Lattice-Based Cryptography

Mihika Deshpande

August 18, 2024

1 Introduction

In 1981, Richard Feynman proposed the creation of the quantum computer, using entangled quantum objects to model physical objects. Such a computer could handle calculations simultaneously and process data faster than current computers. Quantum computing poses a significant risk to current cryptography schemes because they can no longer rely on the limitations of computing power to prevent attackers from learning the private key as security [8].

However, cryptography schemes based on lattices have proven secure against quantum computing because they rely on the complexity of difficult lattice problems instead of relying on number factoring like classical algorithms. As such, it's important to explore the potential usage and implementation of lattice-based cryptography systems [9].

The use of lattices in mathematics traces back to at least the 18$^{\text{th}}$ century, but the use of lattices in cryptography began during the early 1980s. At first, lattices were explored as a means of attacking existing cryptographic schemes, but by the 1990s, the exploration of lattice-based cryptographic schemes was underway [7]. The past few decades have seen significant advancements in the development of secure cryptographic schemes, including the creation of N-th degree Truncated polynomial Ring Units (NTRU), a public-key cryptosystem, Kyber, a key encapsulation mechanism, and Dilithium, a digital signing scheme [9].

The following paper will explore the foundations of lattice-based cryptography, analyze various lattice-based cryptography systems from the past few decades, and evaluate the security of lattice-based cryptography.

2 Mathematical Foundations of Lattices

Lattice-based cryptography involves using lattices, or grids formed by sets of vectors, and computationally hard problems associated with lattices to encode and decode information.

2.1 Definitions

We can start by providing formal definitions of concepts related to lattices to provide context for the following sections.

Definition 1. A *lattice*, $\mathcal{L}$ is a partially ordered set with the property that every $a, b \in L$ has a least upper bound and a greatest lower bound.

In other words, a lattice is a discrete additive subgroup of $\mathbb{R}^n$ [3]. One of the advantages of using lattices in cryptography, is that they require a minimal amount of space to store. This is because lattices can be defined by a *basis*, so only the basis of a lattice needs to be stored, not the actual points.

Definition 2. The *basis* of a lattice is a set of vectors, $\mathbb{B} = (b_1, b_2, ...b_n)$ that defines a lattice.

Ex: The set of vectors, $(0, 1)$ and $(1, 0)$ is the basis for the lattice formed with integer coordinates.

The function f that maps all bases to their lattices is surjective, meaning that a lattice can have multiple bases, but each basis only generates one lattice. For example, the set of vectors $(7, 3)$ and $(2, 1)$ are also the basis for the lattice formed with integer coordinates.

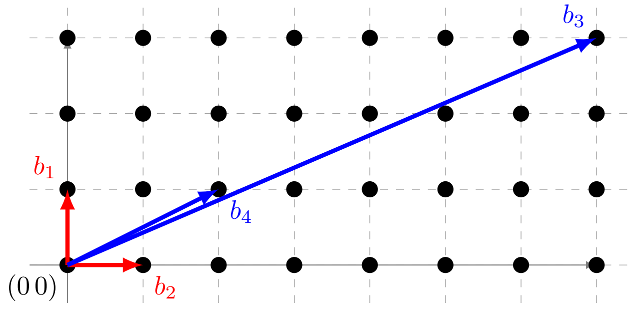

Figure 1: Multiple bases for lattices[11].

One feature of lattices and their bases is the fundamental parallelepiped formed by the basis, which plays a role in signature schemes.

Definition 3. The *fundamental parallelepiped* measures the density of the lattice, where $\mathcal{P} = \sum_i b_i \cdot [-\frac{1}{2}, \frac{1}{2})$.

In the above definition, we use the interval $[-\frac{1}{2}, \frac{1}{2})$ instead of $[0, 1)$ to center the fundamental parallelepiped around the origin, rather than have the origin at each of the vertices.

The *fundamental parallelepiped* is important as it completely fills the spaces between the lattice points with no overlap, effectively relating lattice points to points not on the lattice. We can prove this using the definition of the fundamental parallelepiped:

Theorem 1.
$$\mathbb{R}^n = \bigcup_{v \in \mathcal{L}} v + P(\mathbb{B})$$

Parallel translations of the fundamental parallelepiped by lattice vectors covers the entirety of $\mathbb{R}^n$ with no overlap [3].

Proof. For a point $p \in \mathbb{R}^n$, where x_i is the coordinates of p:

$$\begin{aligned} p &= \sum_i x_i b_i \\ &= \sum_i \lceil x_i \rfloor b_i + \sum_i (x_i - \lceil x_i \rfloor) b_i. \end{aligned}$$

Since, $-\frac{1}{2} \leq (x_i - \lceil x_i \rfloor) < \frac{1}{2}$,

$$\sum_i \lceil x_i \rfloor b_i \in \mathcal{L}$$

and

$$\sum_i (x_i - \lceil x_i \rfloor) b_i \in P(\mathbb{B})$$

Therefore, $\mathbb{R}^n = \bigcup_{v \in \mathcal{L}} v + P(\mathbb{B})$. □

2.2 Hard Lattice Problems

There are several problems in lattices that are hard to solve without access to specific information about the problem, making them ideal for cryptography schemes.

Shortest Vector Problem (SVP): Find $\min_{v \in \mathcal{L}} ||v||$.

The SVP problem to find the shortest vector in the lattice (i.e. the point closest to the origin), given the basis of a lattice.

This is easier to find the more "square" a set of vectors is, and harder to find the closer together vectors are. [3]

Closest Vector Problem (CVP): Given vector t, find $v \in \mathcal{L} : \min_{v \in \mathcal{L}} ||\text{dist}(v - t)||$

The objective is, given a point t not on the lattice, to find the vector v that is closest to the point. This problem is hard, especially with bad bases, because lattice points that are far from t may be vectors v that are very close to t. [9]

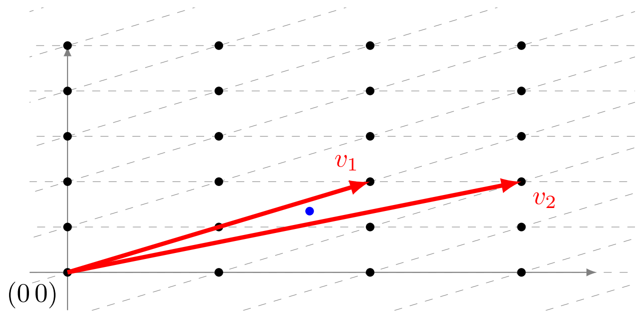

Figure 2: Close vectors to a point not on the lattice [11].

Learning With Errors (LWE) The LWE problem is given n input pairs (a_i, b_i):

$$a_i \in \mathbb{X}_p^n$$
$$b_i = a_i \cdot s + e \pmod q.$$

where e is some small error vector that adds noise, find an s, such that all of the n pairs are valid [10]. The added noise adds the complexity to the problem.

Eventually, the problem simplifies down to the SVP problem.

An example of a simple cryptography scheme based on LWE would be, to start with a square matrix A and a vector s. Using these two, we can generate a vector b, using the following equation, where e is error:

$$b = s \cdot A + e.$$

Since it's hard to recover s from b and A, we can use this in a Diffie-Helman exchange, where Alice's private key is s and public key is b and A is a publicly available matrix. [6]

2.3 Good and Bad Bases

The difficulty of solving lattice problems often depends on the type of basis provided. Given any two sets of bases, $(a_1, a_2, ..., a_n)$ and $(b_1, b_2, ..., b_n)$ that generate the same lattice, if $(a_1, a_2, ..., a_n)$ is more "square" than $(b_1, b_2, ..., b_n)$, as in the vectors in $(a_1, a_2, ..., a_n)$ are closer to perpendicular than the vectors in $(b_1, b_2, ..., b_n)$, then it's easier to solve lattice problems with $(a_1, a_2, ..., a_n)$ than $(b_1, b_2, ..., b_n)$ [9].

3 NTRU

The Nth-degree Truncated polynomial Ring Unit (NTRU) cryptosystem was created by Jeffery Hoffstein, Joseph Silverman, and Jill Pipher in 1996 [5].

3.1 Key Creation

The NTRU cryptosystems begins with three numbers, (N, p, q), where p and q are coprime. We can define the following:

$$\begin{aligned} N &\rightarrow \text{A positive integer} \\ q &\rightarrow \text{A large modulus} \\ p &\rightarrow \text{A small modulus, relatively prime to q} \\ T &\rightarrow \text{The truncated ring of polynomials with a degree of at most } N-1. \end{aligned}$$

For some integer k, let
$$\mathbb{Z}_k = 0, 1, ..., k-1$$

Then, let T be the set of polynomials with degree $N-1$ where the coefficients of the terms in the polynomials are in $\mathbb{Z}_k$. This means that for some $a, b \in T$:

$$\begin{aligned} a &= a_0 + a_1x + a_2x^2 + ...a_{n-1}x^{n-1} \\ b &= b_0 + b_1x + b_2x^2 + ...b_{n-1}x^{n-1} \\ a+b &= (a_0+b_0) + (a_1+b_1)x^1 + (a_2+b_2)x^2 + ... + (a_{n-1}+b_{n-1})x^{n-1} \\ a \cdot b &= (a_0b_0) + (a_0a_1 + b_0a_1)x_1 + (a_0b_2 + a_1b_1 + b_0a_2)x^2 + ... + (a_{n-1}b_{n-1})x^{2n-2} \\ &= (a_0b_0) + (a_0a_1 + b_0a_1)x_1 + (a_0b_2 + a_1b_1 + b_0a_2)x^2 + ... + (a_{n-1}b_{n-1})x^{n-2}. \end{aligned}$$

For $a \cdot b$, there are two truncating procedures: If the degree of the polynomial is greater than or equal to N, we take the modulus of the degree for the new term. We also take the modulus of the coefficients of all of the terms at the end.

Example 3.1. Suppose $N = 4, k = 3$: If the resulting polynomial after multiplication is $x^5 + 4x^3 + 3x$, then after taking the modulus of the degree and the coefficients, we get:

$$\begin{aligned} &x^5 + 4x^3 + 3x \\ =&x + 4x^3 + 3x \\ =&x^3 + x. \end{aligned}$$

Since $\mathbb{Z}_k$ is a group, T is closed under addition and multiplication.

To create the secret key, Alice can pick two polynomials, f and g, of degree $N-1$, where f has an inverse modulo p, F_p, and modulo q, F_q, such that,

$$ff_p \equiv 1 \pmod{p}$$

and

$$ff_q \equiv 1 \pmod{q}$$

Next, Alice computes the polynomial h,

$$h = pf_q g \pmod{q}.$$

Alice's private key is f and f_p, and Alice's public key is h.

3.2 Encryption

To encrypt a message to send to Alice, Bob can use Alice's public key, h. NTRU allows one entity to send a polynomial to another, so Bob must first convert his message into the form of a polynomial m with coefficients modulo p. He can do this by converting his message into binary, and then having the $1s$ and $0s$ as coefficients.

Example 3.2. To encrypt the message `code`:

$$\begin{aligned} c &\to 3 = 00011_2 \\ o &\to 15 = 01111_2 \\ d &\to 4 = 00100_2 \\ e &\to 5 = 00101_2 \end{aligned}$$

We can translate the binary strings into polynomials:

$$\begin{aligned} c &\to m_1 = x + 1 \\ o &\to m_2 = x^3 + x^2 + x + 1 \\ d &\to m_3 = x^2 \\ e &\to m_4 = x^2 + 1 \end{aligned}$$

The block size for messages depends on how large N is so if $N \geq 20$, we could send the message, `code` in one block.

Bob also chooses a random polynomial r, which is relatively small modulo q, which can be used to obscure the message.

Finally, Bob computes the encrypted polynomial message e,

$$e = rh + m \pmod{q}.$$

3.3 Decryption

In order for Alice to decrypt the polynomial message e, she can use her secret key for the following computations:

$$\begin{aligned} a &= fe \pmod q \\ a &= rpg + fm. \\ b &= a \pmod p \\ c &= f_p b \pmod p \end{aligned}$$

These operations can be further simplified to prove that they yield the polynomial message:

$$\begin{aligned} a &= fe \pmod q \\ &= f(rh + m) \pmod q \\ &= f(rpf_q g + m) \pmod q \\ &= frpf_q g + fm \pmod q \\ &= rpg + fm \pmod q. \end{aligned}$$

From here, we can draw the conclusion that $a = rpg+fm$ exactly, because in the setup of NTRU, we guaranteed that r and g are a relatively small polynomials, and p is a small modulus, so multiplying them together will result in a product that is already reduced modulo q.

Next, we can simplify the calculation for b :

$$\begin{aligned} b &= a \pmod p \\ &= rpg + fm \pmod p \\ &= fm \pmod p. \end{aligned}$$

Lastly, we can simplify the calculation for c :

$$\begin{aligned} c &= f_p b \pmod p \\ &= f_p fm \pmod p \\ &= m \pmod p. \end{aligned}$$

Therefore, Alice is able to decrypt the message [4].

3.4 Security

The security of NTRU depends on the fact that it uses the shortest vector problem, which is NP-hard for randomized reductions. While it's possible to break NTRU in around 40 minutes using a lattice-based attack when $N = 100$, increasing the size of N to 500 increases the break-time to around 8.4 years [4].

4 CRYSTAL-Kyber

CRYSTAL-Kyber is a key encapsulation mechanism (KEM) and public-key encryption (PKE) method, used to share a secret key between two parties without leaking it to attackers, created by Joppe Bos, Léo Ducas, Eike Kiltz, and more in 2018 [2].

Kyber relies on the Ring Learning With Errors (Ring-LWE) problem. This is a lattice-based problem which conceals a secret polynomial by adding noisy data [6].

4.1 Required Mathematics

We can start by defining a polynomial ring $R = \mathbb{Z}_p[X]/(X^n + 1)$

Let $a(x) \in R$ be the polynomial we are trying to conceal. Then, let $e(x), s(x) \in R$, where both polynomials have small coefficients. Concealing $a(x)$, we get $b(x) \in R$, where

$$b(x) = a(x) \cdot s(x) + e(x)$$

Multiplication over a polynomial ring uses a *circulant matrix.*

Definition 4. Given a sequence $s = \{a_1, a_2, ...a_n\}$, A *circulant matrix c* is a square, $n \times n$ matrix, where the first column contains the elements of s in order, and each subsequent column is formed by cycling each element down one position from the previous column. The element in the last position is cycled to the first position.

Example 4.1. If $n = 3$ and sequence $s = \{1, 2, 3\}$, then $c = \begin{bmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{bmatrix}$.

To multiply two polynomials a and b, where $a, b \in R$ we can first convert a into a circulant matrix A, with the added condition that every element that is placed above it's original position in a column, that column is negated. Then we can convert b into the $n \times 1$ matrix B with the sole column containing the coefficients of b in order, and then multiply A and B, and then modulo p, which gives us the coefficients of the polynomial $c = a \cdot b$.

Example 4.2. Let $a, b \in \mathbb{Z}_{17}[X]/(X^3 - 1)$

$$a = 12x^2 + x + 16$$
$$b = 2x^2 + 3x + 14.$$

Next, we can convert a into a circulant matrix A and b into matrix B. Multiplying A and B, gives us:

$$\begin{bmatrix} 12 & -16 & -1 \\ 1 & 12 & -16 \\ 16 & 1 & 12 \end{bmatrix} \times \begin{bmatrix} 2 \\ 3 \\ 14 \end{bmatrix} = \begin{bmatrix} -38 \\ -186 \\ 203 \end{bmatrix}.$$

Then, taking modulo 17 of the final above product, we get, $c = \begin{bmatrix} 13 \\ 1 \\ 16 \end{bmatrix}$.

This gives us the final polynomial product of $a \times b = 13x^2 + x + 16$. [6]

4.2 PKE Key Creation

Kyber has several sets of parameters for different levels of security; Kyber512 has the following parameters: $n = 256, k = 2, q = 3329$.

We can start by generating a private key, s, which has k polynomials where all polynomials are in R, and all of them have small coefficients.

$$s = (s_1, ..., s_k).$$

For the purpose of this explanation, we can define what *relatively small* means.

Definition 5. A *relatively small* polynomial in some polynomial ring, $\mathbb{Z}_p[X]/(X^n + 1)$ is a polynomial such that the absolute value of each of its coefficients is relatively small compared to p.

For the purpose of our example, $p = 17$, so coefficients $c \in \{-1, 0, 1\}$ will be considered relatively small. (See Kyber Decryption for the rationale behind using small polynomials).

Then we can generate the public key, which consists of two parts, a matrix A and a vector t. We can generate A, a $k \times k$ matrix with random polynomials, with coefficients less than q. Then, vector $t = A \cdot s + e$, where e is a random vector with small coefficients. Because of the LWE problem, a third party cannot figure out s from just A and t. [6]

4.3 PKE Encryption

We first convert the message to binary and convert the binary to a polynomial, where the place of the binary number is the power of the term and the value is the coefficient. Then we can multiply the polynomial by $\lfloor \frac{q}{2} \rceil$, yielding our message polynomial m.

Then we need to generate two random sets of k small polynomials in R, r and e_1, and one random small polynomial in R, e_2.

To encrypt m, we calculate the set of k polynomials, u and the polynomial t [6]:

$$\begin{aligned} u &= A^T r + e_1 \\ v &= t^T r + e_2 + m. \end{aligned}$$

where T is a transposition.

4.4 PKE Decryption

We can use the secret key s to uncover the noisy message m from u and v :

$$m = v - s^T u.$$

We can prove this works by substituting the encryption equations for u and v and simplifying using transposition properities.

Proof.

$$\begin{aligned} m &= v - s^T u \\ &= t^T r + e_2 + m - s^T (A^T r + e_1) \\ &= t^T r + e_2 + m - (A \cdot s)^T r + s^T e_1 \\ &= t^T r + e_2 + m - (t - e)^T r + s^T e_1 \\ &= t^T r + e_2 + m - (t - e)^T r + s^T e_1 \\ &= t^T r + e_2 + m - t^T r - e^T r + s^T e_1 \\ &= e^T r + e_2 + m + s^T e_1 \end{aligned}$$

□

This gives us the noisy message m. To retrieve the true polynomial message, we can compare m to the nearest valid message. This is made possible by the earlier usage of relative small polynomials.

We require relatively small polynomials, because Kyber relies on the LWE problem, meaning that some minimal error is added as noise to an encryption to make decryption more difficult without the secret key. Once we have retrieved the noisy message, we know that there are only a few valid polynomial messages, since there are only a few possible combinations of coefficients which could be used in the original message: $(-1, 0, 1)$.

We want to ensure that our noisy message is close to one valid message (In other words, its a point not on the lattice, and our actual message is the closest point on the lattice) so that we can undo the noise. However, this is complicated by the use of modulo q. If our final product ends up too close to q, it could end up exceeding q and becoming an integer close to 0. This means that we want to the coefficients of our polynomials to be as close to inbetween 0 and q as possible, hence we multiply by $\lceil \frac{q}{2} \rfloor$ to scale up the polynomial message before adding noise in encryption.

Given a noisy message m, we can start by rounding each coefficient to the nearest value: $\lceil \frac{q}{2} \rfloor$ or O, q. This removes the errors that previously obscured the message.

Then we can undo our scaling by dividing by $\lceil \frac{q}{2} \rfloor$, which gives us our original polynomial message. [6]

4.5 Example

For clarity, we will use reduced values for the demonstration of Kyber-PKE.

Let

$$\begin{aligned} n &= 4 \\ k &= 2 \\ q &= 17 \\ R &= \mathbb{Z}_{17}[X]/(X^4 + 1). \end{aligned}$$

Next we can create our secret key s with $k = 2$ polynomials with small coefficients $(-1, 0, 1)$:

$$s = \begin{pmatrix} -x^3 + x^2 + 1 \\ x^3 + x - 1 \end{pmatrix}$$

For our public key, we can create a random matrix A and a small error vector e to generate a vector t.

$$A = \begin{pmatrix} 6x^3 + 15x^2 + 5x + 14, & 3x^3 + 10x^2 + 5x + 11 \\ 8x^3 + 12x^2 + 4x + 10, & 16x^3 + 4x^2 + 2x + 13 \end{pmatrix}$$

$$e = \begin{pmatrix} x^3 + x \\ x^2 \end{pmatrix}$$

Then we can calculate $t = A \cdot s + e$.

$$t = \begin{pmatrix} 6x^3 + 15x^2 + 5x + 14, & 3x^3 + 10x^2 + 5x + 11 \\ 8x^3 + 12x^2 + 4x + 10, & 16x^3 + 4x^2 + 2x + 13 \end{pmatrix} \cdot \begin{pmatrix} -x^3 + x^2 + 1 \\ x^3 + x - 1 \end{pmatrix} + \begin{pmatrix} x^3 + x \\ x^2 \end{pmatrix}$$

$$t = \begin{pmatrix} (-35x^3 - 14x^2 - 4x - 3) + (8x^3 + 4x^2 + 19x + 18) \\ (-30x^3 - 8x^2 - 2x + 2) + (18x^3 - 7x^2 + 31x + 1) \end{pmatrix} + \begin{pmatrix} x^3 + x \\ x^2 \end{pmatrix}$$

$$t = \begin{pmatrix} 8x^3 + 2x^2 + 16x + 15 \\ 5x^3 + 3x^2 + 12x + 3 \end{pmatrix}$$

Our private key is s, and our public key is (A, t).

To encrypt the message `code,` we first convert each letter's position in the alphabet into binary (or ternary, since -1 is an option as well):

$$\begin{aligned} c &\to 3 = 0011_2 \\ o &\to 15 = 1111_2 \\ d &\to 4 = 0100_2 \\ e &\to 5 = 0101_2 \end{aligned}$$

Each binary string becomes a message to be sent using Kyber-PKE.

$$\begin{aligned} c &\to m_1 = x + 1 \\ o &\to m_2 = x^3 + x^2 + x + 1 \\ d &\to m_3 = x^2 \\ e &\to m_4 = x^2 + 1 \end{aligned}$$

These form the following polynomial values Typically, we would use ASCII values and encode larger blocks of the message, as opposed to alphabet positions and one letter at a time, however, because we are using a small n, and only alphabet letters in our message, we can use the letter positions.

Next, we can scale the messages by $\frac{q}{2} = 9$, giving us:

$$\begin{aligned} m_1 &= 9x + 9 \\ m_2 &= 9x^3 + 9x^2 + 9x + 9 \\ m_3 &= 9x^2 \\ m_4 &= 9x^2 + 9 \end{aligned}$$

To generate the ciphertext, we have to generate five (two for r, two for e_1, one for e_2) random small polynomials:

$$\begin{aligned} r &= \begin{pmatrix} x^3+1 \\ x^2+x \end{pmatrix} \\ e_1 &= \begin{pmatrix} x^3-x-1 \\ -x^2+1 \end{pmatrix} \\ e_2 &= (-x^2+x-1). \end{aligned}$$

Using r, e_1, and e_2 we can generate the ciphertext (u, v) for m_1 :

$$u = \begin{pmatrix} 6x^3+15x^2+5x+14, & 8x^3+12x^2+4x+10 \\ 3x^3+10x^2+5x+11, & 16x^3+4x^2+2x+13 \end{pmatrix} \cdot \begin{pmatrix} x^3+1 \\ x^2+x \end{pmatrix} + \begin{pmatrix} x^3-x-1 \\ -x^2+1 \end{pmatrix}$$

$$u = \begin{pmatrix} 12x^3+8x^2+10x+1 \\ 12x^3+7x^2+14x+4 \end{pmatrix}$$

$$v = \begin{pmatrix} 8x^3+2x^2+16x+15, & 5x^3+3x^2+12x+3 \end{pmatrix} \cdot \begin{pmatrix} x^3+1 \\ x^2+x \end{pmatrix} + (-x^2+x-1) + 9x + 9$$

$$v = 4x^3+15x^2+2x+12$$

This gives us our ciphertext, (u, v).

Now, we can decrypt the ciphertext. First we can extract the noisy message m_{n_1} :

$$\begin{aligned} m_n &= (4x^3+15x^2+2x+12) - \begin{pmatrix} -x^3+x^2+1, & x^3+x-1 \end{pmatrix} \cdot \begin{pmatrix} 12x^3+8x^2+10x+1 \\ 12x^3+7x^2+14x+4 \end{pmatrix} \\ &= (4x^3+15x^2+2x+12) - (x^3+11x^2+10x+3) \\ &= 3x^3-4x^2-8x+9 \\ &= 3x^3+13x^2+9x+9. \end{aligned}$$

To remove the noise, we can round each coefficient to the nearest value (9 or $0, 17$), giving us, $9x+9$

Lastly, we can divide by 9, giving us the original message: $x+1$.

We would repeat this process for the next three blocks to send the full message.

4.6 Security

The security of Kyber-PKE is based on the hardness of the LWE problem over module lattices. The LWE problem is as hard as worst case lattice problems, and can be reduced to the SVP problem, leading to the same conclusion of algorithms based on LWE being secure [1].

References

[1] David Balbás. "The Hardness of LWE and Ring-LWE: A Survey". In: *Cryptology ePrint Archive* (2021). URL: `eprint.iacr.org/2021/1358.pdf`.

[2] Joppe Bos et al. "Crystals - Kyber: A CCA-secure module-lattice-based KEM". In: *2018 IEEE European Symposium on Security and Privacy* (2018). DOI: `10.1109/eurosp.2018.00032`.

[3] Dong Pyo Chi et al. *Lattice Based Cryptography for Beginners.* Cryptology ePrint Archive, Paper 2015/938. 2015. URL: `eprint.iacr.org/2015/938`.

[4] Benjamin Clark. *Understanding the NTRU cryptosystem.* 2023. URL: `ideaexchange.uakron.edu/honors_research_projects/906/`.

[5] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. "NTRU: A ring-based public key cryptosystem". In: *Algorithmic Number Theory* 1423 (1998), pp. 267–288. DOI: `10.1007/bfb0054868`.

[6] Udara Pathum. *Crystals Kyber: The key to Post-Quantum Encryption.* June 2024. URL: `medium.com/@hwupathum/crystals-kyber-the-key-to-post-quantum-encryption-3154b305e7bd`.

[7] Chris Peikert. *Mathematical Background.* 2013. URL: `web.eecs.umich.edu/~cpeikert/lic15/lec01.pdf`.

[8] Robert Relyea. *Post-quantum cryptography: An introduction.* 2022. URL: `www.redhat.com/en/blog/post-quantum-cryptography-introduction`.

[9] Robert Relyea. *Post-quantum cryptography: Lattice-based cryptography.* 2023. URL: `www.redhat.com/en/blog/post-quantum-cryptography-lattice-based-cryptography`.

[10] Aviad Rubinstein. *Lecture 14: Learning with errors.* 2019. URL: `web.stanford.edu/class/cs354/scribe/lecture14.pdf`.

[11] Bill Tourloupis. *Example: Drawing lattice points and vectors.* 2012. URL: `texample.net/tikz/examples/lattice-points/`.