CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Symplectic Orthogonalization and Lattice Reduction Techniques
Student:	Peter Bočan
Supervisor:	Ing. Ivo Petr, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2018/19

Instructions

Lattice-based cryptography is a promising alternative to cryptography based on factorization or discrete logarithm problem since neither classical nor quantum algorithm is currently known that would effectively solve general instances of hard lattice problems. The aim of this work is to get acquainted with lattice-based cryptosystems (NTRU in particular) and to investigate lattice reduction techniques.

In particular, the student will
- study mathematical background underlying lattice-based cryptography and give its thorough description
- investigate orthogonalization methods used in lattice reduction algorithms and their symplectic versions presented in paper [1]
- compare the performance of standard and symplectic orthogonalization algorithms using randomly generated NTRU lattices.

References

[1] Gama N., Howgrave - Graham N., Nguyen P. Q. - Symplectic Lattice Reduction and NTRU, Advances in Cryptology - EUROCRYPT 2006. Lecture Notes in Computer Science, vol 4004. Springer, Berlin, Heidelberg, 2006
[2] Silverman J. H., Pipher J., Hoffstein J. - An Introduction to Mathematical Cryptography, Springer New York, 2008
[3] Galbraith S. D., Mathematics of Public Key Cryptography, Cambridge University Press, 2012

prof. Ing. Róbert Lórencz, CSc.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague November 12, 2017

Czech Technical University in Prague
Faculty of Information Technology
Department of Computer Systems

Bachelor's thesis

Symplectic orthogonalization and lattice reduction techniques

Peter Bočan

Supervisor: Ing. Ivo Petr, Ph.D.

15th May 2018

Acknowledgements

Many thanks to my supervisor – Ing. Ivo Petr, Ph.D. for this long adventure through out the past months.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 15th May 2018 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Bočan, Peter. *Symplectic orthogonalization and lattice reduction techniques.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

V tejto práci sme sa zamerali na matematický popis NTRU kryptosystému založenému na ťažkom probléme na bodovej mriežke a otestujeme viacero ortogonalizačných algoritmov popísaných v práci od Nicolasa Gamu nad malými bodovými mriežkami.

Klíčová slova NTRU, symplektická, mriežka, ortogonalizace, Eigen

Abstract

In this thesis we study the underlying mathematical principles that are fundamental for lattice-based cryptosystem, namely NTRU. We have benchmarked algorithms proposed by Nicolas Gama, et al. on a low-dimensional NTRU matrices.

Keywords NTRU, symplectic, lattice, orthogonalisation, Eigen

Contents

List of Figures

List of Tables

Introduction

In the past decades quantum computing got into the real world implementation and it quickly became the very promising way of computing and solving the very hard problems of computer science threatening the online security.

Scientists have developed many cryptosystems, one of which is called NTRU introduced in 1996 by Jeffrey Hoffstein, Joseph H. Silverman and Jill Pipher. The NTRU is a lattice-based cryptosystem which falls into the category of post-quantum cryptography.

Post-quantum cryptography is a new branch cryptography primarily focused on a set of cryptosystems that are resistant to quantum computing.

The NTRU cryptosystem is based on searching the problem of shortest vector in a given lattice, namely NTRU lattice, in contrast to the RSA or ElGamal which are based on factorisation or discrete logarithm problems and they are susceptible to the power of quantum computing.

The search for the shortest vector in a lattice was proven to be $\mathcal{NP}$-hard problem and there are no quantum or classical algorithms known to solve this problem.

All algorithms mentioned in my work serve as ways of solving such a problem, the most notable are Gram-Schmidt or Cholesky, but We will inspect their symplectic and dual versions which may improve their performance.

I have decided to study this area of cutting edge post-quantum cryptography because of the all new concepts, new approaches to cryptography and cybersecurity.

I will explore the mathematical background of a lattice-based cryptography, especially a group of orthogonalization algorithms and their symplectic versions, which are very closely related to the NTRU cryptosystem.

This thesis is based on the research paper of Nicholas Gama, et al. who applied the orthogonalization algorithms on the symplectic lattices which are proportional to the NTRU lattices.

Chapter 1

Theory of lattices

1.1 Background and Motivation

The NTRU cryptosystem, introduced by mathematicians Jeffrey Hoffstein, Joseph H. Silverman and Jill Pipher in 1996 is based on $\mathcal{NP}$-hard problem of finding the shortest vector in a given high-dimensional lattice. The problem is not related to already established factorization or discrete logarithm problem, which are susceptible to Shor's algorithm.[1]

Definition 1 (Lattice)**.** *Let $\mathcal{B} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$ be a n-tuple of linearly independent vectors in $\mathbb{R}^m$. The* lattice Λ generated by $\mathcal{B}$ *is a set of all integral linear combinations*

$$\Lambda(\mathcal{B}) = \left\{ \sum_{i=1}^{n} \alpha_i \mathbf{b}_i : \alpha_i \in \mathbb{Z} \right\}.$$

We say that n-tuple $\mathcal{B}$ is a basis of a lattice Λ *and basis vectors $\mathcal{B}$* generate *a lattice Λ.*

In general, lattice can occupy a m-dimensional vector space, but it can be of a lower dimension n ($m > n$). In that case, *lattice rank* is n, and *lattice dimension* is m. If $m = n$ then lattice is full rank.

Definition 2 (Basis matrix of a lattice)**.** *Let $\mathcal{B} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$ be a n-tuple of linearly independent vectors in $\mathbb{R}^m$ generating a lattice Λ. We can describe a lattice Λ by a $n \times m$ matrix $\mathbb{B}$ as $\Lambda(\mathcal{B}) = \{\mathbf{v}\mathbb{B} : \mathbf{v} \in \mathbb{Z}^m\}$. The basis matrix of a lattice $\mathcal{B}$ is constructed by placing basis vectors $\mathbf{b}_i$ into rows.*

Definition 3 (Inner product)**.** *Let $\mathbf{a}, \mathbf{b} \in \mathbb{R}^m$, then* inner product *is a map $\langle \cdot, \cdot \rangle : \mathbb{R}^m \times \mathbb{R}^m \mapsto \mathbb{R}$ defined as $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{m} \mathbf{a}_i \mathbf{b}_i$ or written in the vector notation as $\mathbf{a}\mathbf{b}^T$. Vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^m$ are orthogonal if and only if $\langle \mathbf{a}, \mathbf{b} \rangle = 0$.*

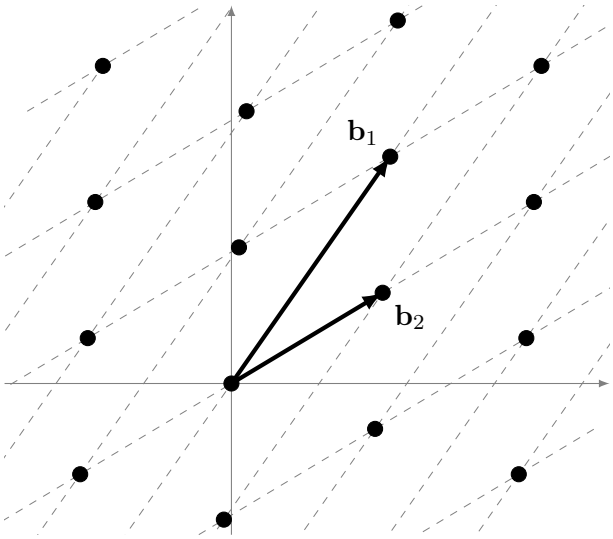


Figure 1.1: An example of a lattice (points) generated by vectors $\mathbf{b}_1, \mathbf{b}_2$

Throughout the thesis we will use $\|\cdot\|$ to refer the standard Euclidean norm, which is defined as $\|\mathbf{a}\| = \sqrt{\langle \mathbf{a}, \mathbf{a} \rangle}$.

Now, before we dive deeper into the properties of lattices, we shall introduce a unimodular matrices and their relation to the lattices.

Remark. General linear group of degree n is the set of all $n \times n$ invertible matrices over field $\mathbb{F}$: $\mathrm{GL}_n(\mathbb{F}) = \{\mathbb{M} \in \mathbb{F}^{n,n} : \text{matrix is invertible}\}$. Especially, the set of all $n \times n$ invertible matrices over field $\mathbb{F}$ with determinant of 1: $\mathrm{SL}_n(\mathbb{F}) = \{\mathbb{M} \in \mathrm{GL}_n(\mathbb{F}) : \det \mathbb{M} = 1\}$.

Lemma 1 (Unimodular matrix)**.** *Let* $\mathbb{U} \in \mathrm{GL}_n(\mathbb{Z})$*, We say that matrix* $\mathbb{U}$ *is* unimodular *if and only if* $\det \mathbb{U} = \pm 1$*.*

Theorem 1 (Properties of unimodular matrix)**.** *Let* $\mathbb{U}, \mathbb{V}$ *be unimodular matrices. Then the following holds*

- $\mathbb{U}^{-1}$ *is unimodular,*
- $\mathbb{U}\mathbb{V}$ *and* $\mathbb{V}\mathbb{U}$ *are unimodular.*

Lemma 2. *Let* $\mathbb{B}$ *be a* $n \times m$ *basis matrix of a lattice* Λ *where* $m > n$ *then there is a linear map* $P : \mathbb{R}^m \mapsto \mathbb{R}^n$ *defined by a matrix* $\mathbb{P} \in \mathbb{R}^{m,n}$ *such that* $P\Lambda$ *is a lattice of rank* n *and* $\forall \mathbf{v} \in \Lambda : \|P\mathbf{v}\| = \|\mathbf{v}\|$ *and for all* $1 \leq i < j \leq n : \langle \mathbf{v}_i, \mathbf{v}_j \rangle = \langle \mathbf{v}_i \mathbb{P}, \mathbf{v}_j \mathbb{P} \rangle$*.*

If the linear map is represented by a $m \times n$ *matrix* $\mathbb{P}$ *then a matrix for the image of* Λ *under the projection* P *is the* $n \times n$ *matrix* $\mathbb{B}\mathbb{P}$ *which is invertible.*

This lemma proves the existence of a projection which projects the a lattice from $\mathbb{R}^m$ to $\mathbb{R}^n$, making the projected lattice a full rank lattice. The projection preserves the Euclidean norm.

Lemma 3. *Two $n \times m$ matrices $\mathbb{B}_1$, $\mathbb{B}_2$ generate the same lattice Λ if and only if $\mathbb{B}_1$ and $\mathbb{B}_2$ are related by unimodular $n \times n$ matrix $\mathbb{U}$. Specifically, $\mathbb{B}_1 = \mathbb{U}\mathbb{B}_2$.*

I will leave out the proof, but this lemma proves the existence of a unimodular matrix $\mathbb{U}$ such that $\mathbb{U}^{-1}\mathbb{B}_1 = \mathbb{B}_2$.

Definition 4 (Integral lattice)**.** *A lattice is* integral *(or integer) if and only if all vectors of its basis have integral entries.*

Because of the nature of computers it will be beneficial to base our cryptography around integers and integral basis vectors of a lattice to avoid operations on floating point numbers as much as possible, as they may introduce rounding errors in computations.

Also, it is clear that we can represent a lattice as a set of points in a vector space, or as a set of vectors, and we will do so interchangeably.

Definition 5 (Determinant of a lattice)**.** *Let Λ be a lattice generated by vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_n) \in \mathbb{R}^m$. Then the* determinant of Λ*, denoted as* $\det \Lambda$ *is the value of* $|\det \mathbb{B}\mathbb{P}|$ *where $\mathbb{P}$ is the projection matrix.*

Lemma 4 (Independence of a basis)**.** *The determinant of a lattice is independent of the choice of basis matrix $\mathbb{B}$ and the choice of projection P.*

Lemma 5 (Hadamard's inequality)**.** *Let Λ be a lattice, take any basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ for Λ and then*

$$\det \Lambda = |\det \mathbb{B}\mathbb{P}| \leq \|\mathbf{b}_1\| \ldots \|\mathbf{b}_n\| .$$

Definition 6 (Hadamard ratio)**.** *Let $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ be a basis of a lattice Λ, then we define the following quantity*

$$\mathcal{H}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \left(\frac{\det \Lambda}{\|\mathbf{b}_1\| \ldots \|\mathbf{b}_n\|} \right)^{1/n}$$

and call it Hadamard ratio.

The Hadamard ratio ranges as $0 < \mathcal{H}(\mathbf{b}_1, \ldots, \mathbf{b}_n) \leq 1$ and it serves us as a good measure of orthogonalization.

Definition 7 (Gram matrix)**.** *Let $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ be a set of vectors. Then matrix $\mathbb{G} \in \mathbb{R}^{n,n}$ is Gramian $\forall i, j \in 1, \ldots, n : \mathbb{G}_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j \rangle$ or written as a matrix $\mathbb{G} = \mathbb{V}\mathbb{V}^T$ where $\mathbb{V}$ is a matrix of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ written in rows. We will refer to Gram matrix of $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ as $G(\mathbf{b}_1, \ldots, \mathbf{b}_n)$.*

We will attribute a volume to a lattice is using the following theorem which will be useful later.

Theorem 2 (Volume of a lattice)**.** *Let Λ be a lattice described by the basis matrix $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ then we can attribute a* volume *to a lattice Λ as follows*

$$\mathrm{vol}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \sqrt{\det G(\mathbf{b}_1, \ldots, \mathbf{b}_n)}$$

1.2 Lattice problems

I will list most notable variants of the $\mathcal{NP}$-hard problems on a lattice, but the fundamental are the first two problems. The symbol $\mathbf{0}$ represents the zero vector of $\mathbb{R}^m$.

Definition 8 (The Shortest Vector Problem (SVP))**.** *Find the shortest nonzero vector in a given lattice Λ, more specifically, minimize the Euclidean norm*

$$\min_{\mathbf{v}\in\Lambda\setminus\{\mathbf{0}\}} \|\mathbf{v}\| .$$

Such vector is a solution to the shortest vector problem.

Definition 9 (The Closest Vector Problem (CVP))**.** *Given a vector $\mathbf{w} \in \mathbb{R}^n$, which does not lie in a lattice Λ, find the shortest nonzero vector in a lattice Λ that is the closest to the vector $\mathbf{w}$. More specifically, minimize the Euclidean norm*

$$\min_{\mathbf{v}\in\Lambda\setminus\{\mathbf{0}\}} \|\mathbf{v}-\mathbf{w}\| .$$

Such vector is a solution to the closest vector problem.

Definition 10 (The Approximate Shortest Vector Problem (apprSVP))**.** *Let Λ be a a lattice of dimension n and $\phi(n)$ be a function of n, find a nonzero vector $\mathbf{v}$ in a lattice Λ that is no more than $\phi(n)$ times longer than the shortest nonzero vector:*

$$\mathbf{v} \leq \phi(n) \|\mathbf{v}_{\text{shortest}}\| .$$

Vector $\mathbf{v}$ is a solution to the approximate shortest vector problem.

Definition 11 (The Approximate Closest Vector Problem (apprCVP))**.** *Given a vector $\mathbf{w} \in \mathbb{R}^n$, which does not lie in a lattice Λ, find the closest nonzero vector in a lattice Λ that is no more than $\phi(n)$ times longer than the closest to the vector $\mathbf{w}$*

$$\mathbf{v} \leq \phi(n) \|\mathbf{v}_{\text{closest}} - \mathbf{w}\| .$$

Vector $\mathbf{v}$ is a solution to the approximate closest vector problem.

However, there might be more than one solution to the problem. For example, a lattice described by the following basis $\{(1,0,0),(0,1,0),(0,0,1)\}$ has three shortest vectors, each of them has the minimal Euclidean norm and every vector suffices as a solution to the problem.

Lemma 6 (Gaussian heuristic)**.** *Let N be a dimension of a lattice Λ. The Gaussian expected shortest length is*

$$\sigma(\Lambda) = \sqrt{\frac{N}{2\pi e}}(\det\Lambda)^{1/N}.$$

The Gaussian heuristic says that a shortest nonzero vector in a randomly generated lattice will satisfy

$$\|\mathbf{v}_{shortest}\| \approx \sigma(\Lambda).$$

1.3 Solving the SVP and CVP in a lattice

Let Λ be a full rank lattice described by the orthogonal basis $(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n) \in \mathbb{R}^n$, i.e. such that $\langle \mathbf{b}_i, \mathbf{b}_j \rangle = 0$ for all $i \neq j$ and we want to solve the SVP and CVP on this particular lattice.

If basis vectors are orthogonal it's easy to solve the SVP and CVP. Every vector $\mathbf{v} \in \Lambda$ has the length given by the following formula

$$\begin{aligned}\|\mathbf{v}\|^2 = \langle \mathbf{v}, \mathbf{v} \rangle &= \langle a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + \cdots + a_n\mathbf{b}_n, a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + \cdots + a_n\mathbf{b}_n \rangle = \\ &= a_1^2\langle \mathbf{b}_1, \mathbf{b}_1 \rangle + a_2^2\langle \mathbf{b}_2, \mathbf{b}_2 \rangle + \cdots + a_n^2\langle \mathbf{b}_n, \mathbf{b}_n \rangle = \\ &= a_1^2 \|\mathbf{b}_1\|^2 + a_2^2 \|\mathbf{b}_2\|^2 + \cdots + a_n^2 \|\mathbf{b}_n\|^2 .\end{aligned}$$

Since $a_1, a_2, \ldots, a_n \in \mathbb{Z}$ then the shortest nonzero vector(s) are in the set of orthogonal basis $\{\pm\mathbf{b}_1, \pm\mathbf{b}_2, \ldots, \pm\mathbf{b}_n\}$.

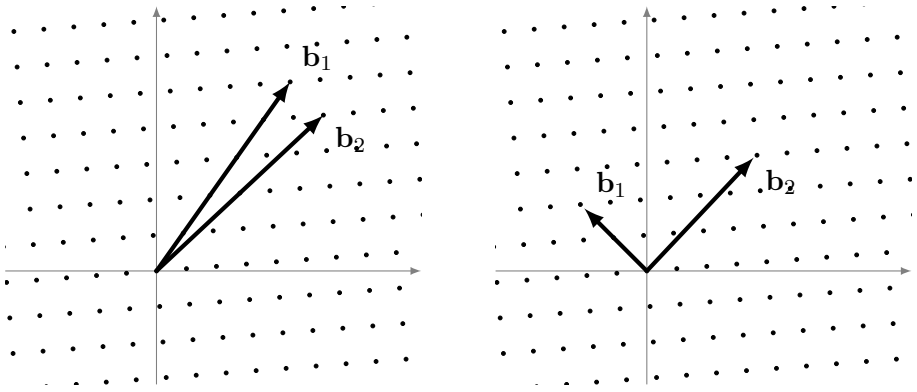
In order to solve the CVP problem in a lattice Λ We will follow similar argument. Let $\mathbf{w}_n \in \mathbb{R}^n$ such that

$$\mathbf{w}_n = s_1\mathbf{b}_1 + s_2\mathbf{b}_2 + \cdots + s_n\mathbf{b}_n \quad \text{where} \quad s_1, s_2, \ldots, s_n \in \mathbb{R}.$$

Then for the vector $\mathbf{w} \in \Lambda$ where $\mathbf{w} = a_1\mathbf{b}_1 + a_2\mathbf{b}_2 + \cdots + a_n\mathbf{b}_n$ we have

$$\|\mathbf{v} - \mathbf{w}\|^2 = (a_1 - s_1)^2 \|\mathbf{b}_1\|^2 + (a_2 - s_2)^2 \|\mathbf{b}_2\|^2 + \cdots + (a_n - s_n)^2 \|\mathbf{b}_n\|^2 .$$

Because $a_1, a_2, \ldots, a_n$ are required to be integers, the given norm is minimised, if we take each a_i to be the closest integer to s_i. Thus solution to the CVP problem is a vector $\mathbf{v} = \sum_{i=1}^{n} \lfloor s_1 \rceil \mathbf{b}_i$ where $\lfloor \cdot \rceil$ is rounding to the nearest integer.



(a) An example of a "bad basis" (b) An example of a "good basis"

Figure 1.2: Two different bases for the same lattice

Therefore solving the SVP and CVP is trivial once the basis of a lattice is orthogonal. We will demonstrate the underlying CVP problem in $\mathbb{R}^2$ geometrically on a simple example where we compare a "good basis" with a "bad basis" (Figure 1.2).

If we try to solve the CVP with a "bad basis" we will not find the correct solution as it is pictured in Figure 1.3. It will miss the closest lattice point and the problem grows with the rank of a lattice.

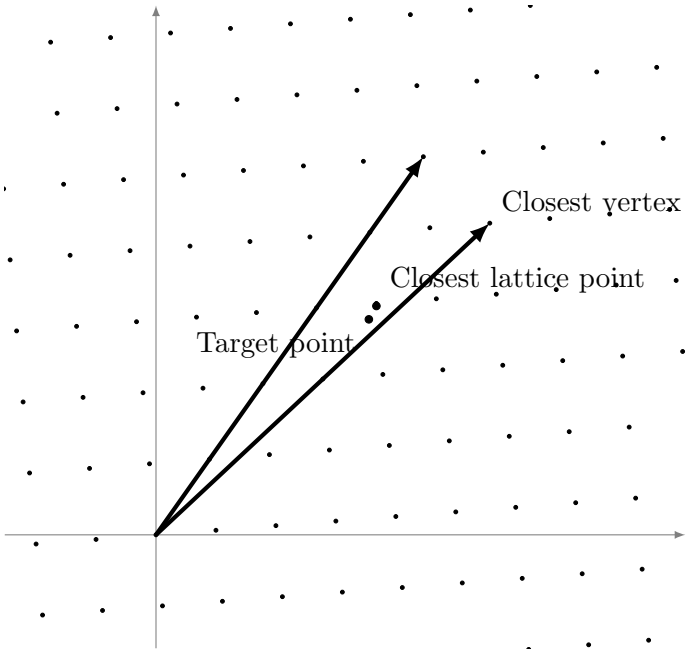


Figure 1.3: An example of a bad solution

Chapter 2

Orthogonalization algorithms

As we have seen, orthogonalization of a lattice solves the SVP and CVP. However, orthogonalization algorithms described in this section serve as underlying layer across linear algebra helping to solve many related computation problems.

Any orthogonalization, given a set of vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ will ompute the set of vectors $(\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$ such that every pair of vectors ($\langle \mathbf{b}_i^*, \mathbf{b}_j^* \rangle = 0$ for all $i \neq j$) is orthogonal.

In this section we will introduce a list of orthogonalization algorithms namely Gram-Schmidt, Lagrange-Gauss, Cholesky and others. We will refer to [2, 3, 4] as the basis for the analysis of mentioned algorithms.

2.1 Gram-Schmidt orthogonalization

The Gram-Schmidt orthogonalization (GSO) algorithm works iteratively on a set of n m-dimensional vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ which components are in any field. The first step is to proclaim the vector $\mathbf{b}_1$ as already "orthogonal". Next step is to project the second vector $\mathbf{b}_2$ onto the vector $\mathbf{b}_1$ and similarly orthogonally projecting k-th vector onto all previous $k-1$ vectors which has been orthogonalized in previous steps.

We define the projection of a vector $\mathbf{v}$ onto $\mathbf{u}$ as a map $\mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}^n$:

$$\mathrm{proj}_{\mathbf{u}}(\mathbf{v}) = \frac{\langle \mathbf{v}, \mathbf{u} \rangle}{\|\mathbf{u}\|^2} \mathbf{u}, \tag{2.1}$$

such map outputs a vector. Computing

$$\mathbf{v}_2^* = \mathbf{v}_2 - \mathrm{proj}_{\mathbf{v}_1}(\mathbf{v}_2)$$

produces a vector $\mathbf{v}_2^*$ which is orthogonal to $\mathbf{v}_1$.

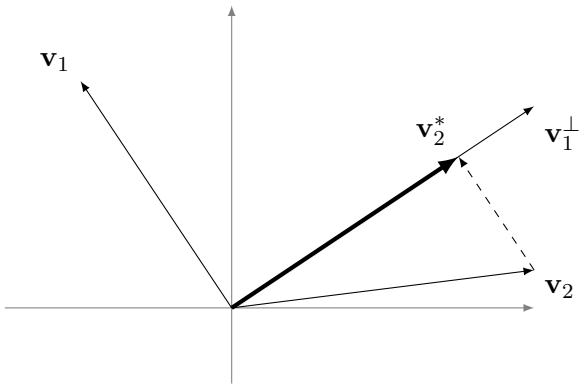


Figure 2.1: A projection of vector $\mathbf{v}_2$ onto the orthogonal vector $\mathbf{v}_1^\perp$ of $\mathbf{v}_1$

Algorithm 1 Gram-Schmidt algorithm

procedure Gram-Schmidt($\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$)
  $\mathbf{b}_1^* \leftarrow \mathbf{b}_1$
  for $k = 2$ upto n **do**
    $\mathbf{d}_k \leftarrow \sum_{i=1}^{k-1} \mathbf{b}_i^* \langle \mathbf{b}_k, \mathbf{b}_i^* \rangle / \|\mathbf{b}_i^*\|^2$
    $\mathbf{b}_k^* \leftarrow \mathbf{b}_k - \mathbf{d}_k$
  end for
  return $(\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$
end procedure

Please note, that throughout this thesis we will refer to matrices as a list of vectors. Matrices are constructed constructed row-wise, if it's not said otherwise.

Algorithm 1 returns a set of vectors that are orthogonal to each other, provides the result in $O(mn^4 \log^2(X))$ operations where X is the upper bound of $\|\mathbf{b}_i\|$ for all i such that $1 \leq i \leq n$.

However Classic GSO is not the best algorithm for the lattice reduction as it is generally numerically unstable in floating point arithmetic. When dealing with vectors over $\mathbb{Z}^n$ in an exact arithmetic in $\mathbb{Q}$ the integers may grow very large. It is commonly referred as *coefficient explosion*.

Also under a closer inspection we can notice that we can express the GSO as a matrix equation as follows

$$\mathbb{B} = \mu \mathbb{B}^*,$$

where $\mu \in \mathrm{GL}_n(\mathbb{R})$ is a lower triangular matrix, $\mathbb{B}$ is the "input" basis and $\mathbb{B}^*$ is the orthogonalized "output" basis. The matrix μ has 1 on the diagonal and under diagonal are the coefficients of the projection $\mu_{k,i} = \langle \mathbf{b}_k, \mathbf{b}_i^* \rangle / \|\mathbf{b}_i^*\|^2$ for all $i < k$. We will exploit this relation in the following section.

2.2 Integral Gram-Schmidt

Integral GSO constructed in the paper [5, p. 7] is specifically built for the "input" basis vectors in $\mathbb{Z}^m$. In that case the $\mathbf{b}_i^*$ and $\mu_{i,j}$ are in general rational. To avoid rational arithmetic we will use integral quantities, namely matrix $\lambda \in \mathrm{GL}_n(\mathbb{Z})$, for all $1 \leq i \leq n$: let $\lambda_{0,0} = 1$ and $\lambda_{i,i} = \prod_{j=1}^{i} \|\mathbf{b}_j^*\|^2 = \mathrm{vol}(\mathbf{b}_1, \ldots, \mathbf{b}_i)^2 \in \mathbb{Z}$ and let the $\lambda_{i,j} = \mu_{i,j}\lambda_{j,j}$ for all $j < i$.

Recall the GSO algorithm where we computed $\mu_{i,j}$ as $\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle / \|\mathbf{b}_j^*\|^2$. Multiplying it $\lambda_{j,j}$ we obtain $\lambda_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle \lambda_{j-1,j-1} \in \mathbb{Z}$.

All steps lead us to replace the μ matrix with integral lower triangular matrix λ which allows us efficiently compute the lower triangular μ matrix ($\mu_{i,j} = \lambda_{i,j}/\lambda_{j,j}$), if necessary. For lattice reduction purposes λ suffices matrix well enough.

Algorithm 2 Integral Gram-Schmidt algorithm

```
1:  procedure IntegralGram-Schmidt(b_1, ..., b_n ∈ Z^m )
2:      for i = 1 upto n do
3:          λ_{i,1} ← ⟨b_i, b_1⟩
4:          for j = 2 upto i do
5:              S = λ_{i,1} λ_{j,1}
6:              for k = 2 upto j − 1 do
7:                  S ← (λ_{k,k} S + λ_{j,k} λ_{i,k}) / λ_{k−1,k−1}
8:              end for
9:              λ_{i,j} ← ⟨b_i, b_j⟩ λ_{j−1,j−1} − S
10:         end for
11:     end for
12:     return λ
13: end procedure
```

Time complexity of the algorithm 2 is $O(mn^2 \log^2 |B|)$ where $|B|$ is an upper bound of $\|\mathbf{b}_i\|$.

2.3 Lagrange-Gauss lattice reduction

For a lattice of a dimension of 2 there exists a very efficient algorithm with solves the SVP and returns a "good basis" for the given lattice. This algorithm was published by Gauss, but was known to Lagrange.

This algorithm however does not work for higher dimensions. If we had a lattice of a rank n ($n > 2$), we would have to find the right linear combination (Algorithm 3, line 6) for a vector $\mathbf{b}_n$ in a lattice of a rank $n - 1$ generated by vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_{n-1})$ which leads us to the search for the solution of the CVP.

It's very similar to the Classical GSO with two nuances, the first being the swapping of a larger vector with the smaller one (in the comparison of the two norms) and the second one is rounding to the nearest integer.

The following algorithm executes $O(\log^3 n)$ operations.

Algorithm 3 Lagrange-Gauss Lattice Reduction

```
1: procedure LAGRANGE-GAUSS(b_1, b_2 ∈ R^2 )
2:     while true do
3:         if ||b_2|| < ||b_1|| then
4:             swap b_1 ↔ b_2
5:         end if
6:         μ = ⌊⟨b_1, b_2⟩ / ||b_1||^2⌉
7:         if μ = 0 then
8:             return (b_1, b_2)
9:         end if
10:        b_2 ← b_2 − μb_1
11:    end while
12: end procedure
```

2.4 Cholesky decomposition

Cholesky decomposition is a symmetric version to more general $\mathbb{LU}$ decomposition where $\mathbb{L}$ is lower triangular matrix and $\mathbb{U}$ is an upper triangular matrix. Cholesky decomposition requires $\mathbb{B} \in \mathbb{R}^{n,n}$ to be a symmetric definite positive matrix then there exists a lower triangular matrix $\mathbb{L} \in \mathbb{R}^{n,n}$ with strictly positive diagonal such that $\mathbb{B} = \mathbb{L}\mathbb{L}^T$. Algorithm returns only the lower triangular matrix $\mathbb{L}$. It's also important to note that orthogonalization is hidden in the following fact that $\mathbb{B}(\mathbb{L}^{-1})^*$.

We will derive the algorithm from the definition of the Cholesky decomposition. A simple decomposition example in $\mathbb{R}^{3,3}$:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{pmatrix}$$

The resulting matrix is as follows

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11}^2 & l_{21}l_{11} & l_{31}l_{11} \\ l_{21}l_{11} & l_{21}^2 + l_{22}^2 & l_{31}l_{21} + l_{32}l_{22} \\ l_{31}l_{11} & l_{31}l_{21} + l_{32}l_{22} & l_{31}^2 + l_{32}^2 + l_{33}^2 \end{pmatrix}.$$

2.4.1 Generalisation of Cholesky decomposition

Let $\mathbb{L} \in \mathrm{GL}_n(\mathbb{R})$ be a symmetric definite positive matrix. Every coefficient can be computed iteratively, for all $1 \leq k \leq n$ computed as follows:

$$l_{kk} = \sqrt{a_{kk} - \sum_{i=1}^{k-1} l_{ki}^2}$$

and coefficients under the main diagonal can be computed as

$$(\forall i : 1, \ldots, n)\,(\forall k : k < n) : l_{ik} = \frac{1}{l_{kk}} \left(a_{ik} - \sum_{i=1}^{k-1} l_{ij} l_{kj} \right).$$

The following algorithm executes $O(mn^2)$ operations.

Algorithm 4 Cholesky decomposition

```
1: procedure CHOLESKY-DECOMPOSITION(b_1, ..., b_n ∈ R^n)
2:     for i = 1 upto n do
3:         for j = 1 upto i do
4:             if i = j then
5:                 l_ij ← sqrt(b_ii − Σ_{m=1}^{i−1} l_im^2)
6:             else
7:                 s ← Σ_{i=1}^{i−1} l_ij l_kj
8:                 l_ij ← (b_ik − s)/l_kk
9:             end if
10:        end for
11:    end for
12:    return L
13: end procedure
```

2.5 $\mu D\mu^T$ factorization

The $\mu D\mu^T$ factorization mentioned in the work of Gama [5] is based on an matrix $\mathbb{LDL}^T$ factorization, where given matrix $\mathbb{A}$ can be expressed as a multiplication of three matrices, where $\mathbb{L}, \mathbb{M}$ are lower triangular matrices and $\mathbb{D} = \mathrm{diag}(d_1, d_2, \ldots, d_n)$, such that $\mathbb{A} = \mathbb{LDM}^T$.

If matrix $\mathbb{A}$ is symmetric then we can save some effort using a variant of $\mathbb{LDM}^T$ called $\mathbb{LDL}^T$ factorization, showing that $\mathbb{M} = \mathbb{L}$.

The couple (μ, D) is a $\mu D\mu^T$ *factorization* of $\mathbb{A}$. Similarly as in the case of Cholesky, we will derive the algorithm from a simple example. We want to factorise a matrix into three matrices such that

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} d_{11} & 0 & 0 \\ 0 & d_{22} & 0 \\ 0 & 0 & d_{33} \end{pmatrix} \begin{pmatrix} 1 & l_{21} & l_{31} \\ 0 & 1 & l_{32} \\ 0 & 0 & 1 \end{pmatrix}$$

obtaining the following result

$$\begin{pmatrix} d_{11} & l_{21}d_{11} & l_{31}d_{11} \\ l_{21}d_{11} & l_{21}^2 d_{11} + d_{22} & l_{21}l_{32}d_{22} + l_{32}d_{22} \\ l_{31}d_{11} & l_{21}l_{32}d_{22} + l_{32}d_{22} & l_{31}^2 d_{11} + l_{32}^2 d_{22} + d_{33} \end{pmatrix}.$$

2.5.1 Generatization of $\mu D \mu^T$ factorization

Now, we can observe how to compute entries of matrices $\mathbb{L}$ and $\mathbb{D}$. For the diagonal matrix $\mathbb{D}$ we do the following

$$1 \leq i \leq n : d_{ii} = a_{ii} - \sum_{k=1}^{j-1} l_{jk}^2 d_{kk}$$

and for the $\mathbb{L}$ matrix, $1 \leq i, j \leq n, i \leq j$

$$l_{ij} = \frac{1}{d_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_{kk} l_{jk} \right).$$

It's very similar to Cholesky factorization, computing starts with the computing d_{11} and expanding row-wise from left to right. This algorithm requires $O(n^3)$.

Algorithm 5 $\mu D \mu^T$ algorithm

```
1: procedure μDμ^T-FACTORIZATION(b_1, ..., b_n ∈ R^n )
2:     d_11 ← b_11
3:     for i = 1 upto n do
4:         for j = 1 upto i do
5:             if i = j then
6:                 d_ij ← b_ii − Σ_{k=1}^{j−1} l_jk^2 d_kk
7:             else
8:                 s ← b_ij − Σ_{k=1}^{j−1} l_ik d_kk l_jk
9:                 l_ij ← s/d_jj
10:            end if
11:        end for
12:    end for
13:    return L, D
14: end procedure
```

2.6 LQ decomposition

Definition 12 (Orthogonal matrix)**.** *Let $\mathbb{Q} \in \mathbb{F}^{n,n}$. Matrix $\mathbb{Q}$ is orthogonal if and only if $\mathbb{Q}^T\mathbb{Q} = \mathbb{Q}\mathbb{Q}^T = \mathbb{E}$.*

From definition 12 follows that, if a matrix is orthogonal then we can interchange its transpose with its inverse, from these facts it's clear that $\det\left(\mathbb{Q}^T\mathbb{Q}\right) = \det\mathbb{E} = 1$.

Before computing LQ decomposition, which as a result returns two matrices – $\mathbb{L}$ a lower triangular matrix and $\mathbb{Q}$ which is a orthogonal matrix, is necessary to introduce the QR decomposition, where $\mathbb{Q}$ is orthogonal and $\mathbb{R}$ is upper triangular matrix.

The QR decomposition is introduced on a rectangle matrix $\mathbb{A} \in \mathbb{R}^{m,n}$ where $m \geq n$. Factoring it into matrix $\mathbb{Q} \in \mathbb{R}^{m,m}$ and $\mathbb{R} \in \mathbb{R}^{m,n}$ and the computation can be done in many ways including Gram-Schmidt. The most of the variants are described in the publication of Golub and Van Loan [3]. The output of the QR decomposition produces an orthonormal vector basis written in the first n columns of $\mathbb{Q}$.

To achieve the LQ decomposition it's only necessary to transpose the input matrix $\mathbb{A}$, because transposing $\mathbb{A} = \mathbb{Q}\mathbb{R}$ will lead to $\mathbb{A}^T = \mathbb{R}^T\mathbb{Q}^T$, and because $\mathbb{L} = \mathbb{R}^T$.

We will base the QR decomposition on the GSO. Let $\mathbb{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_n) \in \mathbb{R}^{m,n}$ be a column-wise matrix. Matrix $\mathbb{Q} = (\mathbf{q}_1, \ldots, \mathbf{q}_m) \in \mathbb{R}^{m,m}$ is a *Q-factor* of a matrix $\mathbb{A}$, and matrix $\mathbb{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_n) \in \mathbb{R}^{m,n}$ is a *R-factor* of a matrix $\mathbb{A}$.

Recall the GSO from the 1 which will generate orthogonal basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$. The Q-factor is constructed by placing orthonormal vectors $\mathbf{q}_k = \mathbf{b}_k / \|\mathbf{b}_k\|$ from the GSO in the columns. $\mathbb{Q} = \begin{pmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \ldots & \mathbf{q}_n\end{pmatrix}$, and constructing the R-factor as follows

$$\mathbb{R} = \begin{pmatrix} \langle \mathbf{a}_1, \mathbf{q}_1 \rangle & \langle \mathbf{a}_2, \mathbf{q}_1 \rangle & \langle \mathbf{a}_3, \mathbf{q}_1 \rangle & \ldots & \langle \mathbf{a}_n \mathbf{q}_1 \rangle \\ 0 & \langle \mathbf{a}_2, \mathbf{q}_2 \rangle & \langle \mathbf{a}_3, \mathbf{q}_2 \rangle & \ldots & \langle \mathbf{a}_n, \mathbf{q}_2 \rangle \\ 0 & 0 & \langle \mathbf{a}_3, \mathbf{q}_3 \rangle & \ldots & \langle \mathbf{a}_n, \mathbf{q}_3 \rangle \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \langle \mathbf{a}_n, \mathbf{q}_n \rangle \end{pmatrix}.$$

Algorithm 6 QR decomposition based on GSO

```
1: procedure QR-Decomposition(b_1, ..., b_n ∈ R^m)
2:     (b*_1, ..., b*_n) ← Gram-Schmidt(b_1, ..., b_n)
3:     for i = 1 upto n do
4:         q_i ← b*_i / ||b*_i||
5:         for j = i upto n do
6:             r_ji ← ⟨a_j, q_i⟩
7:         end for
8:     end for
9:     return (Q, R)
10: end procedure
```

Thus producing the LQ decomposition is trivial.

Algorithm 7 LQ decomposition

1: **procedure** LQ-Decomposition($\mathbb{A} \in \mathbb{R}^{m,n}$)
2: $\mathbb{Q}, \mathbb{R} \leftarrow$ QR-Decomposition($\mathbb{A}^T$)
3: **return** $(\mathbb{R}^T, \mathbb{Q})$
4: **end procedure**

Chapter 3

NTRU

In 2017, Computer Security Resource Centre of National Institute of Standards and Technology (CSRC NIST) have called [6] for the papers for Post-Quantum Cryptography Standardization. NTRUEncrypt, proposed by Jeffrey Hoffstein et al., is one of many cryptosystems.

The NTRU (pronounced as *en-tru*) cryptosystem is can be formulated using polynomials over rings. This cryptosystem was introduced in the 2006 in the paper of J. Hoffstein, et al. [7]. This part of the thesis is heavily reliant upon the book of Hoffstein, et al. [4]

The main advantage against the other public key cryptosystems is that it does not depend on the generating a pair of large primes, which takes up a lot of time and all NTRU operations are quick – encryption, decryption, public key derivation are done within $O(N^2)$ operations, where N is the dimension of the lattice.

3.1 Polynomials

Definition 13 (Quotient ring)**.** *Let R be a ring and let $m \in R, m \neq 0$. For any $a \in R$ we write $\overline{a}$ for the set of all $a' \in R$ such that $a\prime \equiv a \pmod{m}$. The set $\overline{a}$ is called a congruence class. We call $R/(m)$ a* quotient ring *of R by m.*

Definition 14 (Polynomial ring over $\mathbb{Z}_q$)**.** *Let q be a prime. Let R be a ring and let $m \in R, m \neq 0$. For any $a \in R$ we write $\overline{a}$ for the set of all $a' \in R$ such that $a\prime \equiv a \pmod{m}$. The set $\overline{a}$ is called a congruence class. We call $R/(m)$ a* quotient ring *of R by m.*

Definition 15 (Ring of convolution polynomials)**.** *Let N be a natural number. The* ring of convolution polynomials *of rank N is the quotient ring $R = \mathbb{Z}[x]/(x^N - 1)$.*

Definition 16 (Ring of convolution polynomials modulo q)**.** *Let N be a natural number. The* ring of convolution polynomials *of rank N is the quotient ring $R_q = (\mathbb{Z}/q\mathbb{Z})[x]/(x^N - 1)$.*

Definition 17 (Convolution product of polynomials in a quotient ring)**.** *Let $\mathbf{a}(x), \mathbf{b}(x) \in R$. The* convolution product of polynomials *of degree N is a polynomial $\mathbf{c}(x)$, such that*

$$\mathbf{a}(x) \star \mathbf{b}(x) = \mathbf{c}(x) \text{ where } c_k = \sum_{i+j \equiv k \pmod N} \mathbf{a}_i \mathbf{b}_{k-i}, \forall i, j \in 0, \ldots, N-1$$

Definition 17 refers to classical product of two polynomials but there are two additional operations – reducing the degree of a polynomial and, if applicable, reducing the coefficients modulo q.

Example. Let $\mathbf{a}(x) = 1 + 4x + 2x^4 - 5x^6, \mathbf{b}(x) = 1 - 7x - x^2 - 6x^3$ and $N = 6, q = 13$.

$$\begin{aligned}\mathbf{a}(x) \star \mathbf{b}(x) &= 30x^9 + 5x^8 + 23x^7 - 7x^6 - 14x^5 - 22x^4 - 10x^3 - 29x^2 - 3x + 1 \\ &= 30x^3 + 5x^2 + 23x^1 - 7 - 14x^5 - 22x^4 - 10x^3 - 29x^2 - 3x + 1 \\ &= -14x^5 - 22x^4 + 20x^3 - 24x^2 + 20x - 6 \quad \text{in} \quad \mathbb{Z}[x]/(x^6 - 1).\end{aligned}$$

Now we want to reduce the coefficients modulo $q = 13$, so we get

$$\mathbf{a}(x) \star \mathbf{b}(x) = 12x^5 + 4x^4 + 7x^3 + 2x^2 + 7x + 7 \text{ in } (\mathbb{Z}/13\mathbb{Z})[x]/(x^6 - 1).$$

Lemma 7. *Let q be a prime. Then $\mathbf{a}(x) \in R_q$ has a multiplicative inverse in R_q if and only if*

$$\gcd(\mathbf{a}(x), x^N - 1) = 1 \text{ in } (\mathbb{Z}/q\mathbb{Z})[x]$$

and can be computed by Extended Euclidean Algorithm applied on polynomials.

Definition 18 (Set of ternary polynomials)**.** *For any natural numbers d_1 and d_2 we define a set of ternary polynomials $\mathcal{T}(d_1, d_2)$ as follows*

$$\mathcal{T}(d_1, d_2) = \left\{ \begin{array}{c} \mathbf{a}(x) \text{ has } d_1 \text{ coefficients equal to } 1, \\ \mathbf{a}(x) \in R : \mathbf{a}(x) \text{ has } d_2 \text{ coefficients equal to } -1, \\ \mathbf{a}(x) \text{ has all others coefficients equal to } 0 \end{array} \right\}.$$

The set $\mathcal{T}$ is telling us how many coefficients within any polynomial from the set have value 0 or ± 1.

Definition 19 (Concatenating of two vectors)**.** *Let $\mathbf{h}(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}, \mathbf{g}(x) = b_0 + b_1 x + \cdots + b_{n-1}x^{n-1}$ be polynomials in $\mathbb{Z}[x]/(x^N - 1)$ then we will compound a new vector $(\mathbf{h}, \mathbf{g}) = (a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}) \in \mathbb{Z}^{2n}$.*

3.2 NTRU cryptosystem

We can now start building a NTRU cryptosystem. We need to pick four numbers (N, p, q, d) such that N is a prime, $\gcd(p, q) = \gcd(N, q) = 1$ and $q > (6d + 1)p$. The last inequality is very important in order to successfully decrypt the encrypted message.

It may not seem obvious why the very first parameter must be a prime number, it was proposed by Silverman to use the the power of two in order to apply Fast Fourier Transformations, but it was discovered that if N is composite then the NTRU cryptosystem is not secure.

The parameter N is used to define the degree of polynomials to be at most $N-1$, parameter q will be used to truncate the coefficients of the polynomials and parameter p is used for the decryption.

For illustration purposes we will refer to the Table 3.1 to demonstrate how large these parameters according to [8].

Table 3.1: NTRU parameters

type	N	q	p
Moderate Security	167	128	3
Standard Security	251	128	3
High Security	347	128	3
Highest Security	503	256	3

Because the NTRU is an asymmetric lattice-based cryptosystem, it requires to have both public and private keys. Firstly, Alice publishes the mentioned four numbers and derives the public key from the private keys.

3.2.1 Public key derivation

In order for Alice to generate the public key, she must establish two polynomials $\mathbf{f}(x), \mathbf{g}(x)$ as follows $\mathbf{f}(x) \in \mathcal{T}(d+1, d)$ and $\mathbf{g}(x) \in \mathcal{T}(d, d)$. From these private keys, she computes inverses

$$\mathbf{F}_q(x) = \mathbf{f}(x)^{-1} \quad \text{in } R_q \quad \text{and} \quad \mathbf{F}_p(x) = \mathbf{f}(x)^{-1} \quad \text{in } R_p,$$

and Alice repeats this until she successfully established the inverses of the key. Now, she computes $\mathbf{h}(x)$, which will be hers public key

$$\mathbf{h}(x) = \mathbf{F}_q(x) \star \mathbf{g}(x) \quad \text{in } R_q$$

The vector $(\mathbf{f}, \mathbf{F}_p)$ is Alice's private key. She can, however, store only $\mathbf{f}$ and compute $\mathbf{F}_p$ later.

3.2.2 Encryption

Bob chooses a plain text (message) $\mathbf{m} \in R_p$ and also chooses a random ternary polynomial $\mathbf{r} \in \mathcal{T}(d, d)$ then he proceeds and encrypts the message with Alice's public key $\mathbf{h}$ as follows

$$\mathbf{e} \equiv p\mathbf{r} \star \mathbf{h} + \mathbf{m} \quad \text{in } R_q.$$

3.2.3 Decryption

Definition 20 (Centered lift)**.** *Let* $\mathbf{a}(x) \in R_q$*. The* centered lift *of a polynomial* $\mathbf{a}$ *to* R *is a unique polynomial* $\mathbf{a}'(x) \in R$*. Satisying* $\mathbf{a}'(x) \mod q = \mathbf{a}(x)$ *whose integral coefficients are chosen in the interval* $a'_i \in (-q/2, q/2]$*.*

Alice computes $\mathbf{f} \star \mathbf{e} \equiv p\mathbf{g} \star \mathbf{r} + \mathbf{f} \star \mathbf{m} \quad \text{in } R_q$ and centerlifts $\mathbf{f} \star \mathbf{e}$ (and call it $\mathbf{a}$) then she computes

$$\mathbf{m} \equiv \mathbf{F}_p \star \mathbf{a} \quad \text{in } R_p$$

.

3.2.4 The NTRU key recovery problem

Definition 21 (The NTRU Key Recovery Problem)**.** *Given* $\mathbf{h}(x)$*, find a ternary polynomials* $\mathbf{f}(x), \mathbf{g}(x)$ *satisfying*

$$\mathbf{f}(x) \star \mathbf{h}(x) \equiv \mathbf{g}(x) \quad \text{in } R_q$$

It is important to note that the pair $(\mathbf{f}, \mathbf{g})$ which solves the given problem is not unique, another solutions are $(x^k \star \mathbf{f}, x^k \star \mathbf{g}), \forall k \in \{0, \ldots, N-1\}$. The polynomial $x^k \star \mathbf{f}$ is called a *rotation of* $\mathbf{f}$.

In order to brute-force this problem Eve would have to find $\mathbf{f}$ or $\mathbf{g}$ go through all ternary polynomials in a polynomial ring R, which consists of $\#\mathcal{T}(d_1, d_2)$ polynomials. We can compute the size of the set as follows

$$\#\mathcal{T}(d_1, d_2) = \frac{N!}{d_1! d_2! (N - d_1 - d_2)!}$$

number of polynomials is maximised if d_1 and d_2 are approximately $N/3$. However, because all rotations of a private key are a solution to the key recover problem, Eve has to check approximately $\#\mathcal{T}(d_1, d_2)/N$ polynomials.

3.3 NTRU lattices

In this section we are going to interconnect the lattice theory with the NTRU cryptosystem.

Definition 22 (Cyclical permutation matrix)**.** *Let* $\mathbf{h}(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}, a_0, \ldots, a_{n-1} \in \mathbb{Z}$ *be a polynomial of degree* n*. Then we define a* cyclical permutation matrix $\mathbb{P}_{\mathbf{h}} \in \mathbb{Z}^{n,n}$ *such that*

$$\mathbb{P}_{\mathbf{h}} = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-2} & a_{n-1} \\ a_{n-1} & a_0 & \cdots & a_{n-3} & a_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_2 & a_3 & \cdots & a_0 & a_1 \\ a_1 & a_2 & \cdots & a_{n-1} & a_0 \end{pmatrix}$$

Definition 23 (NTRU matrix)**.** *Let* $\mathbf{h}$ *be a public key and* $\mathbb{P}_{\mathbf{h}}$ *be its permutation matrix,* $\mathbb{E}$ *is an identity matrix and* $\mathbf{\Theta}$ *is a zero matrix. Then an NTRU lattice* $\Lambda_{\mathbf{h}}^{NTRU}$ *is described by the matrix* $\mathbb{M}_{\mathbf{k}}^{NTRU} \in \mathrm{GL}_{2n}(\mathbb{Z})$

$$\mathbb{M}_{\mathbf{k}}^{NTRU} = \begin{pmatrix} \mathbb{E} & \mathbb{P}_{\mathbf{h}} \\ \Theta & q\mathbb{E} \end{pmatrix}$$

Lemma 8. *Let* $\mathbf{f}, \mathbf{g}$ *be private polynomials, which were used to generate the public key* $\mathbf{h}(x)$ *Assume* $\mathbf{f}(x) \star \mathbf{h}(x) \equiv \mathbf{g}(x)$ in R_q *and let* $\mathbf{u}(x) \in R$ *be the polynomial satisfying*

$$\mathbf{f}(x) \star \mathbf{h}(x) = \mathbf{g}(x) + q\mathbf{u}(x)$$

Then

$$(\mathbf{f}, -\mathbf{u})\mathbb{M}_{\mathbf{h}}^{NTRU} = (\mathbf{f}, \mathbf{g}),$$

this means that $(\mathbf{f}, \mathbf{g}) \in \Lambda_{\mathbf{h}}^{NTRU}$ *and its rotations are the shortest vectors in the NTRU lattice.*

Now we will assume that $d \approx N/3$ and $q \approx 6d \approx 2N$. The Gaussian heuristic predicts the shortest nonzero vector in a lattice $\sigma(\Lambda_{\mathbf{h}}^{NTRU}) \approx \sqrt{Nq/\pi e} \approx 0.484N$. Because both vectors $\mathbf{f}$ and $\mathbf{g}$ have $2d$ coefficients non-zero, resulting vector $(\mathbf{f}, \mathbf{g})$ has approximately $4d$ non-zero coefficients hence $\|(\mathbf{f}, \mathbf{g})\|$. Therefore

$$\frac{\|(\mathbf{f}, \mathbf{g})\|}{\sigma(\Lambda_{\mathbf{h}}^{NTRU})} \approx \frac{2.39}{\sqrt{N}},$$

so the vector $(\mathbf{f}, \mathbf{g})$ is a factor of $O(1/\sqrt{N})$ than the Gaussian heuristic for the shortest vector and thus vector $(\mathbf{f}, \mathbf{g})$ is the shortest vector in a NTRU lattice.

Chapter 4

Symplectic and dual orthogonalization algorithms

In the previous chapter we have demonstrated that NTRU cryptosystem is based around lattices and the problem of orthogonality. However, we will demonstrate in this chapter that NTRU lattices are actually q-symplectic.

Before we dive into the symplectic versions of orthogonalization algorithms mentioned in the work of Gama [5], we shall explain the symplecticity, duality and other variants.

4.1 Symplectic group, symplecticity, duality

Definition 24 (Isometry)**.** *Let X and Y be metric spaces with metrics d_X and d_Y. A map $f : X \to Y$ is called an isometry (or distance preserving) if for any $a, b \in X$ one has*

$$d_Y(f(x), f(y)) = d_X(x, y).$$

Definition 25 (Dual lattice)**.** *Let Λ be a lattice. We define a* dual lattice $\Lambda^\times$ *such as*

$$\Lambda^\times = \{\mathbf{v} \in \operatorname{span} \Lambda, \forall \mathbf{u} \in \Lambda : \langle \mathbf{v}, \mathbf{u} \rangle \in \mathbb{Z}\}$$

Definition 26 (Quadrant notation)**.** *Let $\mathbb{A}, \mathbb{B}, \mathbb{C}, \mathbb{D} \in \mathrm{GL}_n(\mathbb{F})$ then we define*

$$Q[\mathbb{A}, \mathbb{B}, \mathbb{C}, \mathbb{D}] = \begin{pmatrix} \mathbb{A} & \mathbb{B} \\ \mathbb{C} & \mathbb{D} \end{pmatrix} \in \mathrm{GL}_{2n}(\mathbb{F})$$

Definition 27 (Symplectic matrix)**.** *Let $\mathbb{M} \in \mathrm{GL}_{2n}(\mathbb{F})$. and $\mathbb{J} = Q[0, \mathbb{E}_n, -\mathbb{E}_n, 0]$. We say that matrix is symplectic if and only if $\mathbb{M}^T \mathbb{J} \mathbb{M} = \mathbb{J}$.*

Definition 28 (Isodual lattice)**.** *A lattice Λ is said to be* isodual *if there exists an isometry σ of Λ onto its dual lattice.*

What the definition tells us is that between a lattice and its dual lattice there is an isometry. A map that preserves the distances is useful in

Definition 29 (Symplectic lattice)**.** *Let Λ be a isodual lattice, and let τ be a isometry of Λ. We say that lattice Λ is* symplectic *if and only if $\tau^2 = -1$.*

In other words, if there exists an orthogonal basis of span Λ over which the matrix of τ is $\mathbb{J}$ then there is a basis Λ whose Gram matrix is symplectic. A full rank symplectic lattice Λ has a volume of 1.

Definition 30 (q-symplectic lattice)**.** *A lattice $\Lambda \in \mathbb{Z}^{2n}$ is* q-symplectic *if and only if the lattice $\Lambda/\sqrt{q}$ is symplectic, where $q \in \mathbb{N}_0$ and its volume is q^n.*

4.2 Symplectic LQ

We will implement the symplectic version of QR factorisation based on the symplectic version of GSO.

Time complexity of this algorithm is exactly as in the case of Algorithm 9 $O(n^5 \log^2 B)$

Algorithm 8 Symplectic LQ algorithm

```
1: procedure SymplecticQR(b_1, ..., b_2n ∈ Z^2n )
2:     (b*_1, ..., b*_n) ← SymplecticGram-Schmidtt(b_1, ..., b_n)  ▷ Algorithm 9
3:     for i = 1 upto n do
4:         q_i ← b*_i / ||b*_i||
5:         for j = i upto n do
6:             r_ji ← ⟨a_j, q_i⟩
7:         end for
8:     end for
9:     return (R^T, Q)
10: end procedure
```

Theorem 3. *Let $G \in \mathrm{GL}_n(\mathbb{Z})$ be a symmetric (positive) definite matrix and μ the $\mu D \mu^T$ factorisation of $\mathbb{G}$. Let $\lambda_{0,0} = 1$ and $\lambda_{k,k} = \det \mathbb{G}_k$ where $\mathbb{G}_k$ is the top left $k \times k$ block of $\mathbb{G}$. Let $\lambda = \mu \cdot \mathrm{diag}(\lambda_{1,1}, \ldots, \lambda_{n,n})$ and $U = (\mu^T)^{-1} \cdot \mathrm{diag}(\lambda_{0,0}, \ldots, \lambda_{n-1,n-1})$. Then the following holds:*

$$\lambda \in \mathrm{GL}_n(\mathbb{Z}), \quad \mathbb{U} \in \mathrm{GL}_n(\mathbb{Z}), \quad \lambda = \mathbb{G}\mathbb{U}.$$

From the Theorem 3 follows the dual and symplectic version of the GSO.

4.3 Symplectic Gram-Schmidt

Symplectic version of the GSO devised in the paper of Gama [5] is specifically made for the q-symplectic matrices.

Let $\mathbb{B}$ be a q-symplectic basis. Then $\mathbb{L}$ in LQ decomposition is also q-symplectic, and μ matrix corresponding to the GSO of $\mathbb{B}$ is also q-symplectic. Let $\mathbb{U}$ be a dual integer matrix. Let $\mathbb{R}_n$ be a reversed identity matrix, that is a matrix where the rows (or columns) are in reverse order, that is, on (i, j)-th position is the Kronecker symbol $\delta_{i,n+1-j}$. and let denote quadrants of matrices μ and λ as follows

$$\lambda = \begin{pmatrix} \lambda_1 & 0 \\ \lambda_2 & \lambda_3 \end{pmatrix}, \quad \mu = \begin{pmatrix} \mu_1 & 0 \\ \mu_2 & \mu_3 \end{pmatrix}, \quad \text{and} \quad U = \begin{pmatrix} U_1 & 0 \\ U_2 & U_3 \end{pmatrix}$$

We have $U_1 = \mathbb{R}_n \lambda_3 \mathbb{R}_n$ for symplectic matrices and the following relation for q-symplectic matrices $U_1 = \mathrm{diag}(q^2, q^4, \ldots, q^{2n})$

This algorithm reduces the number of multiplications of the Dual GSO by half.

Algorithm 9 Symplectic Gram-Schmidt algorithm

procedure SymplecticGram-Schmidt($\mathbf{b}_1, \ldots, \mathbf{b}_{2n} \in \mathbb{Z}^{2n}$)
  for $i = 1$ to n **do**
    precompute q^{2i}
  end for
  for $i = 1$ to n **do**
    $U_i \leftarrow \lambda_{i-1,i-1}$ (for all k, U_k represents $U_{k,i}$)
    $U_{i-1} \leftarrow -\lambda_{i,i-1}$
    for $j = i - 2$ down to 1 **do**
      $U_j = -\left(\sum_{k=j+1}^{i} \lambda_{k,j} U_k\right) / \lambda_{j,j}$
    end for
    for $j = 1$ to i **do**
      $\lambda_{2n+1-j,2n+1-i} \leftarrow q^{2(n+1-i)} U_j$
    end for
    for $j = i$ to $2n$ **do**
      $\lambda_{j,i} = \sum_{k=1}^{i} G_{j,k} U_k$
    end for
  end for
  return λ
end procedure

This version of GSO runs in $O(n^5 \log^2 B)$ where B is an upper bound of $\|\mathbf{b}_i\|$

4.4 Dual Gram-Schmidt

This and the symplectic version of GSO are the work of Gama, et al. [5], both dual and symplectic versions are devised to minimise the number of the divisions by getting around using Gram matrix and dual lattice.

Let $\mathbb{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ be a basis and $\mathbb{G} = \mathbb{B}\mathbb{B}^T$ its Gram matrix. If μ is a GSO of $\mathbb{B}$ then $\mathbb{G} = \mu D \mu^T$ where D is positive diagonal matrix. Rewriting the previous equation as $\mu = \mathbb{G}(\mu^T)^{-1}D^{-1}$ we can compute the $k \times k$ topleft triangle of $(\mu^T)^{-1}D^{-1}$. Matrix $(\mu^T)^{-1}$ is the GSO of the dual basis $\mathbb{B} = (\mathbf{b}_d, \ldots, \mathbf{b}_1)$.

Algorithm 10 Dual Gram-Schmidt algorithm

procedure DualGram-Schmidt($\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{Z}^n$)
  for $i = 1$ upto n **do**
    $U_i \leftarrow \lambda_{i-1,i-1}$ (for all k, U_k represents $U_{k,i}$)
    $U_{i-1} \leftarrow -\lambda_{i,i-1}$
    for $j = i - 2$ downto 1 **do**
      $U_j = -\left(\sum_{k=j+1}^{i} \lambda_{k,j} U_k\right) / \lambda_{j,j}$
    end for
    for $j = i$ upto n **do**
      $\lambda_{j,i} = \sum_{k=1}^{i} \langle \mathbf{b}_j, \mathbf{b}_k \rangle U_k$
    end for
  end for
  return λ
end procedure

Algortihm 10 reduces the number of divisions against the Algorithm 2. This version includes $n^2/2$ divisions whereas integral Gram-Schmidt $\Theta(n^3)$.

4.5 Symplectic $\mu D \mu^T$, Symplectic Cholesky

It was proven in the paper [5] that both $\mu D \mu^T$ factorisation and Cholesky decomposition preserve the symplecticity of a matrix, however, in this thesis their implementation is no different from their classical counterparts. Thus for implementation details follow Algorithm 5 and Algorithm 4.

Chapter 5

Implementation

In this section we will describe the implementation and testing facilities used within this thesis.

5.1 Programming language and libraries

- **C++**

  We have decided on the native implementation in C++ with C++11 support enabled, which allows a wide variety of low-level optimisations. It was also chosen for the reason that it is not transpiled or interpreted language, which could negatively skew the results.

- **Eigen**

  Eigen[1] is a very popular library used for matrix manipulation and linear algebra. It's used in TensorFlow[2] library from Google used for machine learning. It's entirely written in C++ with heavy use of templating system which allows compiler to optimise the written code during the compilation stage.

  Also it has a builtin versions of algorithms and matrix manipulations which are explicitly vectorised - the library itself tries to vectorise as much as possible if such option is enabled.

- **MPFR**

  Because of the high dimension lattices and the operations with large vectors we will use MPFR [3] – Multiprecision floating computations library. It relies on the GMP library internally. This library was chosen due to simple, already existing binding for Eigen library.

[1] `http://eigen.tuxfamily.org`
[2] `https://www.tensorflow.org`
[3] `http://www.mpfr.org/`

5.2 Generating random NTRU lattices

In order to test the variety of algorithms it's necessary to precompute different NTRU matrices. However for our testing purposes we decided on randomly generating public key using `std::random_device` in conjunction with `std::uniform_int_distribution` generating random NTRU lattices which had coefficients randomly and uniformly distributed.

We decided on NTRU parameters (N, q, p) as follows $q = 128, p = 3$ and N being primes from 5 to 103. We also generated multiple variants of NTRU lattices in the same dimension, to get better results during testing.

All NTRU matrices are generated and stored in a separate text files.

Listing 5.1: A random public key generator

```
std::vector<int> randomPublicKey(int N, int Q)
{
    std::vector<int> result(N);
    std::uniform_int_distribution<>
                    distribution(0, Q - 1);
    std::random_device rd;
    std::default_random_engine engine(rd());

    for (int i = 0; i < N; ++i) {
        result[i] = distribution(engine);
    }
    return result;
}
```

We shall describe the code listed above. The `std::random_device rd` is our source of entropy, it is platform-dependant where and what the source of the entropy is. In computers it's mostly some mathematical operation (most likely a xor) of many hardware and software counters.

The `std::uniform_int_distribution` serves for our random engine is sort of a filter, however it's not filtering, but distributing integers across the interval $(0, Q - 1)$.

The `std::default_random_engine` is an algorithm, or more precisely, a generator which implementation is also platform-dependent and generates the random output based on the entropy it's given.

Using this algorithm we are able to generate random polynomials (represented as vectors in the code) in $(\mathbb{Z}/q\mathbb{Z})[x]/(x^N - 1)$.

5.3 Testing

Testing was done on a linux-based machine with Intel Core i5-6560U (2 cores, 4 threads) with base frequency of 1.8 GHz (2.8 GHz with TurboBoost tech-

nology) equipped with 8 GB of RAM and Intel SSD 500p series.

Chapter 6

Results

In this chapter we will cover the results of the experimental part.The following results were achieved by running iterations over different matrices within the same dimension and the results are averaged.

In the paper of Gama [5] they included tests on 128-bit long random integer coefficients in a rather large matrices ($N \approx 1000$, remember matrices are $(2N)$-dimensional) thus all our measurements on a small matrices ($N \approx 100$) resulted in quite surprising result – new algorithms mentioned in the research paper are not suitable for low-dimensional sized NTRU lattices except for Integral Gram-Schmidt as the whole purpose of the Dual and Symplectic GSO is to prevent the multiplication and division of very large integers, which involves other complex algorithms.

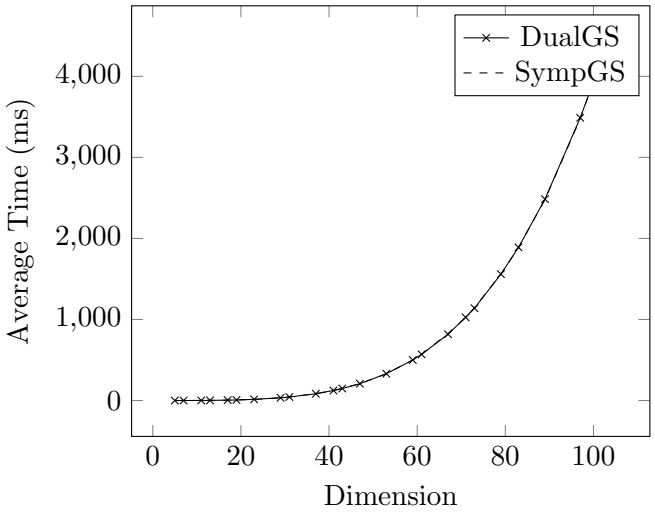


Figure 6.1: Comparison of Gram-Schmidt family

We can clearly see from Figure 6.1 that both Symplectic and Dual Gram-Schmidt are quite slow and are averaging on 4 seconds at lattices around

dimension of 100. Both algorithms are resulting in $O(n^5 \log^2 B)$.

However, from the Figure 6.2 we can clearly see that Integral GSO, which has time complexity of $O(mn^2 \log^2(B))$ is faster than the original one with time complexity $O(mn^4 \log^2(B))$.

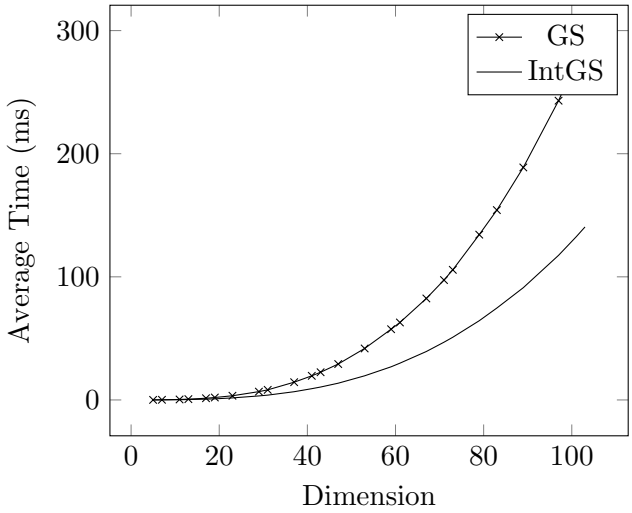


Figure 6.2: Comparison of Integral Gram-Schmidt and Gram-Schmidt

In the Figure 6.3 we can see a comparison between Integral GSO and Cholesky decompostion. Cholesky decomposition has $O(mn^2)$ time complexity and thus it is slightly faster than Integral GSO.

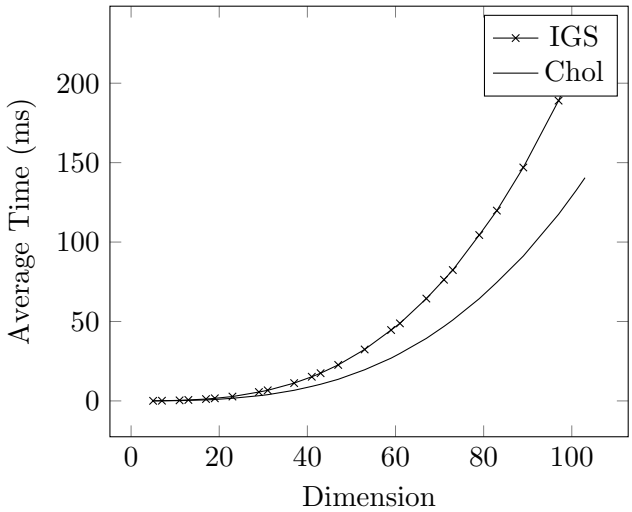


Figure 6.3: Comparison of Cholesky decomposition and Integral Gram-Schmidt

Conclusion

The main aims of this thesis were mostly theoretical, extending and broadening the knowledge of linear algebra and post-quantum cryptography on a lattice-based cryptosystem, in this regard we was very successful.

We laid out the mathematics behind the lattices, defined two $\mathcal{NP}$-hard problems and described the NTRU cryptosystem. We connected orthogonalization methods with the problems which are deeply related.

We developed the understanding between many new algorithms from linear algebra was quite challenging, but also very interesting topic, we have learnt where all the algorithms do find a usage.

Some of the orthgonalization algorithms mentioned in the paper were not improved at all, but they have been proven to maintain symplecticity, thus even though they are implemented, they bring no benefit at all.

This thesis may serve as an advancement of the research paper that it's based on.

Future research

It may be useful to highlight where the future work may go. I suggest the following:

- Measure more extensively algorithms with different size of coefficients – namely the size of the lattice and the q – modulus in a lattice.
- Exploring other implementations of QR factorisation and proving, or disproving their symplecticity.
- Exploring other implementation possibilities of the Cholesky decomposition.
- Explore newer, or other versions of lattice-based cryptosystem.

Bibliography

[1] Shor, P. W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *eprint arXiv:quant-ph/9508027*, 1995, quant-ph/9508027.

[2] Galbraith, S. D.: *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012, ISBN 1107013925.

[3] Golub, G. H.; Van Loan, C. F.: *Matrix Computations*. JHU Press, third edition, 1996, ISBN 9780801854149.

[4] Hoffstein, J.; Pipher, J.; Silverman, J.: *An Introduction to mathematical cryptography*. Springer-Verlag New York, first edition, 2008, ISBN 978-0-387-77994-2, dOI: 10.1007/978-0-387-77993-5.

[5] Gama, N.; Howgrave-Graham, N.; Nguyen, P. Q.: Symplectic Lattice Reduction and NTRU. *Advances in Cryptology - EUROCRYPT 2006*, 2006: p. 233–253.

[6] Computer Security Resource Center NIST: Post-Quantum Cryptography Round 1 Submissions. http://web.archive.org/web/20180408151759/https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions, accessed: 2018-04-08.

[7] Hoffstein, J.; Pipher, J.; Silverman, J. H.: NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory*, e. J. P. Buhler, Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, ISBN 978-3-540-69113-6, p. 267–288.

[8] On Board Security: NTRU PKCS Tutorial. https://assets.onboardsecurity.com/static/downloads/NTRU/resources/NTRU-PKCS-Tutorial.pdf, accessed: 2018-05-01.

Appendix A

Acronyms

CSRC Computer Science Resource Center

NIST National Institute of Standards and Technology

GSO Gram-Schmidt Orthogonalization

Appendix B

Manual

B.0.1 Requirements

Because this implementation is based on a GMP library, it's necessary to have a linux based operating system.

The list of supported algorithms by the program and the associated parameter (argument) for the command line execution.

- GramSchmidt (1),
- IntegralGramSchmidt (2),
- QR/LQ (3),
- Cholesky (4),
- LDL ($\mu D \mu^T$) (5),
- DualGramSchmidt (6),
- SymplecticGramSchmidt (7),
- SymplecticQR (8),
- SymplecticLDL (9),
- SymplecticCholesky (10)

In order to build the executable from scratch it's necessary to have installed these dependencies

- `libgmp-dev`
- `libmpfr-dev`
- CMake version 3. or higher.

Appendix C

Contents of enclosed SD

- `readme.txt` ..... Instructions how to compile the sources
- `exe` ..... the directory with executables
  - `data` ..... datasets
  - `tester` ..... Linux executable
- `src` ..... source codes
  - `code` ..... code of the implemented algorithms
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format