ..., \mathbf{b}_{j-1}, \mathbf{b}, \mathbf{b}_j, ..., \mathbf{b}_{n-1}$. We safely remove the last vector $\mathbf{b}_n$ from the basis as our construction of $\mathbf{b}$ starts with $\mathbf{b} = \mathbf{b}_n$. Unlike the regular BKZ method we do not need to work with a linearly independent set and eliminate a redundant vector.

Schnorr notes that in practice, reduction of the new basis with the sampled vector triggers sporadic jumps. Without introducing sampled vectors the BKZ reduction shows no such behavior. The reason for this difference is explained by the fact that a BKZ reduced basis closely approximates GSA, while inserting a shorter sampled vector brings the basis away from GSA. The sampled vector $\mathbf{b}$ likely has very small values for the first $\mu_j, \mu_{j+1}, \mu_{j+2}, ...$ compared to $\mathbf{b}_j$. Consequently the resulting basis has large orthogonal vectors $\mathbf{b}^*_{j+1}, \mathbf{b}^*_{j+2}, ...$ for the same first vectors. BKZ reduction of this basis will then trigger big jumps as BKZ again reduces the lattice to approximate GSA.

The random sampling reduction method finds a shortest vector no longer than $(\frac{k}{6})^{\frac{n}{2k}}$ times the length of the minimal vector in the lattice. Schnorr claims that this factor is less than the fourth root of other methods obtained within the same computational time.

Another method called *Sampling Reduction* is introduced by Buchmann and Ludwig [2]. As not all bases approximate the GSA of Schnorr's method, they introduce a procedure which without use of GSA finds the probability that a shorter vector exists among 2^u vectors. If this probability is above a given limit, up to 2^u vectors are sampled systematically, until a vector shorter by some factor of the currently shortest vector is found or all the vectors have been sampled. Empirical results of the efficiency is limited in [2], but the authors give some theoretical observations. These are flawed in several ways, primarily due to a misreading of Schnorr's article [20].

3 The NTRUEncrypt cryptosystem

3.1 Public key cryptosystems

Public key cryptosystems are a class of cryptosystems allowing secret communication without access to a shared secret key. They consist of algorithms for key generation, encryption and decryption. The key generation algorithm outputs a a pair of public and private keys of which the public key is made publicly available and the private decryption key is kept secret and used only by the decrypter. The encryption algorithm takes the public key and a message as input and outputs a ciphertext. The decryption algorithms takes the private key and the ciphertext as input and outputs the message. Without the private key it should be infeasible to extract any information about the message from the ciphertext. Due to the difference in keys, it is also called asymmetric cryptosystems, in contrast to symmetric cryptosystems where encrypter and decrypter possess the same key, which has to be distributed and kept in secret.

3.2 NTRUEncrypt

The cryptosystem NTRUEncrypt was first presented at Crypto '96, and is patented by NTRU Cryptosystems, Inc. It has since been reviewed by the cryptographic community, and is specified in the IEEE P1363.1 standard. The main benefit of the system is the speed at which key generation, encryption and decryption can be carried out and that it can be efficiently implemented on very limited systems such as single 8-bit processors. The quick key generation allows for the use of "disposable keys" allowing a new key to be created for every transaction. NTRUEncrypt is often referred to as a public key cryptosystem as it operates with private and secret keys. However there exists validly encrypted messages that will fail to decrypt correctly in some parameter sets of NTRUEncrypt, violating the requirements of public key cryptosystems.

NTRUEncrypt is often called a lattice based cryptosystem, meaning that the hard problem of NTRUEncrypt reduces to certain hard lattice problems. It is a probabilistic cryptosystem meaning that a random element is used in encryption such that two encryptions of the same message with the same key will yield different ciphertexts.

NTRU Cryptosystems has also presented the signing algorithm NTRUSign which is based on the same hard problem as NTRUEncrypt. Our focus will however solely be on NTRUEncrypt.

3.3 The Ring

Operations in NTRUEncrypt are carried out in the ring $\mathcal{R} = \mathbf{Z}[X]/(X^N - 1)$, the ring of truncated polynomials of degree $N-1$. To represent a polynomial p we use the element $f \in \mathcal{R}$, $p = f + (X^N - 1)$. We will write elements in $f \in \mathcal{R}$ as polynomials, or vectors

$$f = \sum_{i=0}^{N-1} f_i X_i = [f_0, f_1, ..., f_{N-1}]$$

Definition 3. *The* star multiplication *or* convolution product *is defined as*

$$f * g = h, \quad h_k = \sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{N-1} f_i g_{N+k-i} = \sum_{\substack{i+j \equiv k \\ (\mathrm{mod}\ N)}} f_i g_j.$$

Multiplication of a scalar p and a polynomial f will be written as pf. Computing the star multiplication would normally require N^2 multiplications, but in most of NTRUEncrypt operations at least one of the polynomials has a large number of zero coefficients allowing faster operations. The term *small polynomial* will be used to denote polynomials with a large portion of its coefficients being zero, and a specified number of coefficients

being 1 and possibly -1. We will also need to multiply and take inverses of polynomials modulo an integer q, and reduce coefficients as expected. Elements and operations will be in the ring $\mathcal{R}_q = \mathcal{R}/q = \frac{\mathbf{Z}/q\mathbf{Z}[X]}{(X^N-1)}$. As star multiplication in the ring $\mathcal{R}_q$ is the only polynomial multiplication we will be concerned with, we simplify notation by dropping equivalences and modulo q such that $f * h = pg$ is used instead of $f * h \equiv pg \pmod{q}$.

For NTRUEncrypt polynomials we will define the *width* of an element $f \in \mathcal{R}$ as

$$|f|_\infty = \max_{1 \le i \le N} f_i - \min_{1 \le i \le N} f_i,$$

serving as a modified L^∞ norm.

3.4 NTRUEncrypt basics

We will first present NTRUEncrypt in its original form (initially the name was just NTRU) to keep the inital presentation simple before introducing new and enhanced parameters and padding schemes.

3.4.1 Security levels and parameter settings

In order to create a pair of keys in the NTRUEncrypt cryptosystem, a proper security level must be chosen. Several security levels with corresponding parameter settings have been specified, for a list see [11]. The security level is specified by the prime number N. Other parameters are the small and large moduli p and q, which are relatively prime integers and the polynomial generation parameters d_f, d_g, d_r and d_m, which specify the number of non-zero coefficients in the polynomials f, g, r and m and the polynomial spaces $\mathcal{D}_f, \mathcal{D}_g$, $\mathcal{D}_r$ and $\mathcal{D}_m$ respectively. Binary polynomials in $\mathcal{D}_f$ have d_f coefficients set to 1 and the rest set to 0, likewise for polynomials in $\mathcal{D}_g$ and $\mathcal{D}_r$. For polynomials in the message space $\mathcal{D}_m$ we require that the number of 1's is larger than d_m and smaller than $N - d_m$.

3.4.2 Key generation

With parameters from the selected security level, choose polynomials $f \in \mathcal{D}_f$ and $g \in \mathcal{D}_g$. Let f_p^{-1} and f_q^{-1} be the inverses of f modulo p and q, and compute $h = f_q^{-1} * pg$. The private key is f and the public key is h. The chosen security level and corresponding parameters are publicly known.

3.4.3 Encryption

The encrypter wants to encrypt the message $m \in \mathcal{D}_m$, and chooses a random blinding value $r \in \mathcal{D}_r$. Encryption is done by computing $e = r * h + m$.

3.4.4 Decryption

The decrypter first computes

$$a \equiv f * e \equiv f * f_q^{-1} * pg * r + f * m \equiv pr * g + f * m \pmod{q}, \quad (7)$$

then takes $a \pmod{p}$ to get

$$b \equiv a \equiv f * m \pmod{p},$$

which will then retrieve the message m when star multiplied by f_p^{-1}

$$c \equiv f_p^{-1} * f * m \pmod{p}.$$

Hopefully $c = m$, but it might happen that coefficients grow too large in (7), leading to decryption failures. When computing a we will normally reduce the coefficients into the interval $[0, q-1]$, but if decryption fails we might have to reduce the coefficients into other intervals $[A, A+q-1]$ for various A, a procedure known as *centering*. If a centering does not produce a valid decryption it is called a *wrap failure*. If no A producing a valid decryption can be found we have a *gap failure* and we know that $|a|_\infty = |pr * g + f * m|_\infty \geq q$. If $|a|_\infty < q$ we may have wrap failures, but there exists a centering producing a valid decryption.

For large values of q the chance of decryption failures decreases, and can be eliminated completely by choosing q larger than the maximum possible value of $|pr * g + f * m|_\infty$. For such q, no centering will be needed. When using parameters from the current standard [29], the chance of decryption failure is estimated to be less than 2^{-104} for the security setting giving 80-bit security [11].

3.5 Multiple transmission attack

If a padding scheme is not used and a message m is encrypted several times with different random polynomials r, an attacker can recover m. If the attacker gathers several $e_i = r_i * h + m$, he may compute $(r_i - r_1) * h^{-1}$ to find $r_i - r_1$. As the coefficients of r are small, $r_i - r_1$ can be found exactly. A few such relations lets the attacker recover large parts of r making it possible to find the rest by brute force. Then the single message m can be decrypted.

3.6 Decryption failure attack

The possibility of failure in decrypting an NTRUEncrypt ciphertext enables new types of attacks and requires new security proofs [8]. If decryption fails due to incorrect centering (wrap failure), new centerings must be tried until one is found that allows successful decryption of the message. Some knowledge about the private key can be gathered by analysis of decrytion

time when more attempts with different centerings are needed. If no suitable centering is found a gap error has occured, leaking even more information about the private key.

Decryption failure attacks can most easily be described using unpadded early versions of NTRUEncrypt, although attacks exist for some suggested paddings as well [18]. The attacker encrypts several messages m with different blinding polynomials r until a pair (m, r) that produces an invalid decryption occurs. On decryption at least one of the coefficients of $pg*r+f*m$ falls outside all intervals of width q. If encryption of $(0, r)$ produces a valid decryption, the attacker can check several $(\bar{m}, r)$ and check if they decrypt correctly or not. The aim is to find several pairs of $\bar{m}$ flipping one bit such that one produces a valid decryption while the others is not decryptable. The coefficients in decryption after multiplication by f are

$$c_i = \sum_{j=0}^{N-1} \bar{m}_j f_{i-j} + p \sum_{j=0}^{N-1} r_j g_{i-j}$$

where some c_i will fall outside the correct interval. If we know that changing $\bar{m}_j$ from 1 (or -1 in older trinary versions) to 0 makes decryption possible, we know that f_{i-j} is 1 (or -1 respectively). If it decrypts correctly with $\bar{m}_j = 0$ and not with 1 or -1, we know that the value of f_{i-j} is -1 or 1 respectively. Thus finding just one pair (m, r) producing an invalid decryption will in most cases let us recover large parts of f. As we know the number of 1's (and possibly -1's) in f, exhaustive search can be used to decide the remaining coefficients of f. Using the initial parameters of NTRUEncryp, Proos [18] empirically finds this attack to be efficient and successful all pairs (m, r) where decryption fails.

In decryption we obtain $f * e = pg * r + f * m$. The chance of wrap and gap failures is related to the width of $|pg*r+(1+pF)*m|_\infty$. If $|pg*r+(1+pF)*m|_\infty < q$, no gap failures may occur, there will always exits a centering reducing the coefficients into the correct interval. For binary polynomials $a, b \in \mathcal{R}$ with d_a, d_b ones, we have $|a * b|_\infty \leq \min(d_a, d_b)$, and consequently $|pg*r+(1+pF)*m|_\infty \leq p\min(d_g, d_r)+1+p\min(d_f, d_m)$. As $p = 2$, $d_r \leq d_g$, $d_f = d_r$ and d_m is on average $N/2$, we get $|pg*r+(1+pF)*m|_\infty < 1+4d_f$.

The NTRU team now recommends the use of parameters that eliminate the chance of decryption failures [11]. We couldn't find any published decryption failure attack against the padding scheme recommended in [29], but several such attacks against other proposed paddings exist [18].

3.7 Evolution of parameters

From the beginning it was suggested without a given reason to let N be a prime. Gentry [5] shows that if N is not taken to be a prime number, security is drastically weakened.

The small modulus p and the large modulus q must be relatively prime in $\mathcal{R}$, or equivalently $X^N - 1, p, q$ must generate the unit ideal in $\mathbf{Z}[X]$. If $\gcd(q, p) > 1$, security is decreased, and if $p|q$ the encrypted message satisfies $e \equiv m \pmod{p}$ and is hence completely insecure. The large modulus q was first chosen to be a power of 2, such as 128 or 256, for their computational efficiency, while [11] suggests using a prime number that has an order at least $(N-1)/2$ modulo N and which is large enough to make the chance of decryption failures vanishingly small. The small modulus was first set to $p = 3$, but it was later recommmended to use the polynomial $X+2$ with some necessary alterations of the decryption proceduce. Now it is recommended to set $p = 2$ [29].

Some changes to key creation and decryption have also been introduced by setting the polynomial f to be $f = 1 + p * F$ with $F \in \mathcal{D}_f$. With f on this form we know that f_p^{-1} exists, but as the decryption process changes, the need for f_p^{-1} is eliminated. At decryption we will have

$$a \equiv f * e \equiv (1 + p * F) * (pr * h + m) \equiv p(r * g + F * m) + m \pmod{q}$$

and the need to multiply by the inverse f_p^{-1} vanishes as $a \pmod{p}$ reveals the message polynomial since $p = 2$ and m is chosen to be a binary message polynomial. If an incorrect centering is used the output bits will be flipped as the coefficients are first reduced by an odd number and then by 2.

Polynomials were first chosen to be trinary with a specified number of coefficients set to 1 and -1 and the rest set to 0. This was later changed to binary polynomials with a set number of 1's among the coefficients as described above. A third way to choose the coefficients of $\mathcal{D}_f$ and $\mathcal{D}_r$ is by *product form polynomials* on the form $a_1 * a_2$ or $a_1 * a_2 + a_3$ where a_1, a_2, a_3 are binary polynomials. This construction allows faster multiplication on the cost of memory usage. Coefficients of this polynomial will have most coefficients equal to 0, a large portion of the remaining equal to 1 and a few coefficients larger than 1, the frequency and size increasing with the degree of the resulting polynomial [11]. For decryption to work properly, f has to be invertible modulo q.

The current standard for NTRUEncrypt is given in [29], which analyses in this paper will be built upon. Other parameters have later been suggested by the NTRU team in [11]. The most important changes are values of q eliminating the possibility of decryption failures and changes to the polynomial spaces. Instead of choosing $d_f = d_g = d_r$, they suggest choosing a lower $d_f = d_r$ for efficiency, and increasing d_g to $\frac{N}{2}$ as this has no impact on efficiency, but will increase the lattice constant governing the hardness of lattice reduction of the public key. The non-binary form of product form polynomials leads to changes for some attacks, but as g is binary, most attacks can be applied with some modification by attacking h^{-1} instead of h.

3.8 Protection against adaptive chosen ciphertext attaks

Among other requirements, a cryptosystem should resist adaptive chosen ciphertext attacks. In such attacks the attacker gains information from the decrypter when messages can not be correctly decrypted and uses this information in choosing subsequent ciphertexts, hoping to gain knowledge about the message or the secret key. The presence of decryption failures leaves NTRU especially prone to such attacks as they leak knowledge of the secret key (see section 3.6). To protect against adaptive chosen ciphertext attacks, an appropriate padding scheme which reduces the attacker's ability in freely choosing ciphertexts is needed. The NTRUEncrypt padding scheme NAEP uses two hash functions in padding the message. One is a cryptographic strong hash function, while the other is based on a hash function and called a *mask generation function* (MGF).

When encrypting, we start by concatenating the binary message string m, the length l of m, random data b and zeros $\bar{0}$,

$$M = b|l|m|\bar{0}$$

to a total length of N bits.

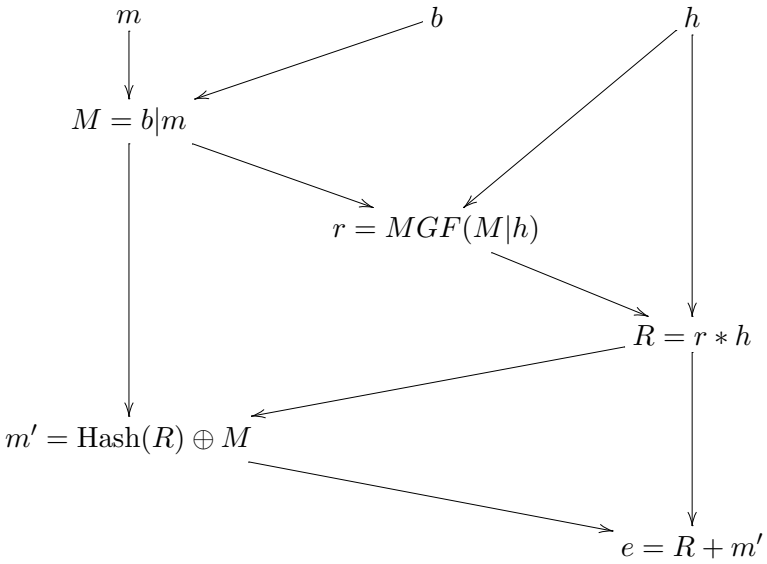


Figure 1: Encryption with NAEP

The MGF takes as input a binary string consisting of m and b as above, bits calculated from the public key h and some constant bits given by the chosen parameter set. This input is hashed and the output functions as a seed to a random number generator, and MGF outputs the polynomial r based on the random number and the parameters for choosing polynomials from $\mathcal{D}_r$.

We compute $R \equiv r * h \pmod{q}$ as in unpadded encryption. We take R modulo 2 and convert the result to a binary string which is the input of the hash function. Then we xor the output of the hash function with M to produce the string m_s which we convert to the binary polynomial m'. We add m' to R to produce the encrypted message e. The process is shown in Figure 1.

On decryption we apply the regular decryption process to recover m'_c from e by the computation

$$f * e = f * f^{-1} * pg + (1 + pF) * m'_c = m'_c \pmod{p}.$$

We can then calculate the polynomial $R_c = e - m'_c$, which we reduce modulo 2 and convert to a bit string which we hash as in the encryption process. We xor the output of the hash function with m'_c to recover M_c of which the message m_c is a subset. To check that decryption has been successful, we call MGF with M_c and the public quantities to obtain the polynomial r_c, compute $r_c * h$ and check that it is equal to R_c. The process is shown in Figure 2. It can be shown [9] that this padding scheme provides security against adaptive chosen ciphertext attacks even in the presence of decryption failures.

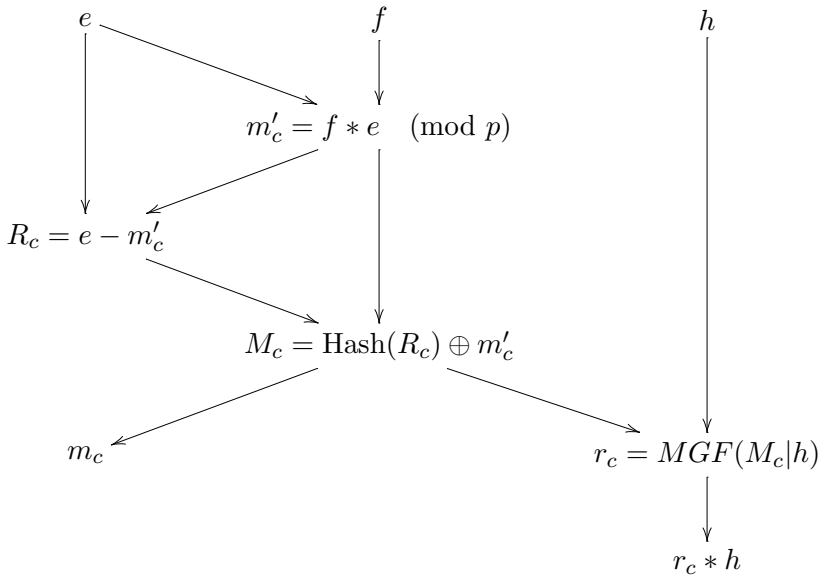


Figure 2: Decryption with NAEP

In the original NTRUEncrypt system an attacker could freely choose r and m, but with this construction an attacker is limited to choosing an r produced by m and the random bits thwarting decryption failure and

adaptive chosen ciphertext attacks. Adding Hash(R) to M ensures that unless an attacker knows all the bits of m'_c, he effectively has no knowledge of m.

4 Key Encapsulation Mechanism

Public key cryptosystems are primarily used to encrypt short messages. We can use a public key cryptosystem as a secure channel for exchange of a symmetric key and transmit a message encrypted with the symmetric key. This construction is called a hybrid cryptosystem and benefits from the simple key distribution of the public key cryptosystem and the speed of encryption and decryption in the symmetric cryptosystem.

It might be easier to encrypt random messages than to encrypt an arbitrary message. The idea is to encrypt random messages from which we will derive the symmetric key. This is called a *key encapsulation mechanism*. A good hash function is used to derive the symmetric key, ensuring that partial break of the random bitstring does not reveal any information about the symmetric key.

4.1 NTRU-KEM

No KEM is specified for NTRUEncrypt, but a KEM variation of NAEP is drafted by the NTRU team in [30] and another suggestion can be found in [27]. We will propose and analyze a simplification of the NTRU team proposal, using parts of the NAEP scheme. A simple sketch of the encryption process is given in Figure 3. The main differences between this KEM and NAEP is that the whole message μ consists of random bits and only one hash function is needed in the encryption process, while another hash function scrambles μ to produce the key.

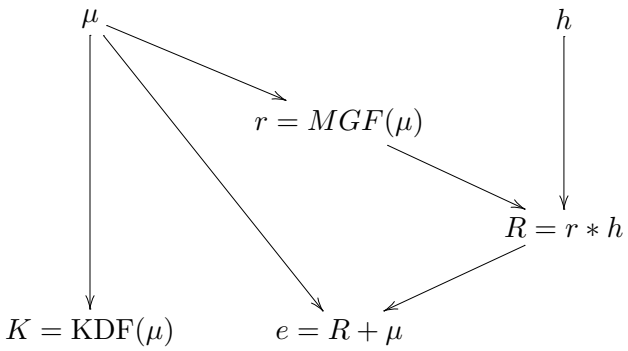


Figure 3: NTRU-KEM proposal

The two hash functions are the *key derivation function* (KDF) and one

which is part of the mask generation function (MGF). Specifically MGF consists of a hash function H and the polynomial generating function `genr` with

$$\mathrm{H} : \{0,1\}^N \to \{0,1\}^N,$$

$$\texttt{genr} : \{0,1\}^N \to \mathcal{D}_r,$$

$$\mathrm{MGF} : \{0,1\}^N \to \mathcal{D}_r$$

such that $\mathrm{MGF}(\mu) = \texttt{genr}(\mathrm{H}(\mu))$.

To produce a key using NTRU-KEM, the encrypter generates a random input string $\mu = \{0,1\}^N$ and encrypts it. The decrypter recovers μ and both parties can compute the key K.

Encryption	Decryption
$r = \mathrm{MGF}(\mu)$	$\tilde{\mu} = f * e \bmod 2$
$e = m + r * h$	if $\mathrm{MGF}(\tilde{\mu}) * h = e - \tilde{\mu}$
$K = \mathrm{KDF}(\mu)$	then $K = \mathrm{KDF}(\tilde{\mu})$

For ease of notation we view bitstrings of length N as binary polynomials of degree N.

4.2 Attack game

We show that our NTRU-KEM is secure in the random oracle model. We model all hash functions as random oracles, which produce a truly random response to any new query, and the same response if a query is repeated. This functionality is implemented with oracles keeping a list of all queries and responses, creating a new entry for new queries and responding by the stored value on repeated queries.

We prove security using "Sequences of Games" [24] bounding the *advantage* of the adversary $\mathcal{A}$, a measure of $\mathcal{A}$'s ability in making a correct guess of a chosen bit. In different games we will use different simulators interacting with the adversary $\mathcal{A}$.

In all games the simulator first runs key generation to obtain a key pair (h, f), finds a random bitstring $\mu \in \{0,1\}^N$ and a random bit b. We will denote sampling x uniformly at random from the set S by $x \xleftarrow{r} S$. We also find the value of K using

$$K = \begin{cases} \mathrm{KDF}(\mu) & \text{if } b = 0, \\ \{0,1\}^N & \text{if } b = 1. \end{cases}$$

The goal of the adversary is to correctly guess the bit b.

The adversary $\mathcal{A}$ is given h, K and the encryption e of μ and may present ciphertext messages e' for the simulator to decrypt. Both the adversary and the simulator has access the random oracles MGF and KDF.

Game 0

1. $(h, f) \leftarrow \texttt{KeyGen}, \quad \mu \xleftarrow{r} \{0,1\}^N, \quad b \xleftarrow{r} \{0,1\}, \quad K' \leftarrow \{0,1\}^N.$
2. $K \leftarrow \begin{cases} \mathrm{KDF}(\mu) & \text{if } b = 0, \\ K' & \text{if } b = 1. \end{cases}$
3. $r \leftarrow \mathrm{MGF}(\mu), \quad e = r * h + \mu.$
4. Send (h, K, e) to $\mathcal{A}$.
5. $\mathcal{A}$ queries the simulator with any ciphertext except e.
6. $\hat{b} \leftarrow \mathcal{A}$.
7. Output 1 if $\hat{b} = b$, otherwise output 0.

The response to a decryption query is either K, or failure if it is not possible to produce a valid decryption, which means that on a ciphertext e' with decryption μ' and $r' = \mathrm{MGF}(\mu')$, $e' \neq r' * h + \mu'$ due either to a decryption failure or a malformed ciphertext.

Game 1

Game 1 is the same as Game 0 except that from now on the simulator simulates the random oracles MGF and KDF.

Game 2

Game 2 is identical to Game 1 except that all queries μ' to MGF and the corresponding e' are stored in a list, and even if e' causes a decryption failure, the simulator still responds by $K' = \mathrm{KDF}(\mu')$.

Game 3

Game 3 is equal to Game 2 except that the simulator always replies failure on a decryption query unless the corresponding μ' is previously queried to MGF by $\mathcal{A}$.

Game 4

In Game 4 the simulator keeps a list of queries as follows: Entries of (μ, r, e, K) are stored such that $r = \mathrm{MGF}(\mu), e = r * h + \mu$ and $K = \mathrm{KDF}(\mu)$. On every query, the simulator checks the list for the given value. If it already exists, the simulator responds by K' on queries to KDF and decryption and r' on queries to MGF. If no such entry can be found, failure is returned on decryption queries while MGF and KDF responds as usual and a new entry

for (μ, r, e, K) is added to the list.

In the games we consider the following events:

- T_2: A properly generated ciphertext causing a decryption failure is submitted.
- T_3: A valid ciphertext c' is submitted, but $\mathrm{MGF}(\mu')$ has not been queried.
- T_4: The adversary queries KDF with μ.

From $\mathcal{A}$'s view Game 0 and Game 1 are identical. In Game 2 the event T_2 may occur. The probability of decryption failure is dependent on the parameter set used, and is negligible or non-existent for current standardized and recently recommended sets. Let ϵ_{fail} be the probability of a decryption failure and Z_{MGF} be the number of oracle calls to MGF. We will then have that $\Pr[T_2] \leq Z_{\text{MGF}}\epsilon_{\text{fail}}$.

$$\begin{aligned}
&\Pr[G_1 = 1] - \Pr[G_2 = 1] \\
&= (\Pr[G_1 = 1|T_2] - \Pr[G_2 = 1|T_2]) \Pr[T_2] \\
&\quad +(\Pr[G_1 = 1|\neg T_2] - \Pr[G_2 = 1|\neg T_2]) \Pr[\neg T_2] \\
&= (\Pr[G_1 = 1|T_2] - \Pr[G_2 = 1|T_2]) \Pr[T_2] \\
&\leq \Pr[T_2] \leq Z_{\text{MGF}}\epsilon_{\text{fail}}.
\end{aligned}$$

For Game 3 we need to bound the probability of event T_3. The simulator responds by failure to all decryption queries for which MGF has not been queried on the plaintext, and the event T_3 is such a decryption query. As there are $|\mathcal{D}_r| = \binom{N}{d_r}$ different r's we get that $\Pr[T_3] = |\mathcal{D}_r|^{-1} Z_D$ where Z_D is the number of decryption queries. As the event T_3 distinguishes between games 2 and 3 we have:

$$\begin{aligned}
&\Pr[G_2 = 1] - \Pr[G_3 = 1] \\
&= (\Pr[G_2 = 1|T_3] - \Pr[G_3 = |T_3]) \Pr[T_3] \\
&\quad +(\Pr[G_2 = 1|\neg T_3] - \Pr[G_3 = 1|\neg T_3]) \Pr[\neg T_3] \\
&= (\Pr[G_2 = 1|T_3] - \Pr[G_3 = 1|T_3]) \Pr[T_3] \\
&\leq \Pr[T_3] \leq \frac{Z_D}{|\mathcal{D}_r|}.
\end{aligned}$$

From the adversary's view, games 3 and 4 are identical such that $\Pr[G_3 = 1] = \Pr[G_4 = 1]$. In Game 4 the simulator makes an entry with μ, r, e and K and e. As $\mathcal{A}$ can't query this e, he can only gain knowledge of b by querying KDF on μ. This is event T_4. If T_4 does not occur, $\Pr[G_4 = 1] = \frac{1}{2}$ and we

have

$$\begin{aligned}
|\Pr[G_4 = 1] - \frac{1}{2}| &= |\Pr[G_4 = 1|T_4]\Pr[T_4] + \Pr[G_4 = 1|\neg T_4]\Pr[\neg T_4] - \frac{1}{2}| \\
&= |\Pr[G_4 = 1|T_4]\Pr[T_4] + (1 - \Pr[T_4])\frac{1}{2} - \frac{1}{2}| \\
&= |\Pr[G_4 = 1|T_4] - \frac{1}{2}|\Pr[T_4] \\
&\leq \frac{1}{2}\Pr[T_4].
\end{aligned}$$

The event T_4 implies that $\mathcal{A}$ is able to invert the encryption function for NTRU, thus solving the NTRU problem. To bound the probability of $\Pr[T_4]$ we use the algorithm A' which is listed as Algorithm 4.

Algorithm 4 Algorithm A'

Input (h,e).

1. $K \xleftarrow{r} \{0,1\}^N$.
2. Send (h, K, e) to $\mathcal{A}$.
3. Respond to queries, if μ is queried to KDF such that $\mu = c - r * h, r = \text{MGF}(\mu)$ output μ.
4. If $\mathcal{A}$ stops, output fail.

Output μ or fail.

The NTRU-solver A' also runs with $\mathcal{A}$ and its running time is essentially the same as $\mathcal{A}$. It solves the NTRU problem on event T_3 and we can thus use it to bound the probability of T_3. Let $\text{Success}_{\text{NTRU}}(A')$ denote the probability that A' is able to invert the NTRU one-way function.

$$\begin{aligned}
&\Pr[G_3 = 1] - \Pr[G_4 = 1] \\
&= (\Pr[G_3 = 1|T_4] - \Pr[G_4 = 1|T_4])\Pr[T_4] \\
&\quad +(\Pr[G_3 = 1|\neg T_4] - \Pr[G_4 = 1|\neg T_4])\Pr[\neg T_4] \\
&= (\Pr[G_3 = 1|T_4] - \Pr[G_4 = 1|T_4])\Pr[T_4] \\
&\leq \Pr[T_4] = \text{Success}_{\text{NTRU}}(A').
\end{aligned}$$

Putting it all together we have that

$$\begin{aligned}
\mathrm{Adv}(A) &= |\Pr[G_0 = 1] - \frac{1}{2}| \\
&= |\Pr[G_0 = 1] - \Pr[G_1 = 1] + \Pr[G_1 = 1] - \Pr[G_2 = 1] + \Pr[G_2 = 1] \\
&\quad - \Pr[G_3 = 1] + \Pr[G_3 = 1] + \Pr[G_4 = 1] - \Pr[G_4 = 1] - \frac{1}{2}| \\
&\leq |\Pr[G_0 = 1] - \Pr[G_1 = 1]| + |\Pr[G_1 = 1] - \Pr[G_2 = 1]| \\
&\quad + |\Pr[G_2 = 1] - \Pr[G_3 = 1]| + |\Pr[G_3 = 1] - \Pr[G_4 = 1]| \\
&\quad + |\Pr[G_4] - \frac{1}{2}| \\
&\leq \Pr[T_2] + \Pr[T_3] + \Pr[T_4] \\
&\leq Z_{\mathrm{MGF}}\epsilon_{\mathrm{fail}} + |\mathcal{D}_r|^{-1} Z_D + \mathrm{Success}_{\mathrm{NTRU}}(A'),
\end{aligned}$$

which is the success probability the algorithm A' has in solving a random instance of the NTRU problem. We have proved the following theorem.

Theorem 1. *For any random oracle adversary $\mathcal{A}$ against NTRU-KEM there exists an adversary A' against the NTRU one-way function with essentially the same run-time and the advantage*

$$\mathrm{Adv}_{\text{NTRU-KEM}}(\mathcal{A}) \leq |\mathcal{D}_r|^{-1} Z_{\mathrm{KDF}} + \mathrm{Success}_{\mathrm{NTRU}}(A').$$

5 Non-lattice attacks

5.1 Brute force attacks

To produce the secret key $f = 1 + 2F$, the polynomial F is drawn from the polynomial space $\mathcal{D}_f$, consisting of polynomials with d_f ones and $N - d_f$ zeros. The number of possible such keys is then $\#\mathcal{D}_f = \binom{N}{d_f}$. Our guess of f is correct if $f * h = pg$. We do not know the polynomial pg, but we know that it has d_g coefficients equalling p and the rest 0. We may also guess $g \in \mathcal{D}_g$ to find $pg * h^{-1} = f$, where f has small coefficients. As $d_f = d_g$ in the current standardized NTRUEncrypt settings, we get the same number of keys to test as $\#\mathcal{D}_g = \#\mathcal{D}_f$.

Similarly we can guess the blinding polynomial r used for encrypting individual messages. Drawing r from $\mathcal{D}_r$, we check if $e - r * h$ has small coefficients. When padding is used, we guess the r produced by the mask generation function. A correct guess of r will recover the encrypted message. As $\#\mathcal{D}_r = \#\mathcal{D}_f = \#\mathcal{D}_g$ in some parameter sets and $\#\mathcal{D}_r = \#\mathcal{D}_f$ in others, this should be as hard as attacks on the public key.

We found that the number of keys to test can be reduced slightly by a partial precomputation of some of the coefficients of $f * h$ or $g * h^{-1}$. We know that coefficients of pg are 0 and 2 and that coefficients of f are $0, 1$ and

2. If we guess polynomials for f with only $d_f - 1$ ones and $N - d_f + 1$ zeros, the possible number of choices for the last coefficient of f giving the correct sum modulo q for coefficients of pg is reduced. Assuming that coefficients of h are randomly distributed in the interval $[0, q-1]$, the expected number of potential f's we need to test is $2\frac{N}{q}\binom{N}{d_f-1}$ if one coefficient of g is specified. If more coefficients of g are included at this point, the number of keys where $f * h$ must be computed is reduced further. As we know the possible coefficient values needed for a valid g we can produce a sorted list for direct look ups of rotations of h to speed up further.

5.2 Meet-in-the-middle attack

If sufficient storage capacity is available, a meet-in-the-middle attack can find the the secret key in a square root of the time needed for the brute force attack. If we split f in two halves $f = f_a + f_b$ setting different halves of the coefficients of f_a or f_b to zeros, we get that $f * h = f_a * h + f_b * h = pg$. The polynomial pg has coefficients 0 and p, and we may reduce p to 1 by multiplying h by p_q^{-1}.Then $f_a * h$ and $-f_b * h$ should have approximately the same values of their coefficients modulo q.

We let f_a and f_b each be half the length of N and have $\frac{d_f}{2}$ of the coefficients set to 1 in F. We know that f_a and f_b can share the non-zero elements in this way as we can freely choose a rotation of f before splitting into the two parts. We do however need to find the location of the constant term in f by adding $X^i * h$, where i is the number of shifts in the rotation. We choose to work with g_a, g_b and h^{-1} instead, as any rotation can be used directly as all coefficients of g are either 0 or 1.

First we compute all $pg_a * h^{-1}$ and store the coefficients. Then we compute $-pg_b * h^{-1}$ and see if all coefficients are close to the other half and continue until a match has been found. We need to compute and store $\binom{N/2}{d_g/2}$ values of $pg_a * h^{-1}$ in the first part, and compute several $pg_b * h^{-1}$ until a match has been found. The precomputed g_a's are put in different "bins" which are then looked up in the next stage.

To save on space we might store only the leading coefficients in the computed polynomial, or we might want want to store both g_a and $pg_a * h^{-1}$ such that only a subtraction is needed to compute f, as we do not have to compute $pg_a * h^{-1}$ again. If we schedule the computations of $pg_a * h^{-1}$ such that we only change one coefficient of g_a we can reduce the number of computations from addition of $d_g/2$ rotations to one subtraction and one addition.

If we introduce the time costs t_c and t_m for computation and memory access time respectively, the total time t_1 in the first part of the process will be

$$t_1 = \binom{N/2}{d_g/2}(t_c + t_m).$$

If we store k bits in each bin, we need 2^k bins of storage, and for each bit there is a $\frac{2}{q}$ chance that we need to check more bins if the number is close to the one represented by the leading coefficient. Consequently we need to check on average $\frac{2k}{q}$ different bins. For each hit on a bin we need to do a calculation, and as $\binom{N/2}{d_g/2}$ of 2^k bins contain one or more entries, we can calculate the expected number of hits. The maximal time in the second part of the process can be found to be

$$t_2 = \binom{N/2}{d_g/2}\left(t_c + \frac{2k}{q}t_m + \frac{\binom{N/2}{d_g/2}}{2^k}t_c\right).$$

If we increase k we may decrease the expected running time at the cost of increased memory usage. It is claimed by the NTRU team [10] that the increase in memory usage is not exponential in k. It is also clear that increasing q to prevent gap failures and attacks based on them will increase the strength of brute force and meet-in-the-middle attacks slightly.

6 Lattice attacks

Lattice attacks on NTRUEncrypt aim to recover the secret key f from the public key h or the message m from the encrypted message e and h. We view the polynomials as vectors, and use these vectors in construction of a lattice. As lattice reduction finds short vectors in the lattice and the NTRUEncrypt polynomials f, g, r and m produce very short vectors, we hope to find f or m in the shortest vector in the lattice. Polynomials directly related to the NTRU ring $\mathcal{R}_q$ will not be written bold face when considered as vectors. Polynomial and vector notation and operations will be mixed as the natural context will be clear.

In sections 2.3-2.7 we discussed basic lattice reduction methods and various improvements allowing improved performance in reduction quality and speed. Even the best current lattice reduction algorithms are very slow in high lattice dimension, and the security of NTRUEncrypt is based on the hardness of reducing a high dimensional lattice. Any new and improved lattice reduction algorithm increasing the efficiency of attacks on NTRUEncrypt will potentially jeopardize the security of the system.

Due to their iterative nature, LLL-type lattice reduction algorithms do not yield themselves easily to parallel processing. As bases are continuously updated, the need for communication between different processes in distributed methods gives little return on the increased computational power. Even though each reduction method is hard to parallelize, several reductions can be run in parallel with different input, letting us choose the best basis among the different reductions. There is also no known quantum computing algorithm for lattice reduction.

In the rest of this sections we will treat the basic NTRUEncrypt lattice attack, a reduction technique with parts that are NTRUEncrypt specific, an attack that uses vectors that are almost as short as the desired vector and ways to change the NTRUEncrypt lattice to improve efficiency of lattice reduction.

6.1 Attack on the NTRUEncrypt public key

Consider a $2N \times 2N$ matrix composed of four $N \times N$ blocks consisting of the coefficients of h, the large modulus q and a constant σ depending on the parameters of the chosen security level,

$$\left(\begin{array}{c|c} \sigma I_N & 0 \\ \hline \mathrm{cir}(h) & qI_n \end{array}\right) = \left(\begin{array}{cccc|cccc} \sigma & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma & 0 & 0 & \cdots & 0 \\ \hline h_0 & h_{N-1} & \cdots & h_1 & q & 0 & \cdots & 0 \\ h_1 & h_0 & \cdots & h_2 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{N-1} & h_{N-2} & \cdots & h_0 & 0 & 0 & \cdots & q \end{array}\right). \quad (8)$$

The lower left block of coefficients of h is called the circulant matrix and consists of the coefficients of the polynomials $h, hX, hX^2, ..., hX^{N-1}$. We will refer to this block as $\mathrm{cir}(h)$. The block qI_N serves to reduce the coefficients of the h-block modulo q, while the σ in the σI_N-block is a balancing constant for optimization of reduction efficiency. Let the columns of this matrix generate the lattice L. We will then have $\det(L) = \sigma^N q^N$.

From the lattice L we want to recover the private key. We know that the lattice contains the vector $\mathbf{y} = (\sigma f^T \ pg^T)$, which will be know as the target vector. For ease of notation, we present the computation producing $\mathbf{y}$ using the matrix above and matrix-vector multiplication. We obtain the target vector by multiplying the matrix by a $2N$ vector consisting of f and the vector s reducing the last half of the target vector modulo q,

$$\left(\begin{array}{cccc|cccc} \sigma & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma & 0 & 0 & \cdots & 0 \\ \hline h_0 & h_{N-1} & \cdots & h_1 & q & 0 & \cdots & 0 \\ h_1 & h_0 & \cdots & h_2 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{N-1} & h_{N-2} & \cdots & h_0 & 0 & 0 & \cdots & q \end{array}\right) \left(\begin{array}{c} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \\ s_0 \\ s_1 \\ \vdots \\ s_{N-1} \end{array}\right) = \left(\begin{array}{c} \sigma f_0 \\ \sigma f_1 \\ \vdots \\ \sigma f_{N-1} \\ pg_0 \\ pg_1 \\ \vdots \\ pg_{N-1} \end{array}\right).$$

The first half of the product is easily seen to be σf, as the upper left block of the matrix only has diagonal elements. For the second half, first recall

the definition of the public key

$$h = f^{-1} * pg$$

which by a multiplication by f becomes

$$f * h \equiv f * f^{-1} * pg \equiv pg \equiv 2g \pmod{q}. \tag{9}$$

In the practical implementations of lattice reduction we can't work modulo q, so s and the qI_N block is needed to do the reductions.

For the details of this computation, consider first$f * f^{-1} = 1$, where all coefficients but the constant term cancels out. When splitting into elements we get

$$\sum_{i=0}^{N-1} f_i f_{-i+j}^{-1} = \begin{cases} 1 \bmod q & \text{if } j = 0 \\ 0 \bmod q & \text{if } j \neq 0 \end{cases} \tag{10}$$

as expected when multiplying a polynomial with its inverse. The reversed order of coefficients in the rows of $\mathrm{cir}(h)$ gives a straightforward computation of the star multiplication as the indices of f and h go in opposite directions as in Definition 3. To prove (9) we let the shifts of $\mathrm{cir}(h)$ be denoted by the term z added to the index, where z equals the row number in the $\mathrm{cir}(h)$ block. With qs for reduction, the substitution $i+j = -k$ and indices modulo N we get

$$\begin{aligned}
\sum_{i=0}^{N-1} f_i h_{N-i+z} - qs &= \sum_{i=0}^{N-1} f_i \sum_{j=0}^{N-1} f_j^{-1} pg_{N-j-i+z} - qs \\
&= p \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f_i f_j^{-1} g_{N-j-i+z} - qs \\
&= p \sum_{k=0}^{N-1} \sum_{i=0}^{N-1} f_i f_{-i-k}^{-1} g_{N+k+z} - qs \\
&= p \sum_{k=0}^{N-1} g_{N+k+z} \sum_{j=0}^{N-1} f_i f_{-i-k}^{-1} - qs \\
&= pg_z - qs
\end{aligned}$$

using (10) in the last equality to remove all terms with non-zero k. Then clearly the second half of the solution vector is pg with $\mathbf{y}_{N+z} = pg_z$.

In lattice reduction we add or subtract multiples of one vector to another. The other lattice operation of sorting the vectors by their length helps us produce and find the shortest vector in the lattice. We can obtain an individual lattice vector as above by multiplying the matrix by an integral vector.

As lattice reduction should recover the shortest lattice vector we want our target vector $\mathbf{y}$ to be the shortest non-zero vector. To ensure this we

need to find a suitable value for σ, and for this we must know what vector lengths to expect. It is known that in a lattice of sufficiently large dimension n, the Hermite constant satisfies

$$\frac{1}{2\pi e} \leq \frac{\gamma_n}{n} \leq \frac{1.744}{2\pi e}$$

giving from the definition of the Hermite constant in (3) a bound for the shortest vector

$$d(L)^{\frac{1}{n}}\sqrt{\frac{n}{2\pi e}} \leq \lambda(L) \leq d(L)^{\frac{1}{n}}\sqrt{\frac{1.744n}{2\pi e}}.$$

Inserting our lattice dimension and determinant we get the expected smallest vector length

$$E_L = (\sigma^N q^N)^{\frac{1}{2N}}\sqrt{\frac{2N}{2\pi e}} = \sqrt{\frac{N\sigma q}{\pi e}}.$$

As E_L is the expected length of the shortest vector in a random lattice, the attacker seeks to find a value of σ maximizing the ratio $\frac{E_L}{\|\mathbf{y}\|}$ making the gap between our target vector and expected length of the next shortest vector as large as possible. This will give the highest probability that a vector of length $\mathbf{y}$ found by lattice reduction is the target vector. It will also make lattice reduction easier, as finding the finding the shortest vector in a lattice resembling a random lattice is hard [6]. Squaring the ratio $\frac{E_L}{\|\mathbf{y}\|}$ we can easily find the optimal value of σ by maximizing

$$\frac{\sigma}{\sigma^2\|(\sigma f^T \ 2g^T)\|^2}.$$

The lengths $\|f\|$ and $\|g\|$ are public quantities, and with the current standardized key generation parameters [29], $\|f\| \approx \|2g\|$. As $\|(\sigma f^T \ 2g^T)\|^2 = \|\sigma f\|^2 + \|2g\|^2 \approx \sigma^2\|f\|^2 + \|f\|^2$, the maximal value is found at $\sigma \approx 1$. In lattice reduction σ serves as a balancing constant giving equal weight to elements in the vector and thereby increasing the effeciency of reduction. We choose $\sigma = 1$ as it is more advantageous to keep L as an integral lattice than to use a floating point value close to 1.

Let the ratio $c_h = \frac{\|\mathbf{y}\|}{E_L}$ be a measure of how far the length of the shortest vector departs from the shortest vector of a random lattice. As c_h decreases, finding $\mathbf{y}$ will be easier as it gets considerably shorter than the expected shortest vector. The value of c_h can be deduced from public quantities

$$c_h = \sqrt{\frac{\pi e(\|\sigma f\|^2 + \|2g\|^2)}{N\sigma q}}.$$

The constant c_h is called the lattice constant, and is sometimes defined with a different norm on the target vector or with the dimension term N removed.

We have chosen this form of c_h as it involves the most natural norm in lattice reduction and gives a simpler analysis of lattices of reduced dimension and varying N.

Another constant impacting the hardness of lattice reduction is the ratio $\frac{N}{q}$, where lower values give easier recovery of the target vector. As a low q leads to more decryption failures, a suitable value of q maximizing lattice strength and minimizing the effectiveness of decryption failure attacks must be chosen.

6.2 Attack on individual NTRUEncrypt message

The attack on individual NTRUEncrypt messages closely resembles the attack on the public key. The lattice used is very similar to the public key attack, but the dimension is increased by one to $2N + 1$. Another vector of length $2N + 1$ consisting of N 0-entries, the N coefficients of the secret message e and finally a 1 is added. The other vectors are appended by a 0 to obtain length $2N + 1$.

The lattice is then generated by the vectors of the matrix

$$\left(\begin{array}{cccc|cccc|c} \sigma & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \sigma & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \sigma & 0 & 0 & \cdots & 0 & 0 \\ \hline h_0 & h_{N-1} & \cdots & h_1 & q & 0 & \cdots & 0 & e_0 \\ h_1 & h_0 & \cdots & h_2 & 0 & q & \cdots & 0 & e_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ h_{N-1} & h_{N-2} & \cdots & h_0 & 0 & 0 & \cdots & q & e_{N-1} \\ \hline 0 & 0 & \ldots & 0 & 0 & 0 & \cdots & 0 & 1 \end{array}\right).$$

We will once more be looking for a short vector. With $s \in \mathcal{R}$, we are looking for the short vector $(r^T \ -m^T \ -1) = (r^T \ (r*h-e+qs)^T \ -1)$. This can be done by selecting multiples of each vector similarly to multiplying the matrix above by $(r^T \ s^T \ -1)$. Determining the balancing constant will similarly be done by setting $\sigma = \frac{\|r\|}{\|m\|}$. The value of $\|m\|$ is not a public quantity, but we can use the expected value $N/2$. Then we can find the value

$$c_m = \sqrt{\frac{2\pi e\|(r^T \ m^T)\|}{Nq}}$$

as we did with c_h. It is desireable to make attacks on h and m equally difficult with $c_m \approx c_h$, $\|(r, m)\| \approx \|(f, 2g)\|$.

6.3 Attack by spurious keys

Shortly after the first presentation of NTRU, Coppersmith and Shamir [4] showed how to make use of lattice vectors that are almost as short as the target vector. The idea is to obtain partial information from several short vectors and combine this in an attack on individual messages.

We define the *centered L_2 norm* on $\mathcal{R}$ as the length of the projection orthogonal to (1 1 ... 1)

$$|f|_\perp = \left(\sum_{i=1}^{N} (f_i - \bar{f})^2 \right)^{\frac{1}{2}}, \quad \text{with } \bar{f} = \frac{1}{N} \sum_{i=1}^{N} f_i.$$

For polynomials a, b we have that $|a * b|_\perp \approx |a|_\perp |b|_\perp$.

First we try to find f' and g' not much longer than f and g such that $pg = f * h \pmod q$. We then look at the polynomial a from (7), which has a centered L_2 norm

$$|a|_\perp^2 = |f * e|_\perp^2 = |t\bar{1} + pg * r + f * m|_\perp^2 \approx p^2 |g|_\perp^2 |r|_\perp^2 + |f|_\perp^2 |m|_\perp^2$$

where $t\bar{1}$ is a polynomial with all coefficients equalling t serving to reduce the coefficients into the right interval. We require that

$$|a'|_\perp^2 = |f' * e|_\perp^2 = |t\bar{1} + pg' * r + f' * m|_\perp^2 \approx p^2 |g'|_\perp^2 |r|_\perp^2 + |f'|_\perp^2 |m|_\perp^2$$

is smaller or not much larger than $|a|_\perp^2$. We also assume that

$$b'_k = \sum_i m_i f'_{k-i} \pmod p$$

which gives a linear relation disclosing message bits. The success of the method depends on the coefficients of b' being in the correct interval, the length of f' and g' and the number of such alternative vectors found. If we find two such pairs, their length may be up to 2.5 times longer than the target, while finding several pairs allows us to use vectors four times as long as the target vector. Longer vectors lead to more errors in the linear relation, and techniques from error correcting codes must be used to find the correct decryption.

Hoffstein, Pipher and Silverman [6] of NTRU claim based on experimental results that finding vectors almost as short as f and g is very unlikely as they hardly saw them occur. It will often be easier and more efficient to find the actual target vector. In addition, using the NTRUEncrypt padding scheme, the linear relations needed for this method are removed.

6.4 Non-deterministic parallel reduction

In an attack by Seidel, Socek and Sramka, the symmetry of automorphism groups of the NTRUEncrypt lattice is exploited [22]. The cyclic structure of

$\mathrm{cir}(h)$ enables the possibility to shift coefficients of single vectors to advance reduction when BKZ type reductions can't find shorter vectors. We seek to find lattice bases with successively shorter vectors until, if successful, the basis vector with the same length as our target vector is found. The core components of the method are random permutations of vectors and BKZ reduction in parallel and shifts of vectors.

By a *rotation* we mean a cyclic shift of coefficients in a vector or polynomial. Let $\mathrm{rotate}_c(\mathbf{v})$ be a cyclic shift of the vector $\mathbf{v}$ by c positions. Similarly an NTRUEncrypt polynomial corresponding to $\mathbf{v}$ can be shifted by multiplication by X^c. Due to the block structure of the NTRUEncrypt lattice, we need to treat the two halves corresponding to different blocks separately. For a vector $\mathbf{w} = (\mathbf{u}\ \mathbf{v})$ we introduce *birotation* such that $\mathrm{birotate}_c(\mathbf{w}) = (\mathrm{rotate}_c(\mathbf{u})\ \ \mathrm{rotate}_c(\mathbf{v}))$.

Let L be an NTRUEncrypt lattice and B be the matrix given in (8). Any lattice vector can be written as a linear combination of vectors from B, $\mathbf{w} = B\mathbf{x}$ where $\mathbf{x}$ is an integral vector $\mathbf{x} = (x_1, ..., x_{2N})$. For a birotation we have

$$\mathrm{birotate}_c(\mathbf{w}) = B\,\mathrm{birotate}_c(\mathbf{x}),$$

such that $\mathrm{birotate}_c(\mathbf{w}) \in L$.

Let P_c be an $N \times N$ permutation matrix shifting each vector c positions. We can then construct the block matrix

$$P = \begin{pmatrix} P_c & 0 \\ 0 & P_c \end{pmatrix}$$

with P_c and two $N \times N$ zero matrices. We have the equality

$$\mathrm{birotate}_c(\mathbf{w}) = P\mathbf{w}.$$

Using the identity above and multiplying by P we get

$$P\mathbf{w} = PB\mathbf{x} = PBP^{-1}P\mathbf{x}.$$

From the construction of $\mathrm{cir}(h)$ (8) we get that

$$P_c\,\mathrm{cir}(h) = \mathrm{cir}(h)P_c$$

as shifting the rows or the columns of $\mathrm{cir}(h)$ c positions produces the same result. Consequently

$$\begin{aligned} PBP^{-1} &= \begin{pmatrix} P_c & 0 \\ 0 & P_c \end{pmatrix} \begin{pmatrix} I & 0 \\ \mathrm{cir}(h) & qI \end{pmatrix} \begin{pmatrix} P_c^{-1} & 0 \\ 0 & P_c^{-1} \end{pmatrix} \\ &= \begin{pmatrix} P_c & 0 \\ P_c\,\mathrm{cir}(h) & qP_c \end{pmatrix} \begin{pmatrix} P_c^{-1} & 0 \\ 0 & P_c^{-1} \end{pmatrix} \\ &= \begin{pmatrix} P_cP_c^{-1} & 0 \\ P_c\,\mathrm{cir}(h)P_c^{-1} & qP_cP_c^{-1} \end{pmatrix} = \begin{pmatrix} I & 0 \\ \mathrm{cir}(h) & qI \end{pmatrix} = B \end{aligned}$$

so clearly

$$\text{birotate}_c(\mathbf{w}) = B\,\text{birotate}_c(\mathbf{x}),$$

and we conclude that $\mathbf{w} \in L$ if and only if $\text{birotate}_c(\mathbf{w}) \in L$.

The birotation method works by replacing a long vector in the current lattice basis $B = (\mathbf{b}_1, ..., \mathbf{b}_{2N})$ with the birotation of a short vector in the same lattice. Let $m, n \in \mathbf{Z}$ such that $1 \leq m < n \leq 2N$ and $\|\mathbf{b}_m\| < \|\mathbf{b}_n\|$ be vectors in the lattice and set

$$\mathbf{b}_n \leftarrow \text{birotate}_c(\mathbf{b}_m), \quad c = 1, ..., N-1.$$

If the vectors in this new basis B' are linearly independent, we have obtained a basis with smaller weight, $\text{wt}(B) = \|\mathbf{b}_1\|\|\mathbf{b}_2\| \cdots \|\mathbf{b}_{2N}\|$. To check for linear independence we test if $\det(B) = \det(B')$, if they are not equal we increment c to test another birotation. If the birotation procedure succeeds we proceed with the new lattice basis.

Permutation of basis vectors is another main part of this method. By rearranging the vectors of a reduced basis, we hope to achieve further reductions. Let $\alpha(B, k)$ be a permutation of B producing a new lattice basis differing from B in exactly k vectors. As BKZ reduction is sensitive to the order of vectors in the basis, we hope to achieve further reduction by changing the ordering of vectors in the reduced basis. Permutations and subsequent BKZ reduction is carried out in M parallel processes and the basis with the shortest vector is chosen for further reduction.

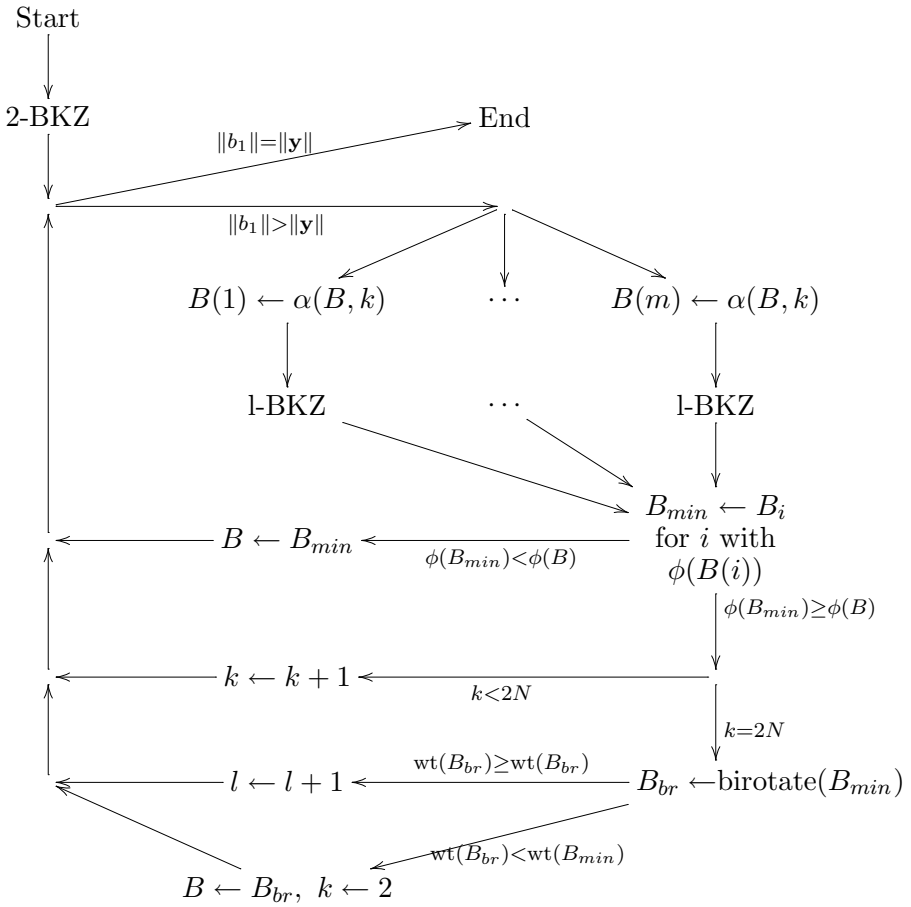
The natural goal of the algorithm is to recover a short vector of the same length as the target vector $\mathbf{y}$. We define the function $\phi(B) = \frac{\|\mathbf{b}_1\|}{\|\mathbf{y}\|}$ as the ratio of the length of the shortest vector in B to the length of the target vector. With a basis B_i and a new and potentially shorter basis B_{i+1}, we will change to the shorter basis if $\phi(B_{i+1}) < \phi(B_i)$. If the basis is not improved we will increase the blocksize of BKZ reduction or the number of vectors to permute and perform another permutation and reduction.

The algorithm is shown schematically as Algorithm 5. Initially we have the lattice basis $B, k = 2$ and $l = 2$. Seidel et. al. [22] does not specify how to choose m, n for birotations, but a natural choice would be to use a low index for m and a high for n as this would produce the greatest reduction of $\mathbf{b}_n$.

If the algorithm is successful in finding a vector of the same length as the target vector, the output should be the target vector or one of its rotations. There are now only N rotations to check, and if the key generation procedure setting $f = 1 + pF$ is used with $p = 2$, we can find the correct rotation by setting the odd coefficient in the first position.

Seidel, Socek and Sramka do not give an estimate of the performance of the algorithm. Socek does however treat an earlier version of this attack in his master thesis [26], finding that it in many cases outperforms BKZ reduction with increasing blocksize, but that its behaviour sometimes is

Algorithm 5 Parallel symmetric non-deterministic attack



worse. Due to the non-deterministic permutations, the effectiveness even on repetitions of a single instance of the algorithm is unpredictable.

Our general experience with lattice reduction shows that with a l-BKZ reduced lattice basis, finding a l+1-BKZ reduced lattice basis is generally much faster than finding the l+1-BKZ basis directly. Permutation of vectors in a reduced lattice basis leads to further reduction with a high time cost. We can expect the parallelized part of the algorithm to produce good reductions of the lattice. Meskanen [17] and we have found that reducing the NTRUEncrypt lattice and the same lattice with its vectors permuted yields very different reduction times, and we may expect that some of the permutations in parallel may be much slower than the others. In this case other processors stay idle, and some mechanisms should be implemented to select the basis with the shortest vector, even if it is still being reduced, or let idle

processors work on lattices from slower processes with a new permutation.

6.5 Adjustments to the NTRUEncrypt lattice

Some properties of the NTRUEncrypt public key and the corresponding lattice offer possibilities for improvements to the standard lattice based attacks on NTRUEncrypt. Using these tricks we can alter lattices to achieve more efficient lattice reduction and reduce the dimension of the lattice. Experiments show that the time needed to discover the secret key can be reduced significantly.

6.5.1 Zero forcing

The polynomials f and g created in NTRUEncrypt key generation have a large portion of zero coefficients, and this is exploited in an attack by May [15] and an improved attack by May and Silverman [16]. With the NTRUEncrypt lattice

$$\begin{pmatrix} I & 0 \\ \text{cir}(h) & qI \end{pmatrix} \begin{pmatrix} f \\ s \end{pmatrix} = \begin{pmatrix} f \\ 2g \end{pmatrix}$$

we see that to obtain the target vector we have to sum vectors corresponding to the polynomial $(f^T \ s^T)$. With the current NTRUEncrypt parameters, the private key f has d_f or $d_f - 1$ twos, the constant term one or three and the rest of the coefficients zero. Vectors in the lattice that correspond to a zero in f will not be included in the set of vectors that sum to the target vector. As lattice reduction has an emphasis on finding the short vectors of a lattice and longer vectors will be handled less, we can give some vectors less attention in reduction by making them longer. The idea is that this will reduce the practical dimension of the lattice and speed up the reduction.

We do not know the location of any zeros in f, as they should be randomly distributed except for the non-zero constant term. However, due to the rotational symmetry of the NTRUEncrypt public key, we have that $h*X^i$ is a right rotation of the public key polynomial h, all coefficients shifted i positions to the right. The same applies to products of polynomials. For the public key

$$f * h = pg$$

we have

$$(X^i * f) * h = X^i * (f * h) = X^i * (pg)$$

which means that any rotation of the public key can be used. Consequently we do not have to guess the exact location of zeros in f or g, as long as the zeros create a pattern that exists somewhere in the desired polynomial.

May's first idea was to look for a "zero run" in f or g, a number of consecutive zeros appearing somewhere in the polynomial. By multiplying

columns or rows involving the cir(h)-block by a constant θ before reduction we may guess the location of zeros in f or g respectively. By multiplying a column by θ, the length is increased and linear combinations with this vector will most likely be longer than others. By multiplying rows by θ we guess the locations of zeros in g as nonzero elements in the last half of the target vector now will be 2θ, contributing to longer vectors than the original value of 2. It is also possible to alter both rows and columns, but in this case we lose the rotational symmetry and the exact location of the zeros must be guessed.

Important adjustments to May's method have been suggested in [15, 17]. Instead of guessing a run of zeros in f or g, a pattern of zeros may be guessed instead as the zeros do not have to be consecutive. Instead of multiplying a column assumed to correspond to a zero in f by a constant we may instead remove it. This leaves a zero row in the lattice among the first N vectors, and this row may also be removed, reducing the dimension of the lattice by the number of zeros in the pattern.

As the dimension is reduced we can in general expect that the time needed for reduction is also shortened, although the target vector will not be found at all if the selected pattern is not included in the intended part of the target vector. With longer patterns the chance of guessing the correct pattern decreases. There is no known way to run the LLL algorithm itself in parallel, but this situation lets us run the same public key with different patterns on several processors.

With a pattern of length l and an NTRUEncrypt key of length N created by polynomials with d ones, we can calculate the probability of guessing the correct pattern. If we have chosen our pattern correctly, all d nonzero elements are among the remaining coefficients of the polynomial and the probability for a specific location of the pattern is

$$\Pr[a_{i_1} = a_{i_2} = \cdots = a_{i_l} = 0] = \frac{\binom{N-r}{d}}{\binom{N}{d}} = \prod_{i=0}^{l-1} \left(1 - \frac{d}{N-i}\right).$$

Due to the rotational symmetry the pattern may be located anywhere in the polynomial, resulting in the improved probability

$$\Pr[a_{i_1+k} = \cdots = a_{i_l+k} = 0] = 1 - \left(1 - \prod_{i=0}^{l-1} \left(1 - \frac{d}{N-i}\right)\right)^N$$

where the indices are taken modulo N.

We say that the attacker "wins" if the chosen pattern corresponds to zeros in a rotation of the polynomial. Theoretically nothing is gained if the same pattern appears in different rotations of the polynomial, but practically one rotation might be easier to find by LLL reduction than the other. As the time needed to break an NTRUEncrypt key is found experimentally to

be exponential in the dimension of the lattice of the form $\log t = \alpha N + \beta$ the gain G for a single pattern containing of l zeros is

$$G = 1 - \left(1 - \prod_{i=0}^{l-1}\left(1 - \frac{d}{N-i}\right)\right) * 10^{\alpha l}.$$

If we parallelize the process nothing is gained if more than one pattern wins. The gain in parallelization is therefore not linear in the number of processors, and we should adjust the number of zeros in our pattern to the number of processors available. Let $T(l)$ be the time to run a pattern with l zeros or l vectors removed and $P(l)$ be the probability of guessing the valid pattern. With C processors, the expected running time is $E_t(C, l)$ given by

$$\begin{aligned} E_t(C,l) &= \frac{T(l)}{\Pr[\text{at least one pattern wins}]} \\ &= \frac{T(l)}{1 - \Pr[\text{all patterns wrong}]} \\ &= \frac{T(l)}{1-(1-P(l))^C} = \frac{10^{\alpha(N-l)+\beta}}{1-\left(1-\prod_{i=0}^{l-1}\left(1-\frac{d}{N-i}\right)\right)^{NK}}. \end{aligned}$$

For very small $P(l)$ we have $E_t(C, l) \approx \frac{T(l)}{CP(l)}$. With a very large number of processors and constant $P(l)$ we get $\lim_{C\to\infty} E_t(P, l) = T(l)$, but this requires that $P(l)$ is smaller than $1/C$, which may not be the optimal value for $P(l)$.

May and Silverman [16] computes the gain of parallelizing to allow more processor work on patterns with more zeros than would be optimal with just one processor. Examining their results we find that the gain relative to a single processor is slightly better than $\sqrt{C}$.

6.5.2 Tricks exploiting p and optimizing lattice constants

The standard NTRUEncrypt lattice as presented in section 6.1 turns out not to be the optimal for lattice reduction. Several adjustments can be made to improve the performance of lattice reduction to obtain the private key or message polynomial. Starting with the lattice suggested by Coppersmith and Shamir [4] and the vectors involved in the solution we have

$$\begin{pmatrix} I & 0 \\ \text{cir}(h) & qI \end{pmatrix} \begin{pmatrix} f \\ s \end{pmatrix} = \begin{pmatrix} f \\ 2g \end{pmatrix}$$

if we use the current NTRUEncrypt key generation $h = f^{-1} * pg$ with $p = 2$ and set the value $\alpha = 1$.

With the current parameter settings, polynomials with binary coefficients are used to create f and g. The private key f is produced by first

creating a polynomial F with d_f ones and the other coefficients zero, and then taking $f = 1 + pF$. As $h = f^{-1} * pg$, $f * h = pg$. With the current parameter settings, it turns out that the length of the target vector is approximately $\|\mathbf{y}\| \approx \sqrt{8d}$, with $d = d_f = d_g$. The actual length is either $\sqrt{8d+1}$ or $\sqrt{8(d-1)+3^2}$ depending on the constant term of f which is either 1 or 3. For the NTRUEncrypt parameter set `ees251ep4` ($N = 251$, $d = 72$, $p = 2$ and $q = 239$), this leads to a value of $c_h \approx 0.286$. For the attack on an NTRUEncrypt message, we get $c_m \approx 0.267$.

We found an adjustment to the NTRUEncrypt lattice that exploits the structure of f by noting that all coefficients except the first are either 0 or 2. For the corresponding columns in the lattice, we multiply all elements by 2 and reduce $\pmod q$. The value of c_h is improved by a factor close to $\sqrt{2}$ to $c_h \approx 0.202$, which should make it easier for lattice reduction methods to discover the target vector. We also suspect that this has other implications on the lattice reduction. Experiments show an improvement in the time and block size needed to find the target vector. Due to the structure of f, adding an odd number of some vectors can not produce the target vector, and by eliminating the possibility of adding odd numbers of these vectors, we reduce the number of unhelpful operations in the reduction. Our first idea in this direction was to allow only addition and subtraction of vectors in LLL for $|\mu_{i,j}| \geq 1$ in contrast to the regular $|\mu_{i,j}| \geq 0.5$, but this turns out to be too rare in the NTRUEncrypt lattice. Multiplying the vectors by 2 turns out to be the better choice.

Meskanen also discovers this same method in a slightly different way in his thesis [17]. He introduces a number of tricks to improve the value of c_h and reduce the time to find the target vector. By constructing adjusted lattices, the target vector can be found more efficiently, but in many cases the possibilities for combining with other improvements to lattice reduction are reduced.

In his first trick, Meskanen adds a lattice vector that is always close to f and $2g$. Let this vector be $a\bar{1} = (a \;\; a \;\; \ldots \;\; a)$. To minimize the distance between $(f^T \;\; 2g^T)$ and $(a\bar{1} \;\; a\bar{1})$ an a is selected such that $\sqrt{\|f-a\|^2 + \|2g-a\|^2}$ is minimized. This leads to $a \approx \frac{2d}{N} \approx \frac{1}{2}$. As an integral lattice is preferred, we multiply all elements of the lattice by 2 to obtain

$$\begin{pmatrix} 2I & 0 & -\bar{1} \\ \mathrm{cir}(2h) & 2qI & -\bar{1} \end{pmatrix} \begin{pmatrix} f \\ s \\ 1 \end{pmatrix} = \begin{pmatrix} 2f - \bar{1} \\ 2g - \bar{1} \end{pmatrix}$$

which gives $c_h \approx 0.243$. Adding the vector $(\bar{1} \;\; \bar{1})$ leads to linear dependent vectors and the new vector is now the shortest vector in the lattice. We do not deal with these problems as subsequent tricks will give a solution.

This first trick can also be applied to the NTRUEncrypt message with

the lattice

$$\begin{pmatrix} 2I & 0 & \bar{1} \\ \mathrm{cir}(2h) & 2qI & 2e+\bar{1} \\ \bar{0}^T & \bar{0}^T & 1 \end{pmatrix} \begin{pmatrix} f \\ s \\ -1 \end{pmatrix} = \begin{pmatrix} 2f-\bar{1} \\ 2g-\bar{1} \end{pmatrix}$$

which has a solution where all coefficients are ± 1 and hence has a length of $\sqrt{2N+1}$ and has $c_m \approx 0.134$.

The second trick of Meskanen takes advantage of the same observation as ours above. Noting that the first of his first trick leads to

$$f * 2h + 2qs = 4g$$

we expand f to obtain

$$(1+2F) * 2h + 2qs = 2F * 2h + 2qs + 2h = F * 4h + 2qs + 2h = 4g.$$

We then move $2h$ which will correspond to the constant term into the second part of the vector we added above.

$$\begin{pmatrix} 4I & 0 & -\bar{1} \\ \mathrm{cir}(4h) & 2qI & 2h-\bar{1} \end{pmatrix} \begin{pmatrix} F \\ s \\ 1 \end{pmatrix} = \begin{pmatrix} 4f-\bar{1} \\ 4g-\bar{1} \end{pmatrix}.$$

In addition to this we can exploit the parity of s by setting $s = 2s_1 - s_2$. First we go back a few steps and consider

$$\begin{pmatrix} 2I & 0 & \bar{0} \\ \mathrm{cir}(2h) & 2qI & h \end{pmatrix}.$$

Noting that as our target vector is $(2F\ \ 2g)$, all elements of the target vector are 0 mod 2. In (6.5.2) all columns except the last have coefficients that are 0 mod 2. As the last column is included in the solution, this column must also have coefficients that are 0 mod 2. This is obtained by subtracting q from all elements as needed, written as the operation of subtracting qs_2 from h. All elements are then 0 mod 2 and we may move the coefficient 2 from s_1 into the lattice to obtain the following lattice and calculation

$$\begin{aligned} &\begin{pmatrix} 4I & 0 & -\bar{1} \\ \mathrm{cir}(4h) & 2qI & 2h-\bar{1} \end{pmatrix} \begin{pmatrix} F \\ 2s_1 - s_2 \\ 1 \end{pmatrix} \\ =&\begin{pmatrix} 4I & 0 & -\bar{1} \\ \mathrm{cir}(4h) & 4qI & 2h-2qs_2-\bar{1} \end{pmatrix} \begin{pmatrix} F \\ s_1 \\ 1 \end{pmatrix} \\ =&\begin{pmatrix} 4f-\bar{1} \\ 4g-\bar{1} \end{pmatrix}. \end{aligned}$$

With this lattice we get $c_h \approx 0.121$. The last vector is not the shortest anymore, and even though the lattice is not linearly independent, we can make it linearly dependent by removing the first column if we assume that the constant term $f_0 = 1$, otherwise add the first and last column as $f_0 = 3$.

In a third trick, further balancing is achieved by knowledge of the NTRUEncrypt parameter d. As we know that the last column must be part of the solution we add d to all elements of this vector and 1 to all elements of the first N vectors. Let $\check{1}$ be a block with all elements 1, we then get the lattice

$$\begin{pmatrix} 4I + \check{1} & 0 & -\bar{1} - \bar{d} \\ \mathrm{cir}(4h) + \check{1} & 4qI & 2h - 2qs_2 - \bar{1} - \bar{d} \end{pmatrix} \begin{pmatrix} F \\ s_1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4f - \bar{1} \\ 4g - \bar{1} \end{pmatrix}.$$

where we remove the first column to make the lattice linearly independent as before. This lattice yields good performance results in our and Meskanen's experiments [17]. This is also expected as c_h decreases slightly. The length of the target vector is unchanged, and the expected length increases, but this increase is dependent on h and we can therefore not give a general calculation of c_h.

By removing the first vector from the lattice, we assume that the constant term of f is 1. For the NTRUEncrypt parameters $N = 251, d = 72$ this is true with a probability of about 71%. If the constant term is 3, reduction of this lattice will not be successful in finding the target vector. In this case we add the first and last column together and should then find the target vector.

When applying our own adjustment or the second or third trick of Meskanen, we lose the rotational symmetry of the NTRUEncrypt lattice. Zero forcing and pattern guessing techniques of May and Silverman [16] will also lose some of their effectiveness due to this. Guessing a run of zeros or a pattern of zeroes is still possible, but the exact possition of the pattern has to be guessed, greatly reducing the effectiveness of the attack. We may however guess the location of another 1 in F by adding the last vector above to a vector other than the first. This enables us to test both values of the constant term of F at once by guessing the location of another 1 in F, but the chance of guessing correctly is only 29%. As we seek to locate any 1 in F, we can guess more locations hoping that exactly one is correct. By adding the last vector to two, three or four other vectors the probability of hitting exactly one correct vector is above 40%. There does however seem to be practical problems probably due to the balancing weights cancelling each other out during lattice reduction.

The birotation method proposed by Seidel, Socek and Sramka [22] can also be used with caution for these lattices. Birotation of a vector normally yields another vector in the lattice. Birotating a vector that involves the vector corresponding to the constant term will not be helpful as this leads to multiples of vectors that are not part of the solution. Elements in the

top left block can be used to determine if this is the case. For the message attack lattice, we do not include the last element in the rotation. The last element will tell us if a birotation is possible, as it can only be performed when this element is 0 meaning that we only rotate a sum of h-vectors.

Another idea we tried is based a combination of the tricks by Meskanen [17] and a comment by Coppersmith and Shamir [4]. If we multiply all the elements of $\mathrm{cir}(h)$ by p^{-1} modulo q denoted as p_q^{-1}, we get $(f\ g)$ and not $(f\ 2g)$, a shorter target vector. If we guess the coefficient of the constant term we may also substitute F for f to get the minimal target vector. This causes an increase of c_h, as the decrease in the expected shortest vector is greater than the decrease of the target vector. If we use the lattice of trick three substituting $p_q^{-1}h$ for h, we get that $4p_q^{-1}h = 2h$ as $2p_q^{-1} = pp_q^{-1} = 1$. Hence the reduction block must now be set to $2qI$, and the parity considerations of s are obsolete. Experiments show that this lattice is comparable to but does not appear to be better than trick three of Meskanen. To retain the size and parity of the reductions modulo q we may multiply h and q a constant other than 2 and find another constant a in the vector $(a\bar{1}\ \cdots\ a\bar{1})$ to add in, but other natural choices increases the length of the target vector even more.

6.5.3 An attack on the inverse of the public key

Using the tricks above, we lose the rotational symmetries because of the way f is created, forcing us to locate and guess the constant term. We may however search for g instead by replacing h by its inverse h^{-1}. We get $2g * h^{-1} = 1 + 2F$. All the coefficients of $2g$ are either 0 or 2. We can then construct the lattice as before

$$\begin{pmatrix} I & 0 \\ \mathrm{cir}(h^{-1}) & qI \end{pmatrix} \begin{pmatrix} 2g \\ t \end{pmatrix} = \begin{pmatrix} 2g \\ f \end{pmatrix}$$

and move the factor 2 into the lattice as before to obtain

$$\begin{pmatrix} 2I & 0 \\ \mathrm{cir}(2h^{-1}) & qI \end{pmatrix} \begin{pmatrix} g \\ t \end{pmatrix} = \begin{pmatrix} 2g \\ f \end{pmatrix}.$$

This lattice can further be enhanced by tricks similar to the ones above. We might add another vector of ones as before, resulting in a linearly dependent system of vectors. This can be avoided by a similar consideration of the parity of t as in trick two above, but using this method we lose the rotational symmetry by choosing the location of the constant term in f.

We may however use a method similar to trick three above that retains the symmetry. When computing $2g * h^{-1} = f$, a set number of reductions modulo q has been performed. When adding together d rotations h^{-1}, we know that the sum of all the elements is $2g(1)h^{-1}(1)$. The sum of the

elements of f is $f(1)$, and hence the number of reductions is

$$t(1) = \frac{2g(1)h^{-1}(1) - f(1)}{q} \tag{11}$$

which we use to balance the lattice above. We increase the dimension of the lattice by one to obtain

$$\begin{pmatrix} 4I & 0 & -\bar{1} \\ \text{cir}(4h^{-1}) & 2qI & -\bar{1} \\ \bar{0}^T & \bar{m}^T & -mt(1) \end{pmatrix} \begin{pmatrix} g \\ t \\ -1 \end{pmatrix} = \begin{pmatrix} 4g - \bar{1} \\ 4f - \bar{1} \\ 0 \end{pmatrix}. \tag{12}$$

We may also balance by adding the scalar m to the end of the first N vectors and set the last element in the last vector to $-md_g$. We will then obtain

$$\begin{pmatrix} 4I & 0 & -\bar{1} \\ \text{cir}(4h^{-1}) & 2qI & -\bar{1} \\ \bar{m}^T & \bar{0}^T & -md_g \end{pmatrix} \begin{pmatrix} g \\ t \\ -1 \end{pmatrix} = \begin{pmatrix} 4g - \bar{1} \\ 4f - \bar{1} \\ 0 \end{pmatrix} \tag{13}$$

which is also fairly similar to the construction in (11) in terms of effectiveness. There are also differences between the two lattices when it comes to application of zero forcing methods. In (13) we can reduce the dimension using zero forcing, as we have removed vectors that are not part of the solution, the last element still sums to 0. The value of (11) used in (12) will not be correct if the dimension is reduced, as a part of h^{-1} is removed from every vector in $\text{cir}(h^{-1})$. May's original zero forcing idea where some elements on the diagonal in the upper left block is increased may however still be used.

If a specific pattern of 0's and 1's is sought, the vectors corresponding to the selected 1's can be added to the last vector in the lattice. In the case of (13), this enables us to reduce the dimension the lattice by the total length of the pattern including the ones.

If our guess is correct we get shortened rotations of f and g. We will need to append the selected pattern or number of zeros to g and compute $2g * h^{-1} = f$ as some coefficients of f are missing after reduction. When f is retrieved, we may find the position of the constant term as this is the only coefficient equal to 1 and use this to rotate both vectors to put the coefficients in the correct positions.

When applying the birotation attack to these lattices, we do not have to consider the location of any particular vectors. For the lattices of dimension $2N + 1$ we do not include the last element of the vectors in the rotation. This element can be left unchanged in the birotation, as it represents the number and times selected vectors have been added or subtracted, and this representation remains unchanged in the birotation.

6.5.4 Removing rows

To reduce the dimension of the lattice rows involving the $\text{cir}(h)$ or $\text{cir}(h^{-1})$ block can be removed. As with column removal in zero forcing, removing a

row will leave a zero column, which can also be removed. A number of such rows can be removed from the lattice, but as the qI block has a significant contribution to the expected length of the lattice, reducing the dimension of the qI block decreases s and hence c_h is increased. Specifically for a public key of length N with l rows and zero columns removed has expected length

$$E_L = q^{\frac{N-l}{N}} \sqrt{\frac{(2N-l)\sigma}{\pi e}}$$

while the target vector new has average length

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

The decrease in $\mathbf{y}$ is small compared to the decrease in E_L causing an increase in c_h as the dimension is removed. The expected time needed to recover the secret key is reduced by the reduction in lattice dimension, but increased by the larger c_h. The impact of c_h on reduction time does not seem to be well studied, and experimental results will be needed to determine the effectiveness of row removal and finding the optimal number of rows to remove.

7 Results

We have implemented lattice reduction using the BKZ method of Victor Shoup's NTL package [23]. NTL offers several choices when it comes to floating point precision and orthogonalization techniques. We have primarily used the floating point (FP) version of BKZ as this precision is sufficient and thus offers the highest speed.

The LLL algorithm and improvements to it generally behaves much better than expected by the worst case measures of both running time and achieved reduction. Even though the results are generally much better than expected, proving the improved performance seems to be hard. Running times may vary greatly on fairly similar input, making it hard to predict running times even for similar sized problems. Reduction algorithms are for instance sensitive to the order the lattice vectors. Meskanen [17] finds that randomizing the order of vectors generally reduces the required time to recover the secret key. Our findings in section 7.3 does however dispute this and examplify the unpredictability of lattice reduction methods.

The BKZ algorithm has a complexity that is polynomial in the dimension of the lattice and exponential in the blocksize. We would therefore want to keep the blocksize at a value offering an optimal trade-off between running time and quality of reduction. Unfortunately it is hard to predict both the needed blocksize and the optimal blocksize to find the target vector. The

smallest blocksize that finds the target vector may not be the optimal in reduction time, and Meskanen [17] also finds that higher blocksizes may fail to find the target vector.

When scalars like θ in the zero forcing methods are added or multiplied into the NTRUEncrypt lattice, the optimal value must also be found experimentally. Large numbers will slow down the reduction process, while small numbers may contribute too little to the desired effect, slowing down the calculations. Due to the unpredictabilities described above, we need a number of samples for problems of various sizes to find a optimal values. As our trials have been run on lattices where no such scalars are needed, we have not carried out such tests.

The LLL algorithm is originally specified with Gram-Scmidt orthogonalization, but in NTL it is also with Givens orthogonalization. Gram-Schmidt is the fastest, but becomes unstable as the dimension increases [23], and like [17] we had to shift to Givens orthogonalization to regain stability at the expense of speed.

We made some adjustments to the BKZ method of NTL. As we know the structure of the target vector, we included a search for it in the BKZ method. The search scans the half of each vector corresponding to the polynomial g checking if the coefficients agree with the expected values. When a mismatch is found the search proceeds to the next vector. As we only seek the target vector and do not need a BKZ reduced lattice, we may abort BKZ reduction as soon as the target vector has been found.

We also realized that the matrix representing the lattice in NTL's BKZ method was copied into another matrix on which all operations were performed before it was copied back into the first matrix. We did not study the reason for this redundant matrix, but as removing it didn't cause any problems we have removed it from our version of BKZ, enjoying a decrease in reduction time.

We implemented key generation in MATLAB using the methods for finding the inverse of truncated polynomials from [25]. Binary polynomials with random distribution of ones were used. To mimic the lattice constants c_h and $\frac{N}{q}$ of the standardized parameter set `ees251ep4`, q was chosen to be a prime not much smaller than N and d_f, d_g was chosen such that the ratio of $\frac{d_f}{N}$ and $\frac{d_g}{N}$ was preserved by setting approximately 29% of the coefficients in F and g to zero.

7.1 Results of improved lattices

We tried lattice reduction using different lattices as explained in previous chapters. On successful alterations of the lattice and reduction methods, we experienced noticeable speed-ups. Due to the somewhat unpredictable nature of lattice reduction in the BKZ algorithm, a change decreasing the runtime of one instance might still cause an increase to others. We are

however interested in the general trend, and especially speed-ups that significantly speeds up key recovery for large N.

Starting with the most basic tricks of exploiting p in $f = 1 + pF$, we experienced a significant reduction in running time for all instances. The time to recover the key with a given blocksize was shorter, and it was possible to recover the key with a smaller blocksize. As a smaller blocksize generally implies shorter runtime, this lattice allows even shorter breaking times.

Other adjustments to the lattices to arrive at trick three of Meskanen, randomization of the order of vectors and adjustments to the BKZ routines of NTL all improved the running time in most of the tested instances. We found a good correlation with theory regarding the effect of the lattice constant c_h as expected. The same applies to other attacks such as attacks on the inverse of the private key and removing rows, which turned out not to be faster than trick three of Meskanen. Therefore we chose to carry out experiments using the same construction.

7.2 Breaking times for NTRUEncrypt lattices

In the style of [17], we created NTRUEncrypt public keys with N ranging from 100 to 112, keys that are possible to break within hours or days. We have to choose between reduction methods solving lattices where f has constant term 1 or 3. We have chosen reduction methods and keys with constant term 1, which means that the constant term of F is 0 as this is the most likely case. Five keys of each length were randomly generated and broken.

Experiments where run on a server with two CPUs each of 3.6 GHz with 2MB cache memory and 4GB common DDR2 main memory. As this computer is a shared student facility, we only had access to one of the processors most of the time as the other was in use by another student. At times other processes were run on the computer slowing down all processes. We did not monitor such events, but our experience is that they were limited in number and duration, and thereby only had a small impact on some of the broken keys.

The lattices we tested are equal to those of [17]. The framework for producing lattices from the public key, randomizing the order of the vectors, displaying and storing the result has no considerable impact on total decryption time, essentially all the time is spent in BKZ reduction of the lattice. We can therefore not expect any difference in running time due to differences in the framework. We have however made changes to the NTL library as mentioned above, and would expect a speed-up due to the decrease in memory usage and hopefully an increase due to termination of lattice reduction by the test looking for the target vector.

We ran the tests with an initial blocksize value of 2 and increasing the blocksize until the target vector was found. For LLL reduction we chose

$\delta = 0.99$. In BKZ a parameter called *pruning* is specified. Our initial experiments with different values of the pruning parameter showed that predicting the optimal value for pruning was very hard and that some values even may have a negative effect on running time. We used the pruning value 0 meaning that no pruning was performed.

The results of our experiments are given in Table 1 and breaking time as a function of key length N is shown in Figure 4.

N	b_1	T_1	b_2	T_2	b_3	T_3	b_4	T_4	b_5	T_5
100	24	1689	23	927	22	488	22	384	23	1066
101	23	1382	25	2180	22	933	22	420	23	1030
102	23	2435	25	1093	23	1454	24	2804	23	884
103	23	1115	26	16194	23	6433	23	914	23	3428
104	23	3245	25	17797	25	6673	24	2675	23	2564
105	23	2273	25	24143	24	13058	24	4220	25	37606
106	23	4379	24	19574	24	8748	25	4077	24	26293
107	25	6298	26	15718	24	21092	25	7746	24	17975
108	25	114688	23	7429	24	13965	24	21270	25	62126
109	25	21449	25	15681	24	15394	25	55802	25	54738
110	24	8465	26	241938	25	53030	26	77721	25	49661
111	24	32690	25	142474	25	243523	26	184976	27	124188
112	27	495544	26	690999	25	136306	25	145307	26	650550

Table 1: Needed blocksizes and breaking times in seconds for NTRUEncrypt keys

Analyzing the block sizes and the breaking times we find that the average blocksize as a function of N is $0.225 + 0.38N$. For the breaking times we find the regression line to be

$$\log T = 0.20108N - 17.268.$$

Comparing the breaking times to Meskanen's $\log T = 0.1749N - 14.15$ [17] found using a 700 MHz processor, our results is weaker for large N. Extraplotating the regression lines we find that our results are better for keys with $N \leq 119$ but worse for larger keys. Extrapolating to $N = 251$ which is the smallest key size in NTRUEncrypt recommendations we get a breaking time of about $1.6 \cdot 10^{33}$ seconds or $1.82 \cdot 10^{29}$ MIPS years on this computer.

The NTRU team has also done similar tests obtaining running times that are much better than ours and Meskanen's [7]. They find $\log T = 0.1095N - 12.5402$, but do however use a lattice constant c_h that is much smaller than ours, and their ratio for $\frac{N}{q}$ is also much smaller, contributing to the improved performance.

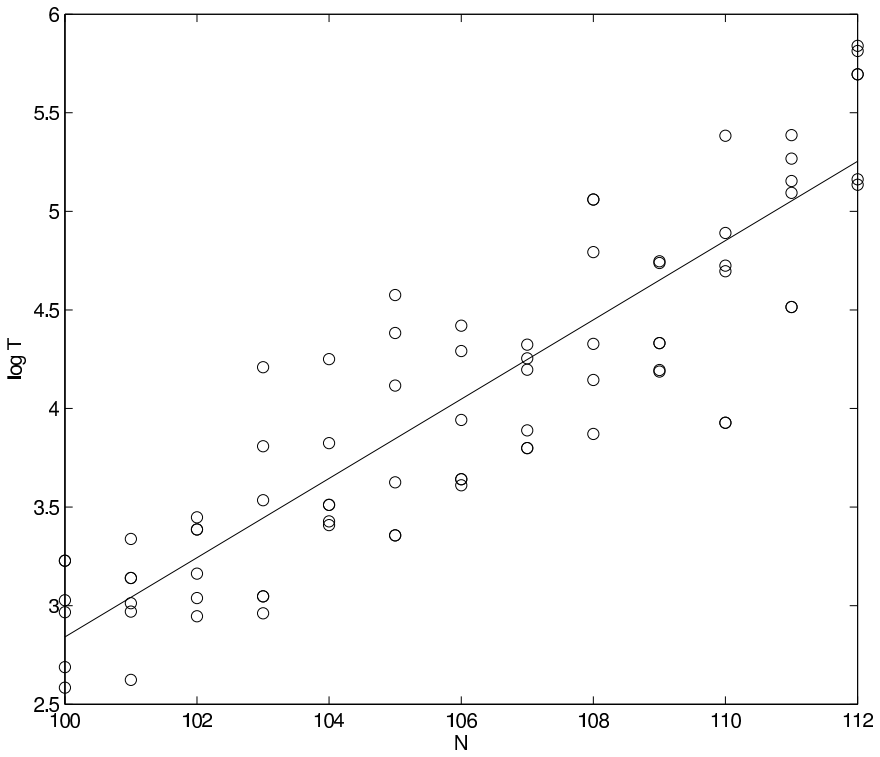


Figure 4: Breaking times of NTRUEncrypt lattices

7.3 Other results

Meskanen [17] notes that randomizing the order of vectors seems to speed up the reduction. We implemented exactly the same code for randomizing of vectors and used it for our key breaking trials given in Table 1. We have also performed a test of the effect of randomization, comparing breaking times for vectors given in the standard order, randomized order, doubly randomized order and in reverse order. The aim of double randomization is only to produce a randomization that differs from the single randomization. The results are given in Table 2 and shows that breaking times are highly dependent on the order of the vectors and that the effect of different order changes is unpredictable. The randomization is also dependent on the seed to the random number generator. We have used the same seed in all our tests, but the randomization and hence the breaking time is also dependent on the seed.

The NTRU team claims that experimental evidence suggests that finding vectors almost as short as the target key as needed for the attack by spurious keys (see 6.3) is unlikely and therefore the attack does not threaten the security of NTRUEncrypt [6] and [7]. On one occasion using the standard

Randomization	Key 1	Key 2	Key 3	Key 4	Key 5
None	1201	522	1158	252	492
Single	1688	925	487	384	1062
Double	549	2451	301	2969	598
Reverse order	800	1465	578	1004	320

Table 2: Breaking times for keys with $N = 100$ for various permutations.

lattice we discovered a solution vector that was almost as short as the target vector, and it turns out that this vector really was the sum of two rotations of the target vector, which we can not expect to be easier to find than the target vector itself.

8 Concluding remarks

We have examined the public key cryptosystem NTRUEncrypt with a main focus on lattice attacks, covering different lattice reduction algorithms and lattices for attacks on NTRUEncrypt. We have shown that breaking NTRUEncrypt keys of recommended security levels in a reasonable time is infeasible with the current state of lattice reduction algorithms and the best known lattices for attacking NTRUEncrypt. We did not have the possibility to test a full range of lattice reduction methods on NTRUEncrypt lattices, but would suggest a further survey of the impact of the lattice constant c_h on breaking times as well as the impact of trade offs between c_h and N in the methods of zero forcing and row removing. These methods for improving the lattice as well as some some lattice reduction enhancements may serve to improve lattice attacks on NTRUEncrypt further.

We have also proposed the key encapsulation method NTRU-KEM, which similarly to NTRUEncrypt with a proper padding scheme is proven secure against adaptive chosen ciphertext attacks.

References

[1] Miklós Ajtai. The Shortest Vector Problem in L_2 is NP-hard for Randomized Reductions. *Electronic Colloquium on Computational Complexity*, TR97-047, 1997.

[2] Johannes Buchmann and Christoph Ludwig. Practical Lattice Basis Sampling Reduction. Cryptology ePrint Archive, Report 2005/072, 2005. `http://eprint.iacr.org/`.

[3] J.W.S. Cassels. *An Introduction to the Geometry of Numbers*. Springer, Berlin, 1971.

[4] Don Coppersmith and Adi Shamir. Lattice Attacks on NTRU. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 52–61. Springer, 1997.

[5] Craig Gentry. Key Recovery and Message Attacks on NTRU-Composite. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2001.

[6] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In Joe Buhler, editor, *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[7] Jeffrey Hoffstein, Joseph H. Silverman, and William Whyte. Estimated breaking times for NTRU lattices. NTRU Cryptosystems Technical Report # 012, Version 2.

[8] Nick Howgrave-Graham, Phong Q. Nguyen, David Pointcheval, John Proos, Joseph H. Silverman, Ari Singer, and William Whyte. The Impact of Decryption Failures on the Security of NTRU Encryption. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 226–246. Springer, 2003.

[9] Nick Howgrave-Graham, Joseph H. Silverman, Ari Singer, and William Whyte. NAEP: Provable security in the presence of decryption failures. Cryptology ePrint Archive, Report 2003/172, 2003. `http://eprint.iacr.org/`.

[10] Nick Howgrave-Graham, Joseph H. Silverman, and William Whyte. A Meet-In-The-Middle Attack on an NTRU Private Key. NTRU Cryptosystems Technical Report # 004, Version 2.

[11] Nick Howgrave-Graham, Joseph H. Silverman, and William Whyte. Choosing Parameter Sets for NTRUencrypt with NAEP and SVES-3.

In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 118–135. Springer, 2005.

[12] A. Korkine and G. Zolotarev. Sur les formes quadratiques. *Mathematische Annalen*, 6(3):366–389, 1873.

[13] Jeffrey C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM Journal of Computing*, 14:196–209, 1985.

[14] Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[15] Alexander May. Cryptanalysis of NTRU. Unpublished preprint, 1999. `http://www.informatik.uni-frankfurt.de/~alex/ntru.ps`.

[16] Alexander May and Joseph H. Silverman. Dimension Reduction Methods for Convolution Modular Lattices. In Joseph H. Silverman, editor, *CaLC*, volume 2146 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2001.

[17] Tommi Meskanen. *On the NTRU Cryptosystem*. PhD thesis, University of Turku, 2005.

[18] John Proos. Imperfect Decryption and an Attack on the NTRU Encryption Scheme. Cryptology ePrint Archive, Report 2003/002, 2003.

[19] C. P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53(2-3):201–224, 1987.

[20] Claus-Peter Schnorr. Lattice Reduction by Random Sampling and Birthday Methods. In Helmut Alt and Michel Habib, editors, *STACS*, volume 2607 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2003.

[21] Claus-Peter Schnorr and M. Euchner. Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. In *Fundamentals of Computation Theory*, pages 68–85, 1991.

[22] Tanya E. Seidel, Daniel Socek, and Michal Sramka. Parallel Symmetric Attack on NTRU using Non-Deterministic Lattice Reduction. *Des. Codes Cryptography*, 32(1-3):369–379, 2004.

[23] Victor Shoup. NTL: A Library for doing Number Theory, version 5.4. `http://www.shoup.net/ntl`.

[24] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. http://eprint.iacr.org/.

[25] J. Silverman. Almost Inverses and Fast NTRU Key Creation. NTRU Cryptosystems Technical Report # 014.

[26] Daniel Socek. Deterministic and Non-deterministic basis reduction techniques for NTRU lattices. Master's thesis, Florida Atlantic University, 2002. Available March 20 2007 from http://www.google.com/search?q=cache:KCUW_t9HcikJ:kaiso.cse.fau.edu/~dsocek/Thesis.pdf+birot+ntru&hl=en&ct=clnk&cd=2&client=opera.

[27] Martijn Stam. A Key Encapsulation Mechanism for NTRU. In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 410–427. Springer, 2005.

[28] Peter van Emde Boas. Another NP-complete partition problem and the complexity of computing short vectors in lattices. Technical Report 04, Mathematics Department, University of Amsterdam, 1981.

[29] Efficient Embedded Security Standard (EEES) version 2.0. Consortium for Efficient Embedded Security, June, 2003.

[30] William White. Choosing NTRUEncrypt Parameters. P1363 working group presentation, March 2004. grouper.ieee.org/groups/1363/WorkingGroup/presentations/Parameters-1363-2004-03.ppt.