Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Emil Haagensen Garli

Instantiating the GPV-framework

Bachelor's thesis in Mathemathical Sciences
Supervisor: Kristian Gjøsteen
June 2022

NTNU
Norwegian University of
Science and Technology

Emil Haagensen Garli

Instantiating the GPV-framework

Bachelor's thesis in Mathemathical Sciences
Supervisor: Kristian Gjøsteen
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Instantiating the GPV-framework

Emil Haagensen Garli

June 27, 2022

Contents

1 Introduction **2**
1.1 Notation 2

2 NTRU **2**
2.1 Preliminaries 3
2.2 Design 3
2.3 Choice of parameters 6
2.4 Attacks on NTRU 8
2.5 NTRU Lattice 9
2.5.1 Lattice problems 11

3 GPV-Framework **13**
3.1 Instantiation 14
3.1.1 GGH and NTRUSign 15
3.1.2 Generating $\boldsymbol{v}$ using Klein's algorithm 15
3.2 Verification 18

References **18**

1 Introduction

Most of the cryptosystems used today bases its security on one of three mathematically hard problems: the factorization of large integers, computing a discrete logarithm, or computing an elliptic curve discrete logarithm. While these problems are hard for classical computers, they have been shown to be easily solved using a quantum computer. Since quantum computing is still in its early stages, there is no immediate threat to our current cryptosystems, but post quantum secure candidates to replace them has already been created.

One of the top candidates("NIST", 2020) to replace current cryptosystems is NTRU, which is not known to be vulnerable to attacks on quantum computers. NTRU is an asymmetric cryptosystem, which means that it has pairs of keys, one public and one private to encrypt and decrypt data.

The GPV (Gentry, Peikert, Vaikuntanathan) (Peikert, 2008) framework is a description of how to generate secure signatures from a private and a public key represented as bases for a lattice space in matrix form. When a ciphertext is sent, the recipient needs a way to ensure the identity of the sender. This is done by creating a signature s. We want the signature to be as short as possible to minimize the time it takes to create it, but not too short, or an attacker might be able to guess a valid signature.

In this paper we will describe how NTRU public and private keys are created and how they are used to encrypt and decrypt messages. We will then look at how they are instantiated in the GPV framework, which will use the NTRU keys to generate signatures for encrypted messages.

1.1 Notation

- $\mathcal{O}$ denotes the big-o notation, which is the worst case run time of an algorithm.
- Ω denotes the big-omega notation, unless stated otherwise, which is the best case run time of an algorithm.
- The spectral norm of $\boldsymbol{B}$ is denoted $||\boldsymbol{B}||_2 = (\text{maximum eigenvalue of } \boldsymbol{B}^*\boldsymbol{B})^{1/2}$, where $\boldsymbol{B}^*$ is the conjugate transpose of $\mathbf{B}$.

2 NTRU

NTRU is an asymmetric cryptosystem that is based on the NP-hard SVP (Shortest Vector Problem) and CVP (Closest Vector Problem) in lattices. In

contrast to the current RSA cryptosystem that is based on the hardness of solving factorization problems, NTRU has keys built on polynomials translated to lattices.

In this chapter we will describe how to create public and private keys and how to encrypt and decrypt with them. We will also look at some ways to attack this encryption scheme.

2.1 Preliminaries

A lattice can be described as a free abelian group over a vector space $\mathbb{R}^n$, such that the dimension of the lattice is n, and that coordinate wise addition and subtraction of lattice points is still a point in the lattice. Consider the basis $B = \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right)$ in $\mathbb{Z}^2$ which generates a lattice over $\mathbb{R}^2$. The lattice is now an infinite grid over $\mathbb{R}^2$ with points on every integer. When creating an NTRU lattice, the basis is created from the coefficients of randomly generated polynomials, with several hundred dimensions.

When we combine polynomials in a ring R, we use convolution, denoted with $*$.

Definition 1 *The convolution of two polynomials $a(x)$ and $b(x)$ in a polynomial ring is defined as*

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

where a_i and b_j are the coefficients of $a(x)$ and $b(x)$, and $N-1$ is the largest degree of any variable in the polynomials.

Definition 2 *For a positive integer N, the ring of convolution polynomials is the quotient ring*

$$R = \mathbb{Z}[x]/(x^N - 1)$$

such that all polynomials in the ring has integer coefficients and no variable of degree larger than N-1.

2.2 Design

For encryption of a message m to begin, a public and private key will have to be generated. These keys are generated by the public parameters

$$(p, q, N, d) \in \mathbb{Z}^+$$

and a Lift operator, that takes a polynomial $a \in R_q$ or R_p and center-lifts it to a polynomial in R. When choosing the public parameters, we ensure that $gcd(N, q) = 1$ and $gcd(p, q) = 1$. The proof for the choice of parameters will follow in the next section. $N - 1$ is the maximum degree of any polynomial used. The co-prime parameters p and q are used to generate the convolution polynomial rings R_p, R_q such that

$$R_p = \mathbb{Z}_p[x]/(x^N - 1)$$

and

$$R_q = \mathbb{Z}_q[x]/(x^N - 1)$$

are $R = \mathbb{Z}[x]/(x^N - 1)$ reduced by modulo p and modulo q respectively.

The parameter d is used to describe the coefficients of the polynomials f and g. The operator $T(d_1, d_2)$ calls an algorithm that outputs a random polynomial $a(x) \in R$ with d_1 coefficients equal to 1, d_2 coefficients equal to -1, and the rest equal to 0. If T has only one input, the number of coefficients equal to 1 and -1 are equal.

We can now make two polynomials

$$f \in T(d + 1, d) \text{ and } g \in T(d, d)$$

where f will be used to calculate its inverses

$$F_p(x) = f^{-1}(x) \in R_p$$

and

$$F_q(x) = f^{-1}(x) \in R_q.$$

When these operations are done, the public key used for encryption is

$$h(x) = F_q(x) * g(x) \in R_q.$$

If f has been chosen so that no inverse exists, the process is restarted with new polynomials. The plaintext to be encrypted is a polynomial

$$m(x) \in R$$

with coefficients that satisfies $(-\frac{1}{2}p < m_i < \frac{1}{2}p)$. To encrypt a message, we add $m(x)$ to the recipients public key multiplied by a random polynomial, that is used to generate noise in the ciphertext

$$e(x) \equiv p \cdot h(x) * r(x) + m(x) \in R_q,$$

where $r(x)$ is a random polynomial generated from $T(d, d)$.

When the ciphertext is recieved, the recipient can use their private polynomial $f(x)$ to compute a new polynomial

$$a(x) \equiv f(x) * e(x) \mod q$$

and then use the Lift operator in the public parameters to take $a(x)$ out of R_q into R, and then take the resulting polynomial into R_p. This new polynomial is

$$b(x) \equiv F_p(x) * a(x) \mod p$$

which is equal to the plain text $m(x)$, given that the parameters have been chosen such that $q > (6d + 1)p$. This is because $a(x)$ can be written as

$$\begin{aligned} a(x) &\equiv f(x) * e(x) \mod q \\ &\equiv f(x) * (p \cdot h(x) * r(x) + m(x)) \mod q \\ &\equiv p \cdot f(x) * F_q(x) * g(x) * r(x) + f(x) * m(x) \mod q \\ &\equiv p \cdot g(x) * r(x) + f(x) * m(x) \mod q \end{aligned}$$

If we now ensure that the coefficients of the polynomial $a(x)$, are all less than q, so that when $a(x)$ is lifted to R, no information is lost from having it computed modulo q, we get the plaintext $m(x)$.

For the polynomials $g(x)$ and $r(x)$, both in $T(d, d)$, the worst case scenario is if the 1 and -1 coefficients match up. This would lead to the largest coefficient in $g(x) * r(x)$ being $2d$. Now consider $f(x) \in T(d + 1, d)$ and the plaintext $m(x)$ whose coefficients m_n are

$$-\frac{1}{2}p < m_n < \frac{1}{2}p.$$

The largest possible coefficient of the product $f(x) * m(x)$ is $(2d + 1) \cdot \frac{1}{2}p$. Now we can look at the equation

$$a(x) \equiv p \cdot g(x) * r(x) + f(x) * m(x) \mod q.$$

The largest possible coefficient for $a(x)$ is

$$p \cdot 2d + (2d + 1)\frac{1}{2}p = (3d + \frac{1}{2})p = \frac{1}{2}(6d + 1)p,$$

which means that any coefficient of $a(x)$ is smaller than $\frac{1}{2}q$, given that p, q are chosen such that $q > (6d + 1)p$. This ensures that the polynomial $a(x)$ mod $q = a(x)$ in R.

The plaintext $m(x)$ can now be recovered:

$$\begin{aligned} b(x) &\equiv F_p(x) * a(x) \mod p \\ &\equiv F_p(x) * f(x) * e(x) \mod p \\ &\equiv F_p(x) * f(x) * (p \cdot h(x) * r(x) + m(x)) \mod p \\ &\equiv m(x) \mod p \end{aligned}$$

Because this equation is reduced $\mod p$, the product $p \cdot h(x) * r(x)$ becomes 0 and $F_p(x) * f(x) = 1 \equiv m(x) \mod p$. The following theorem is now proven.

Theorem 1 *Let $f \in T(d+1, d)$ and $g \in T(d, d)$. The polynomials f, g and their inverses modulo p and modulo q can be used to generate public and private keys that can be used to encrypt and decrypt a message.*

The proofs for the valid choice of parameters *f,g,N,p,q* will follow in the next section.

2.3 Choice of parameters

The way the NTRU-parameters are chosen is crucial for the speed and security of the algorithm. The polynomials f and g are generated randomly by using an algorithm called NTRUGen, which outputs *f,g,F,G*. A public key can only be generated from f and g iff f is invertible modulo q, which is true iff NTT(f) has no coefficient equal to zero. To give a definition of NTT, we first need to define the Fast Fourier Transform.

Definition 3 *The Fast Fourier Transform (FFT): Let $f \in \mathbb{Q}[x]/(\phi)$. Let Ω_ϕ be the set of complex roots of ϕ, $\zeta \in \Omega_\phi$, and suppose the polynomial ϕ is monic with distinct roots over $\mathbb{C}$, so that $\phi(x) = \Pi_{\zeta \in \Omega_\phi}$. The Fast Fourier transform of f with respect to ϕ is:*

$$FFT_\phi(f) = (f(\zeta))_{\zeta \in \Omega_\phi}$$

When ϕ is clear from context we note $FFT(f)$.

Definition 4 *NTT (Number Theoretic Transform) is the analog of the FFT in the field $\mathbb{Z}_p$, where p is a prime such that $p \equiv 1 \mod 2n$. The polynomial ϕ now has exactly n roots (ω_i) over $\mathbb{Z}_p$. We can now represent any polynomial $f \in \mathbb{Z}_p/(\phi)$ as $f(\omega_i)$.*

Having a polynomial represented in its NTT form allows for faster computing of binary operations addition, subtraction, multiplication and division, because they can be done element wise.

When we generate the parameters p and q, we require that $\gcd(p, q) = 1$, or an attacker can easily get the plain text from the cipher text $e(x)$.

Proof: Proof by contradiction:

case 1: p=q:

If p=q then

$$R_p = \frac{\mathbb{Z}/p\mathbb{Z}[x]}{x^N - 1} = R_q = \frac{\mathbb{Z}/q\mathbb{Z}[x]}{x^N - 1}$$

the cipher text

$$\begin{aligned} e(x) &\equiv p \cdot h(x) * r(x) + m(x) \mod q \\ &\equiv m(x) \end{aligned}$$

since $p = q$ and $p \mod p \equiv 0$, $e(x) \equiv m(x) \mod q$

Case 2: $p|q$

$$e(x) \equiv p \cdot h(x) * r(x) + m(x) \mod q$$

Apply the Lift-function:

$$\begin{aligned} \text{Lift}(e(x)) \equiv & p \cdot h(x) * r(x) + m(x) + n \cdot q \\ & n \cdot q \mod p \equiv 0 \\ & p \cdot h(x) * r(x) \mod p = 0 \\ & \text{Lift}(e(x)) \mod p \equiv m(x) \end{aligned}$$

We have now proved that p=q and $p|q$ breaks the encryption, so to achieve any security, we should choose p,q such that gcd(p,q)=1. ■

We also have the requirement that $gcd(q, N) = 1$. If $q|N$, then the lattice problems are reducible from $\mathbb{Z}_q/(X^N - 1)$ down to $\mathbb{Z}_q/(X^q - 1)$. When solving lattice problems, they are easier to do in a smaller lattice, so if we can reduce a lattice to a smaller lattice, equivalent problems become easier to solve.

When generating the polynomial f, the parameters of T cannot be (d, d), or f will never have an inverse. Proof:

$$\text{Let } f(x) \in T(d, d)$$

Now, since f has equal amounts of coefficients equal equal to 1 as -1,

$$f(1) = \sum f_i = 0$$

Assume, for the sake of contradiction, that $f^{-1}(x) = b(x)$

$$\begin{aligned} f(x) * b(x) &\equiv 1 \pmod{x^n - 1} \\ \Rightarrow f(x) * b(x) + c(x)(X^n - 1) &= 1 \end{aligned}$$

Now, let $x = 1$. This gives us

$$\begin{aligned} f(1) * b(1) + c(1)(1^n - 1) &= 1 \\ 0 &= 1 \end{aligned}$$

This is a contradiction, hence we can conclude that we need at least one more coefficient equal to 1 than -1 such that $f \in T(d+1, d)$ for f to have an inverse.■

2.4 Attacks on NTRU

When the public key $h(x)$ has been made public, there is a hidden relationship

$$f(x) * h(x) \equiv g(x) \mod q.$$

An attacker can now try to break the NTRU encryption system by solving the NTRU key recovery problem.

The NTRU Key Recovery Problem: Given $h(x)$, find ternary polynomials $f(x)$ and $g(x)$ satisfying $f(x) * h(x) \equiv g(x) \mod q$

Definition 5 *A ternary polynomial has coefficients equal to 1, 0 or -1.*

While one solution to this problem is the polynomials $(f(x), g(x))$, the rotations of these polynomials are also solutions to the problem, because the coefficients of $f(x)$ and $g(x)$ have been rotated by k positions.

$$(x^k \cdot f(x), x^k \cdot g(x)), \text{ for } 0 \leq k < N.$$

A solution that is rotated yields the rotated plain text $x^k \cdot m(x)$. If an attacker tries to extract the private key from solving the NTRU key recovery problem by brute force, they will have to check every polynomial in $T(d+1, d)$, and since all rotations of the solution is also a solution, the number of solutions are given by $\#T(d+1, d)/N$. The size of the set $T(d_1, d_2)$ is given by choosing d_1 coefficients to be 1, and d_2 of the remaining $N - d_1$ coefficients to be -1, as follows:

$$\#T(d_1, d_2) = \binom{N}{d_1}\binom{N - d_1}{d_2}.$$

We now chose the NTRU public parameters (N, p, q, d) to be $(509, 3, 1024, 56)$, so that $(6d-1)\cdot 3 = 1005 < q = 1024$. The number of possible solutions for the NTRU key recovery problem is

$$\frac{1}{1024}\binom{1024}{57}\binom{968}{56} \approx 8.25^{185}.$$

2.5 NTRU Lattice

The public key $h(x)$, and the parameter q can be used to express NTRUEncrypt as a lattice cryptosystem, where $h(x)$ is a polynomial of degree $N-1$ and can be written as

$$h(x) = h_0 + h_1 x + ... + h_{N-1}x^{N-1}.$$

We can now use $h(x)$ and q to express the NTRU lattice L_h^{NTRU} in terms of linearly independent basis vectors in the 2N-dimensional matrix

$$M_h^{NTRU} = \begin{bmatrix} I & \mathbf{h} \\ 0 & qI \end{bmatrix},$$

where $\mathbf{h}$ is all cyclical permutations of the coefficients of the public key $h(x)$. The private key is the matrix

$$P^{\text{NTRU}} = \begin{bmatrix} \mathbf{f} & \mathbf{g} \\ \mathbf{F} & \mathbf{G} \end{bmatrix},$$

where the entries in the matrix are all cyclical permutations of the polynomials f, g, F and G and has to satisfy the NTRU equation

$$fG - gF = q \mod x^N - 1. \tag{1}$$

While the public key matrix M_h^{NTRU} and the private key matrix generate the same lattice, P^{NTRU} consists of small polynomials, which allows for solving lattice problems for the lattice generated from these matrices. When M_h^{NTRU} is made public, an attacker could theoretically extract

$$h(x) = F_q(x) * g(x) \in R_q$$

to find F_q and $g(x)$. The hardness of computing small polynomials f', g' such that $h = g' * (f')^{-1}$ constitutes the inversion version of the NTRU assumption. The decisional version of the NTRU assumption states that h is indistinguishable from $g' * (f')^{-1}$ or a uniformly random polynomial in the same ring.

The lattice L_h^{NTRU} can be viewed as a a grid with vertices in $\mathbb{Z}^n$ over $\mathbb{R}^n$. The fundamental domain $\mathcal{F}$ of L_h^{NTRU} is the volume of the space within the grid.

Definition 6 *For a lattice L, the fundamental domain $\mathcal{F}$ of L is n-dimensional and is called the determinant of L, denoted det(L).*

In figure 1, $\mathcal{F}$ is represented as the shaded parallelogram in between the vectors.

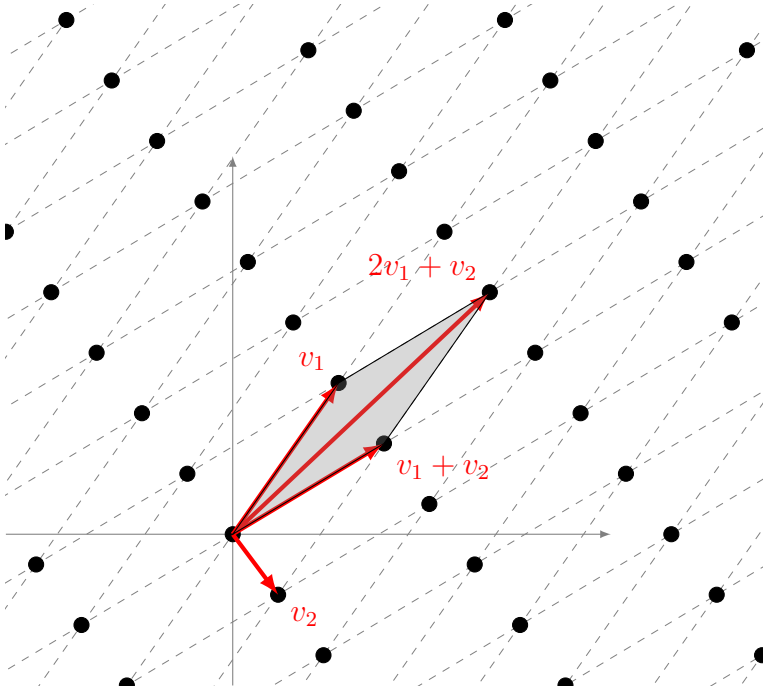


Figure 1: A simple illustration of a 2-D lattice space

This volume can be calculated by

$$\det(L_h^{NTRU}) = \text{Vol}\{t_1v_1 + t_2v_2 + ... + t_nv_n \mid 0 \leq t_i < 1\}$$

2.5.1 Lattice problems

The L_h^{NTRU} lattice can be used as a basis for a cryptographic algorithm because of a collection of NP-hard problems called the *Shortest vector problem* and the *Closest vector problem.*

Definition 7 *Shortest Vector Problem (SVP): Finding the shortest non-zero vector in the lattice $\mathcal{L}$, denoted $\lambda(\mathcal{L}) = \min_{v \in \mathcal{L}/\{0\}} ||v||_N$.*

Definition 8 *Closest Vector Problem (CVP): Finding the closest vector $v \in \mathcal{L}$ to a given vector $w \notin \mathcal{L}$*

When the long public basis is made public, it is possible for an adversary to reduce the vectors in this basis using the LLL-algorithm(Deng, 2016). It is a way to reduce a basis for a lattice to more easily solve SVP. The algorithm works by orthogonalizing a basis $\mathbf{B}$ to a basis $\boldsymbol{B}^\dagger$ for a lattice $\mathbf{L}$ and reducing it.

Definition 9 *(Gram-Schmidt Orthogonalization method) Let $\boldsymbol{b_n} = \{\boldsymbol{b_1}, \boldsymbol{b_2}, ..., \boldsymbol{b_i}\}$ be a basis for a subspace $S_i \in \mathbb{R}^n$. The orthogonal basis $\boldsymbol{b_n^\dagger} = \{\boldsymbol{b_1^\dagger}, \boldsymbol{b_2^\dagger}, ..., \boldsymbol{b_n^\dagger}\}$ is defined:*

$$\boldsymbol{b_1^\dagger} = \boldsymbol{b_1}$$

$$\boldsymbol{b_2^\dagger} = \boldsymbol{b_2} - \frac{\boldsymbol{b_2 b_1^\dagger}}{\|\boldsymbol{b_1^\dagger}\|^{\boldsymbol{2}}} \boldsymbol{b_1}$$

$$\vdots$$

$$\boldsymbol{b_n^\dagger} = \boldsymbol{b_i} - \boldsymbol{\Sigma_{i<n}} \frac{\boldsymbol{b_i b_n^\dagger}}{\|\boldsymbol{b_n^\dagger}\|^{\boldsymbol{2}}} \boldsymbol{b_i}$$

Definition 10 *Let $\boldsymbol{b_n} = \{\boldsymbol{b_1}, \boldsymbol{b_2}, ..., \boldsymbol{b_n}\}$ be a basis for a n-dimensional lattice $\boldsymbol{L}$ and let $\boldsymbol{b_n^\dagger} = \{\boldsymbol{b_1^\dagger}, \boldsymbol{b_2^\dagger}, ..., \boldsymbol{b_n^\dagger}\}$ be the Gram-Schmidt orthogonalized basis for $\boldsymbol{L}$. Let*

$$u_{i,k} = \frac{\boldsymbol{b_k b_i}}{\boldsymbol{b_i^\dagger b_i^\dagger}}$$

$\boldsymbol{b_n}$ is a LLL reduced basis if two conditions are met:

1. *$\forall i \neq k, u_{i,k} \leq \frac{1}{2}$*
2. *For each $i, \|\boldsymbol{b_{i+1}^\dagger} + u_{i,i+1}\boldsymbol{b_i^\dagger}\|^2 \geq \frac{3}{4}\|\boldsymbol{b_i^\dagger}\|^2$*

Algorithm 1 LLL algorithm

Input: $\{b_1, b_2, \ldots, b_n\}$

Step 1: Gram-Schmidt orthogonalization

for $i = 1$ to n **do**

  for $k = i - 1$ to 1 **do**

    $m \leftarrow$ nearest integer of $u_{k,i}$

    $\boldsymbol{b_i} \leftarrow \boldsymbol{b_i} - m\boldsymbol{b_k}$

Step 2: Check Condition 2, and swap

for $i = 1$ to $n - 1$ **do**

  if $||\boldsymbol{b}^{\dagger}_{\boldsymbol{i+1}} + u_{i,i+1}\boldsymbol{b}^{\dagger}_{\boldsymbol{i}}||^2 < \frac{3}{4}||\boldsymbol{b}^{\dagger}_{\boldsymbol{i}}||^2$ **then**

    swap $\boldsymbol{b_{i+1}}$ and $\boldsymbol{b_i}$

    go to step 1

Being able to reduce a lattice gives a potential attacker more efficient ways of breaking the system. If the LLL-algorithm manages to reduce a basis, the resulting basis is more orthogonal than the original. To measure the orthogonality of a basis we use the Hadamard ratio.

Definition 11 *The Hadamard ratio* $0 \leq \mathcal{H}(b) \leq 1$ *of a basis* $\boldsymbol{b} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots \boldsymbol{v}_n\}$ *is defined as*

$$\mathcal{H}(b) = \left(\frac{detL}{\|\boldsymbol{v}_1\|\|\boldsymbol{v}_2\| \cdots \|\boldsymbol{v}_n\|}\right)^{1/n}$$

where n is the dimension of the lattice generated from the basis $\boldsymbol{b}$*.*

The higher the Hadamard ratio, the more orthogonal the vectors.

For small dimension lattices, reducing the basis for the public key can reveal the coefficients in the polynomials used to create the secret key. To show this, we choose the NTRU public parameters $(N, p, q, d) = (7, 3, 41, 2)$, so that $41 = q > (6d + 1)p = 39$. We now choose our private polynomials f and g so that $f \in T(3, 2)$ and $g \in T(2, 2)$:

$$f(x) = x^6 - x^4 + x^3 + x^2 - 1 \text{ and } g(x) = x^6 + x^4 - x^2 - x$$

The public key is $h(x) = F_q(x) \cdot g(x)$ and the private key is $(f(x), F_p(x))$. Using $h(x)$ to create a basis for the NTRU lattice

$$M_h^{NTRU} = \begin{bmatrix} I & \boldsymbol{h} \\ 0 & qI \end{bmatrix}$$

we can apply the LLL-algorithm to reduce it to the basis M_{LLL}^{NTRU}, so that the vectors are smaller and to increase its Hadamard ratio. Calculating the Hadamard ratio of the two bases gives

$$\mathcal{H}(M_h^{NTRU}) = 0.1184 \text{ and } \mathcal{H}(M_{LLL}^{NTRU}) = 0.8574.$$

The smallest row vector in the reduced basis is

$$(1, 0, -1, 1, 0, -1, -1, -1, 0, -1, 0, 1, 1, 0),$$

which we now split in two, and use each piece as the coefficients for the polynomials f' and g' so that

$$\begin{aligned} f'(x) &= 1 - x^2 + x^3 - x^5 - x^6 \\ g'(x) &= -1 - x^2 + x^4 + x^5. \end{aligned}$$

These polynomials are not the private polynomials f and g chosen at the beginning, but simple rotations of them. Simply rotating f by $-x^3$ we get

$$-x^3 \cdot f(x) = -x^3 \cdot (x^6 - x^4 + x^3 + x^2 - 1) = -x^9 + x^7 - x^6 - x^5 + x^3$$

Since we are in the ring $\mathbb{Z}_q[x]/(x^N - 1)$ where $N = 7$,

$$-x^3 \cdot f(x) \equiv 1 - x^2 + x^3 - x^5 - x^6 = f'(x)$$

and doing the same with g, we get

$$\begin{aligned} -x^3 \cdot g(x) =& - x^3 \cdot (x^6 + x^4 - x^2 - x) = -x^9 - x^7 + x^5 + x^4 \\ \equiv& x^5 + x^4 - x^2 - 1 = g'(x) \end{aligned}$$

We have now successfully applied the LLL-algorithm to get the private polynomials used to create the private key, breaking the encryption. While the LLL-algorithm can easily reduce a basis with only $2N = 14$ dimensions, it is less efficient at reducing higher dimensional bases(Nguyen & Stehlé, 2009), such as $500 < N < 1000$ that is used in actual implementations of NTRU(Chen et al., 2019).

3 GPV-Framework

The GPV framework describes a way to obtain secure lattice based signatures. It is instantiated by a short (private) and a long (public) basis for a lattice space, and an algorithm that can compute short signatures based on the information in the short basis. There are many available choices for both lattices and signature generation algorithms when instantiating the GPV framework, but in this text we will focus primarily on NTRU lattices and two different choices for signature algorithms.

3.1 Instantiation

Using the NTRU-lattices generated by the public and private key polynomials $h(x)$ and $(f(x), F_p(x))$, we can instantiate the GPV framework with its public basis

$$\mathbf{A} = \begin{bmatrix} I & \mathbf{h} \\ 0 & qI \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$$

and the private basis

$$\mathbf{B} = \begin{bmatrix} \mathbf{f} & \mathbf{g} \\ \mathbf{F} & \mathbf{G} \end{bmatrix} \in \mathbb{Z}_q^{m \times m},$$

where $m > n$. These matrices are orthogonal, such that $(\forall a \in \mathbf{A} \exists b \in \mathbf{B} | < a, b >= 0)$, so that the lattice generated from $\mathbf{B}$ is orthogonal to the lattice generated from $\mathbf{A}$.

Sending an encrypted message requires a signature to verify the identity of the sender. When signing a message $\mathbf{m}$, the signature is a short value $\boldsymbol{s} \in \mathbb{Z}_q^m$. To compute this signature, a preimage $\boldsymbol{c}_0 \in \mathbb{Z}_q^m$ is computed which satisfies

$$\boldsymbol{c}_0 \boldsymbol{A}^t = H(\boldsymbol{m}),$$

where $H(m)$ is a hash function $H : \{0,1\}^* \to \mathbb{Z}_q^n$.
The private key

$$\mathbf{B} = \begin{bmatrix} \mathbf{f} & \mathbf{g} \\ \mathbf{F} & \mathbf{G} \end{bmatrix}$$

is now used to compute a vector $\boldsymbol{v} \in P^{NTRU}$ which is close to $\boldsymbol{c}_0$. The signature $\boldsymbol{s}$ is the difference $\boldsymbol{s} = \boldsymbol{c}_0 - \boldsymbol{v}$.

To compute the vector $\boldsymbol{v}$, a trapdoor sampler is used because they leak no information about the private key $\mathbf{B}$. A trapdoor is a function $f : D \to R$ where f can be efficiently computed, but the inverse $f^{-1} : R \to D$ is hard to compute unless an attacker can obtain some secret information about the trapdoor y, so that given $f(x)$ and y, one can easily compute x. The trapdoor function

$$f_{\mathbf{A}}(\boldsymbol{s}) = \mathbf{A}\mathbf{s} \mod q$$

where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, $m > n$, $\mathbf{s} \in \mathbb{Z}_q^m$, is many-to-one, which means that it is impossible to find an inverse of this function.

A trapdoor sampler outputs a short vector $\boldsymbol{v}$ such that $\boldsymbol{v}^t \boldsymbol{A} = \boldsymbol{c}_0$, given an input of a matrix $\mathbf{A}$, a trapdoor $\mathbf{T}$ and a target $\boldsymbol{c}_0$.

3.1.1 GGH and NTRUSign

GGH and NTRUSign are both public-key cryptosystems based on lattices, and NTRUSign can be viewed as a variant of GGH instantiated with NTRU lattices.

The challenging part about signature generation is the computation of the vector $\boldsymbol{v}$. While the high level description of computing $\boldsymbol{v}$ is the same, the deterministic algorithms of GGH and NTRUSign frameworks works by expressing $\boldsymbol{c}_0$ as a real linear combination of the rows of the private key $\mathbf{B}$, and then rounding these vectors coefficient wise before multiplying them by $\mathbf{B}$ again, using a round off algorithm:

$$\boldsymbol{v} \leftarrow \lfloor \boldsymbol{c}_0 \boldsymbol{B}^{-1} \rceil \boldsymbol{B}$$

When this procedure is done, $\boldsymbol{s} = \boldsymbol{v} - \boldsymbol{c}_0$ is guaranteed to be in the parallelepiped $[-\frac{1}{2}, \frac{1}{2}]^m \times \boldsymbol{B}$, which guarantees that the norm $||\boldsymbol{s}||$ has the same Ω and $\mathcal{O}$ which means that the norm of $\boldsymbol{s}$ is tightly bound. While GGH and NTRUSign both produce valid signatures $\boldsymbol{s}$, since $\boldsymbol{s}$ lies in $[-\frac{1}{2}, \frac{1}{2}]^m \times \boldsymbol{B}$, information about the secret basis $\mathbf{B}$ will gradually leak, making these signature generation procedures unsafe. This problem is known as *The Hidden Paralellepiped Problem* (Nguyen & Regev, 2006).

3.1.2 Generating $\boldsymbol{v}$ using Klein's algorithm

Klein's algorithm (Klein, 2000) is a randomized version of the round-off algorithm used in GGH and NTRUSign. It uses an input vector $\mathbf{x}$ and the vectors $\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n$ from the NTRU private key $\mathbf{B}$ to find a vector $\boldsymbol{y}$ in the lattice $L(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n)$ that minimizes the distance $|\boldsymbol{x} - \boldsymbol{y}|$. If we provide the input parameter $\boldsymbol{x} = \boldsymbol{c}_0$, and the vectors $\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n$ being the basis vectors for P^{NTRU}, this algorithm will, with high probability, return the closest lattice vector $\boldsymbol{y} = \boldsymbol{v} \in P^{NTRU}$ to our input $\boldsymbol{c}_0$.

The first part in describing Klein's algorithm is a procedure for rounding a rational number to an integer, called $randRound_c(r)$:

Algorithm 2 $randRound_c(r)$

$r = p + a$, $p \in \mathbb{Z}$, $0 \leq a \leq 1$
$b = 1 - a$
$s = \Sigma_{i \geq 0} e^{-c(a+i)^2} + \Sigma_{i \geq 0} e^{-c(b+i)^2}$
Randomly choose $Q \in \mathbb{Z}$ from:
for $i \geq 0$ **do**
$P[Q = r - (a+i)] = e^{-c(a+i)^2}/s$ and $P[Q = r + (b+i)] = e^{-c(b+i)^2}/s$

The input $r \in \mathbb{Q}$ is first split into an integer $p \in \mathbb{Z}$ and a decimal $a, 0 \leq a \leq 1$. Then, $b = 1 - a$ so that $0 \leq b \leq 1$. The output integer Q is now chosen from the stated probability distribution. This probabilistic procedure rounds a rational number to an integer based on a spherical Gaussian distribution, so the output will most likely be an integer close to the input, but it may also output an integer far from the input, or even round an integer to another integer.

The next procedure is $near_A(\boldsymbol{x}, d)$, which is used to determine the nearest lattice vector to a vector $\mathbf{x}$. This procedure uses the probabilistic $randRound_c$ procedure (Algorithm 2). Since the rounding in this procedure is randomized, we need to call the procedure many times so that the nearest vector returned will most likely be the nearest lattice vector if the number of calls are large enough. Let the Gram-Schmidt orthogonalized vectors corresponding to a basis $\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n$ be the vectors $\boldsymbol{b}_1^\dagger, \cdots, \boldsymbol{b}_n^\dagger$, where $\boldsymbol{b}_d^\dagger$ is the projection of $\boldsymbol{b}_d$ orthogonal to the vector space $V(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_{d-1})$. For this algorithm, we assume that the input vector $\mathbf{x}$ lies in $V(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_d)$.

Algorithm 3 $near_A(\boldsymbol{x}, d)$

if $d = 0$ **then** ▷ It is the zero vector
  return $\mathbf{x}$
else
  Let $r_d \boldsymbol{b}_d^\dagger$ be the projection of $\mathbf{x}$ in the direction of $\boldsymbol{b}_{\boldsymbol{d}}^\dagger$
  Let $c_d = A|\boldsymbol{b}_d^\dagger|^2$
  Let $\lambda_d = randRound_{c_d}(r_d)$
  Let $\boldsymbol{x}' = \boldsymbol{x} + (\lambda_d - r_d)\boldsymbol{b}_d^\dagger$
  Return $near_A(\boldsymbol{x}' - \lambda_d \boldsymbol{b}_d, d-1) + \lambda_d \boldsymbol{b}_d$

Lemma 1 *The recursion parameter* $\boldsymbol{x}' - \lambda_d \boldsymbol{b}_d$ *in near is a vector in* $V(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_{d-1})$.

Assuming $\mathbf{x}$ lies in $V(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_d)$, we know that $\boldsymbol{x}' = \boldsymbol{x} + (\lambda_d - r_d)\boldsymbol{b}_d^\dagger$ and $\boldsymbol{x}' - \lambda_d \boldsymbol{b}_d$ also lies in $V(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_d)$, since $\boldsymbol{b}_{\boldsymbol{d}}^\dagger$ and $\boldsymbol{b}_d$ must lie in $V(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_d)$. Now, since $\boldsymbol{b}_{\boldsymbol{d}}^\dagger$ is the projection of $\boldsymbol{b}_{\boldsymbol{d}}$ orthogonal to $V(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_{d-1})$, and the projection of $\boldsymbol{x}' - \lambda_d \boldsymbol{b}_d$ in the direction of $\boldsymbol{b}_d^\dagger$ is the zero vector, $\boldsymbol{x}' - \lambda_d \boldsymbol{b}_d$ lies in $V(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_{d-1})$.

When this algorithm has been called sufficiently many times, we can output the nearest vector returned by these calls. The returned vector is the vector $\boldsymbol{v}$ that will be used in the signature generation.

The key difference between Algorithm 3 and the algorithms used by GGH and NTRUSign is that Algorithm 3 utilizes a random rounding of real numbers, instead of using Babai's Round-off algorithm, which prevents the basis $\mathbf{B}$ from leaking.

When the *near* algorithm is called, the probability that it will return a vector in the lattice L, and therefore the number of calls we need to ensure a usable output, depends on the choice of the constant A, the distance from our input vector $\mathbf{x}$ to the lattice vector $\mathbf{y}$, and the product of the terms $1/s(\cdot)$, as given in the following lemma.

Lemma 2 *Let $\hat{\boldsymbol{y}}$ be a vector in the lattice $L(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_d)$. Let x be a vector in $V(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_d)$. The probability that $near_A(\boldsymbol{x}, d)$ returns $\hat{\boldsymbol{y}}$ is*

$$\left(\prod_{i \leq d} 1/s(A|\boldsymbol{b}_i^\dagger|^2)\right) \exp(-A|y - \boldsymbol{x}|^2)$$

Proof: To prove this probability we will use mathematical induction
Let $\hat{\boldsymbol{y}} = (\delta_1 \boldsymbol{b}_1, \delta_2 \boldsymbol{b}_2, \cdots, \delta_d \boldsymbol{b}_d)$
Base Case: $d = 1$
The function $near_A(\boldsymbol{x}, d)$ returns $\hat{\boldsymbol{y}}$ only if $randRound_{A|\boldsymbol{b}_1^\dagger|^2}(r_1) = \delta_1$, which, by definition of randRound, happens with a probability

$$\frac{1}{s(A|\boldsymbol{b}_1^\dagger|^2)} \exp\left(-A|\boldsymbol{b}_1^\dagger|^2 |r_1 - \delta_1|^2\right) = \frac{1}{s(A|\boldsymbol{b}_1^\dagger|^2)} \exp\left(-A|\boldsymbol{x}' - \hat{\boldsymbol{y}}|^2\right).$$

Induction step:
Suppose the formula holds up to $d - 1$, show that it also holds for d. The probability that $near_A(\boldsymbol{x}, d)$ returns $\hat{\boldsymbol{y}}$ is

$$P(\lambda_d = \delta_d) \cdot P(near_A(\boldsymbol{x} + (\lambda_d - r_d)\boldsymbol{b}_d^\dagger - \lambda_d \boldsymbol{b}_d, d-1) = \hat{\boldsymbol{y}} - \delta_d \boldsymbol{b}_d).$$

The first probability is

$$P(\lambda_d = \delta_d) = \frac{1}{s(A|\boldsymbol{b}_d^\dagger|^2)} \exp\left(-A|\boldsymbol{b}_d^\dagger|^2 (r_d - \delta_d)^2\right).$$

The second probability is

$$\left(\prod_{i<d} (1/s(A|\boldsymbol{b}_i^\dagger|^2))\right) \exp\left(-A|(\boldsymbol{x}' - \delta_d \boldsymbol{b}_d) - (\hat{\boldsymbol{y}} - \delta_d \boldsymbol{b}_d)|^2\right).$$

When we multiply them together, we get the first part

$$\frac{1}{s(A|\boldsymbol{b}_d^\dagger|^2)} \cdot \prod_{i<d} \frac{1}{s(A|\boldsymbol{b}_i^\dagger|^2)} = \prod_{i \leq d} \frac{1}{s(A|\boldsymbol{b}_i^\dagger|^2)}$$

and the second part

$$\exp\left(-A|\boldsymbol{b}_d^\dagger|^2(r_d-\delta_d)\right)\cdot\exp\left(-A|(\boldsymbol{x}'-\delta_d\boldsymbol{b}_d)-(\hat{\boldsymbol{y}}-\delta_d\boldsymbol{b}_d)|^2\right).$$

Since $\boldsymbol{x}'-\hat{\boldsymbol{y}}\in V(\boldsymbol{b}_1,\cdots,\boldsymbol{b}_{d-1})$ and $\boldsymbol{b}_d^\dagger$ are orthogonal to $V(\boldsymbol{b}_1,\cdots,\boldsymbol{b}_{d-1})$, we have that $|\boldsymbol{x}'-\hat{\boldsymbol{y}}|^2+|\boldsymbol{b}_d^\dagger|^2(r_d-\delta_d)^2=|\boldsymbol{x}-\hat{\boldsymbol{y}}|^2$. From $near_A$: $\boldsymbol{x}'=\boldsymbol{x}+(\lambda_d-r_d)\boldsymbol{b}_d^\dagger$, and since in this case, $\lambda_d=\delta_d$:

$$\begin{aligned}|(\boldsymbol{x}'+(r_d-\delta_d)\boldsymbol{b}_d^\dagger)-\hat{\boldsymbol{y}}|^2&=|(\boldsymbol{x}'+(r_d-\lambda_d)\boldsymbol{b}_d^\dagger)-\hat{\boldsymbol{y}}|^2\\&=|(\boldsymbol{x}'-(\lambda_d-r_d)\boldsymbol{b}_d^\dagger)-\hat{\boldsymbol{y}}|^2\\&=|\boldsymbol{x}-\hat{\boldsymbol{y}}|^2\end{aligned}$$

Putting it all together, the probability is

$$\prod_{i\leq d}\frac{1}{s(A|\boldsymbol{b}_i^\dagger|^2)}\exp\left(-A|\boldsymbol{x}-\hat{\boldsymbol{y}}^2|\right),$$

which is the same as is stated in the lemma. ■

3.2 Verification

When a signed ciphertext is recieved, we can decrypt the message with our private key, since it has been encrypted by using our public key. The signature has been computed with the senders private key, so we can check if it is a valid signature by using their public key. Since $\boldsymbol{s}=\boldsymbol{c}_0-\boldsymbol{v}$, and $\boldsymbol{c}_0\boldsymbol{A}^t=H(m)$

$$\boldsymbol{c}_0\boldsymbol{A}^t=\boldsymbol{s}\boldsymbol{A}^t+\boldsymbol{v}\boldsymbol{A}^t,$$

and since $\boldsymbol{v}\in\boldsymbol{B}$, the product $\boldsymbol{v}\boldsymbol{A}^t=0$, which leads to

$$\boldsymbol{c}_0\boldsymbol{A}^t=\boldsymbol{s}\boldsymbol{A}^t.$$

Verifying that $\boldsymbol{s}$ is a valid signature is done by checking if $\boldsymbol{s}$ is short and that

$$\boldsymbol{s}\boldsymbol{A}^t=H(\boldsymbol{m}).$$

References

Chen, C., Danba, O., Hoffstein, J., Hülsing, A., Rijneveld, J., Schanck, J. M., ... Zhang, Z. (2019). Algorithm specifications and supporting documentation. *Brown University and Onboard security company, Wilmington USA*.

Deng, X. (2016). *An introduction to lenstra-lenstra-lovasz lattice basis reduction algorithm.* Massachusetts Institute of Technology (MIT).

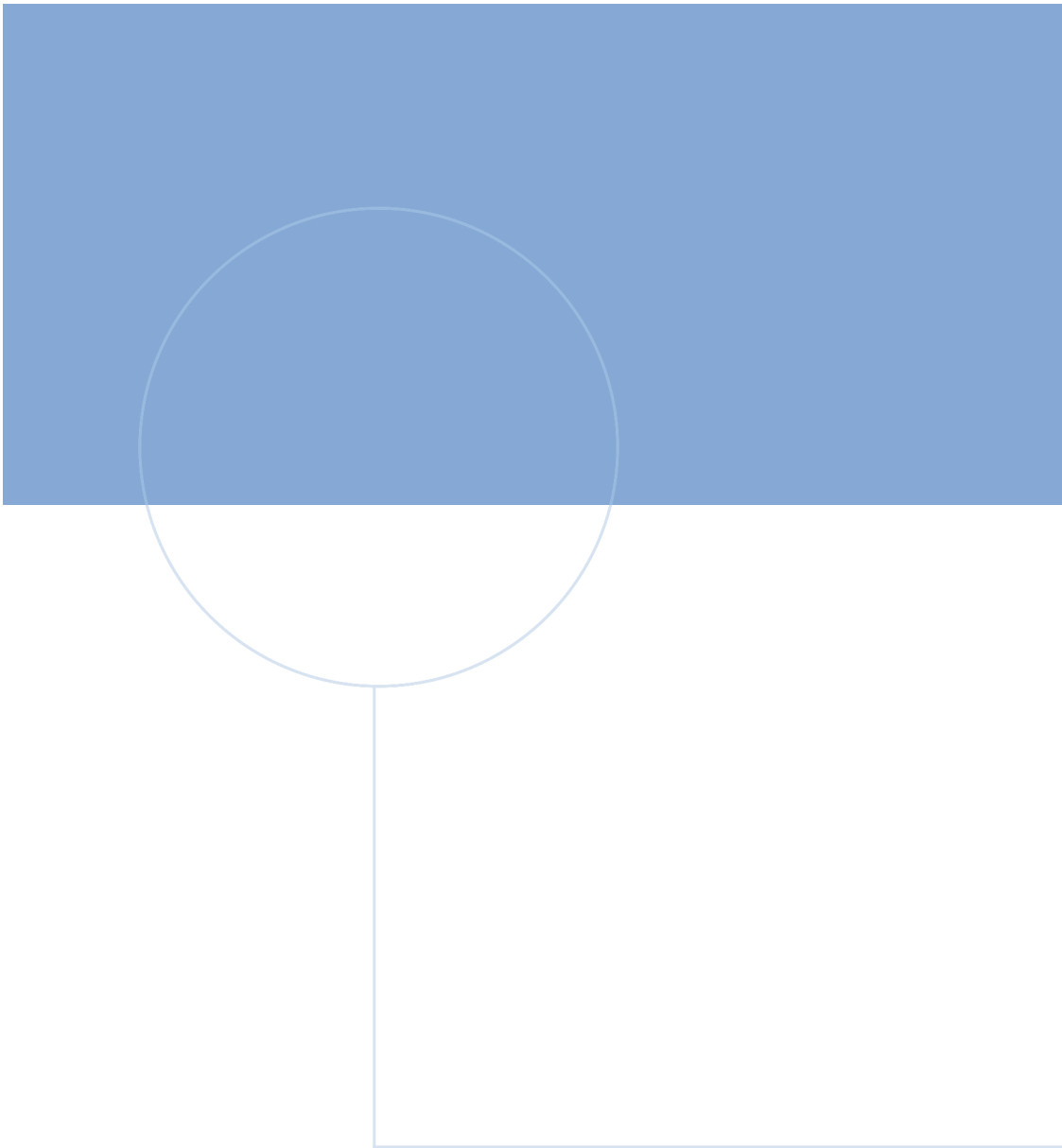
Klein, P. (2000). Finding the closest lattice vector when it's unusually close. In *Proceedings of the eleventh annual acm-siam symposium on discrete algorithms* (pp. 937–941).

Nguyen, P. Q., & Regev, O. (2006). Learning a parallelepiped: Cryptanalysis of ggh and ntru signatures. In *Annual international conference on the theory and applications of cryptographic techniques* (pp. 271–288).

Nguyen, P. Q., & Stehlé, D. (2009). An lll algorithm with quadratic complexity. *SIAM Journal on Computing*, *39*(3), 874–903.

Nist. (2020). `https://csrc.nist.gov/News/2020/pqc-third-round-candidate-announcement`.

Peikert, C. (2008). *How to use a short basis: Trapdoors for hard lattices and new cryptographic constructions.*



NTNU

Norwegian University of
Science and Technology