Python Implementation of Quantum-Resistant Lattice Cryptography

Liangyu Wang

Abstract

In August 2015, the U.S. National Security Agency (NSA) released a major policy statement urging the need to move away from Elliptic Curve-based cryptographic systems to post-quantum cryptography. One such candidate is cryptographic system based on the hardness of solving certain lattice problems. In this paper, we will look at a public key cryptographic system based on the hardness of solving the learning with error problem and demonstrate an implementation in the python programming language.

Contents

1 Why Lattice Cryptography

Since its creation in the 1980s, NSA has been a major proponent of Elliptic Curve cryptographic(ECC) systems, it has recommended ECC to be used to protect the communications of all US government agencies. However in 2015, NSA released a major policy statement on its website on the need to move away from elliptic curve cryptography and develop standards for post-quantum cryptography:

> Unfortunately, the growth of elliptic curve use has bumped up against the fact of continued progress in the research on quantum computing, which has made it clear that elliptic curve cryptography is not the long term solution many once hoped it would be. Thus, we have been obligated to update our strategy.

In 2012, prominent NSA-watcher James Bamford wrote in WIRED magazine:

> NSA made an enormous breakthrough several years ago in its ability to cryptanalyze, or break, unfathomably complex encryption systems employed by not only governments around the world but also many average computer users in the US.

In 2013, Der Spiegel reports that the Snowden leaks indicate that NSA has been able to spy on BlackBerry communications, which has been protected by ECC since blackberry purchased cryptography company Certicom, which owns the world's largest ECC-based patent portfolio.

These developments have prompted widespread speculations that NSA has found a way to break elliptic curve cryptographic systems.

In 2016, NIST(National Institute of Standards and Technology) began the Post-Quantum Cryptography Standardization project in order to "standardize one or more quantum-resistant public-key cryptographic algorithms" that is publicly disclosed worldwide and "capable of protecting sensitive government information well into the foreseeable future" .

Of the 26 submissions that advanced to the second round, most fall into three large families: lattice (12), code-based (7), multivariate polynomial (4). Lattice cryptography is based on hard to solve problems from the mathematical theory of lattices; Code-based cryptography is based on the hardness of decoding linear error-correcting codes; Multivariate polynomial cryptography is based on the hardness of solving multi-variate system of polynomial equations.

Older proposals of multivariate polynomial cryptography has already been broken, and recent proposals have not yet been thoroughly studied; Code-based cryptography is generally considered not very efficient and requires very large key size. This leaves lattice cryptography as the strongest contender for post-quantum cryptography, its encryption and decryption speed is even faster than existing public key systems like RSA.

Lattice cryptography is considered quantum-resistant because at present there is no known quantum algorithm that can solve the hard lattice problems faster than conventional computer. The best known algorithms for solving lattice problems either run in exponential time or have very bad approximation ratios.

2 Mathematics of Lattice

In the study of Algebra, a lattice $\mathcal{L}$ in the Euclidean vector space $\mathbb{R}^n$ is defined as:

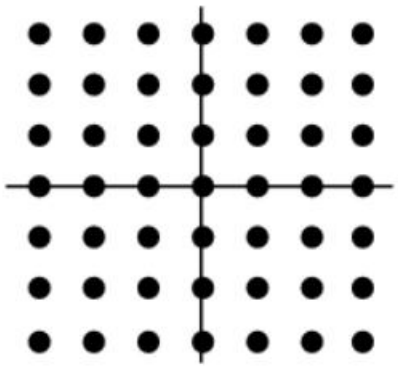
$$\mathcal{L} = \left\{ \sum_{i=1}^{n} x_i * v_i \mid x_i \in \mathbb{Z} \right\}$$

where $\{v_1, v_2, ..., v_n\}$ is a set of basis vectors in $\mathbb{R}^m$.

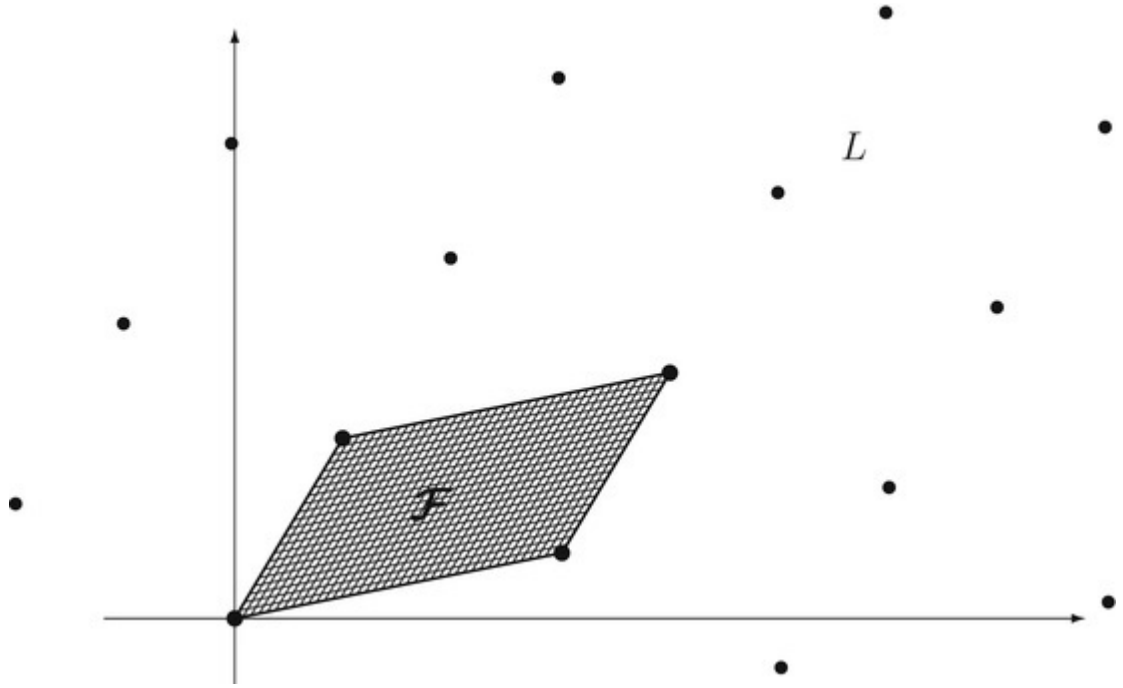
In plain English, a lattice in n-dimensional vector space $\mathbb{R}^n$ is the set of all integer linear combinations of a set of n linearly independent vectors in $\mathbb{R}^m$. A simple example of a 2-dimensional lattice is one generated by the basis vectors $\{(1,0),(0,1)\}$. The elements of this lattice would be:

$$\mathcal{L} = \left\{ \begin{array}{c} 1*(1,0)+1*(0,1)=(1,1), \\ 1*(1,0)+(-1)*(0,1)=(1,-1), \\ -1*(1,0)+1*(0,1)=(-1,1), \\ -1*(1,0)+(-1)*(0,1)=(-1,-1), \\ . \\ . \\ . \end{array} \right\}$$

we can plot $\mathcal{L}$ on a graph:



We can think of a lattice as dividing the whole of $\mathbb{R}^m$ into equal copies of an n-dimensional parallelepiped, known as the fundamental region of the lattice. In this instance, $\{x \mid x \in [0,1)^2\}$ is the fundamental parallelogram of the lattice $\mathcal{L}$. In general, $\{\sum_{i=1}^{n} x_i * v_i \mid x_i \in [0,1)^n\}$ is called the fundamental parallelepiped.



From this we can see that lattices have an n-periodic structure with periods given by the norm of its basis vectors.

Suppose we have a set of basis vectors $\{v_1, v_2, ..., v_n\}$ where each vector has m entries, then we can define an $m \times n$ matrix

$$B = [\ v_1 \ \ v_2 \ \ . \ \ . \ \ v_n\ ]$$

The lattice generated by matrix B is

$$\mathcal{L}(B) = \{Bx \mid x \in \mathbb{Z}^{\kappa}\}$$

and the determinant of B is the volume of the fundamental parallelepiped of $\mathcal{L}$.

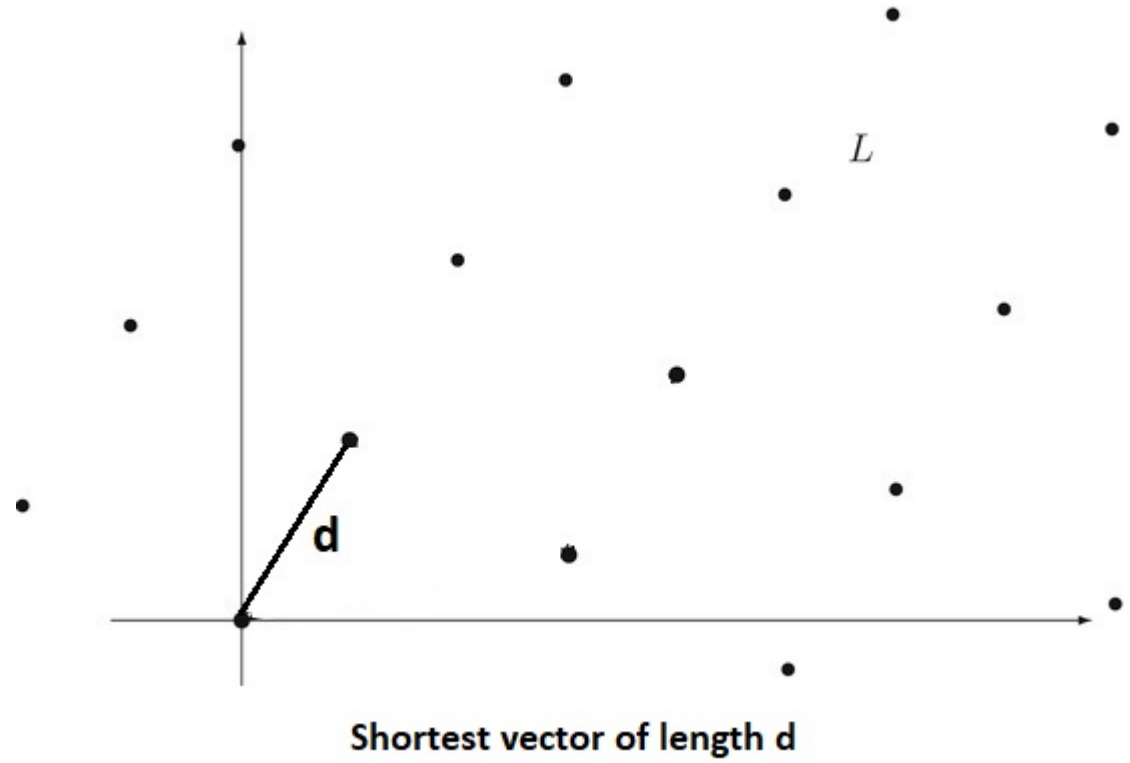
Operations on such matrices form the basis of lattice-based cryptography.

3 Shortest Vector Problem(SVP) and SVP$_\gamma$

Let the length of a vector $v \in \mathcal{L}$ be defined by the standard Euclidean norm

$$\|v\| = \sqrt{v_1^2 + \cdots + v_n^2}$$

We want to find the shortest non-zero vector in a Lattice $\mathcal{L}$ that is an integer linear combination of the basis of $\mathcal{L}$, that is we want to find $v_{shortest} \in \mathcal{L}$ that minimizes the Euclidean norm $\|v\|$ and that $\|v\| \neq 0$. This is called the shortest vector problem.



Shortest vector of length d

Suppose we are given the basis $B = \{(1,0),(0,1)\}$, then it is trivial to see that the length of the shortest vector in this lattice generated by B is 1. But if we are given the basis $B' = \{(9,4),(11,5)\}$ which generates the exact same lattice, then finding the shortest vector becomes difficult because we have to solve the optimization problem

$$\begin{cases} 9j + 11k = y \\ 4j + 5k = z \end{cases}$$

such that $\|(y,z)\|$ is the smallest non-zero in the whole lattice. In principle, this type of system of equations is over-determined and thus have infinite number of solutions, but we are constrained to find minimum integer solutions, making this type of optimization problem extremely difficult to solve, especially in high dimensional lattices.[7]

So we can see that the difficulty of the SVP can depend on the basis we are given. There are "good" basis in which SVP is easier to solve, and "bad" basis in which SVP is more difficult to solve. For cryptography, we want to find the most "bad" basis in which the SVP is the hardest to solve.

In general, SVP can be solved relatively easily in 2 dimensions but becomes exponentially more difficult if we scale the problem to hundreds or thousands of dimensions. In fact,in higher dimensions, there is no better way to solve the problem than enumerating all possible vectors lesser than a given length which requires exponential time to compute. Which is why SVP is generally considered secure against quantum computers.

If instead of solving the SVP which is hard, we try to solve an approximate SVP. That is we want to find $v \in \mathcal{L}$ such that $\|v\| \leq \gamma(n)\|v_{shortest}\|$, where n is the dimensions of the lattice $\mathcal{L}$ and $\gamma(n)$ is a function of n. In higher dimensions, this approximation of the SVP can be solved in polynomial time using an algorithm called LLL lattice basis reduction algorithm, a vector analogue of the Euclidean algorithm to compute greatest common divisor of two integers. We can choose a large enough lattice dimension n and γ such that $\gamma \approx n$ to make SVP_γ quantum resistant against LLL basis reduction algorithms.

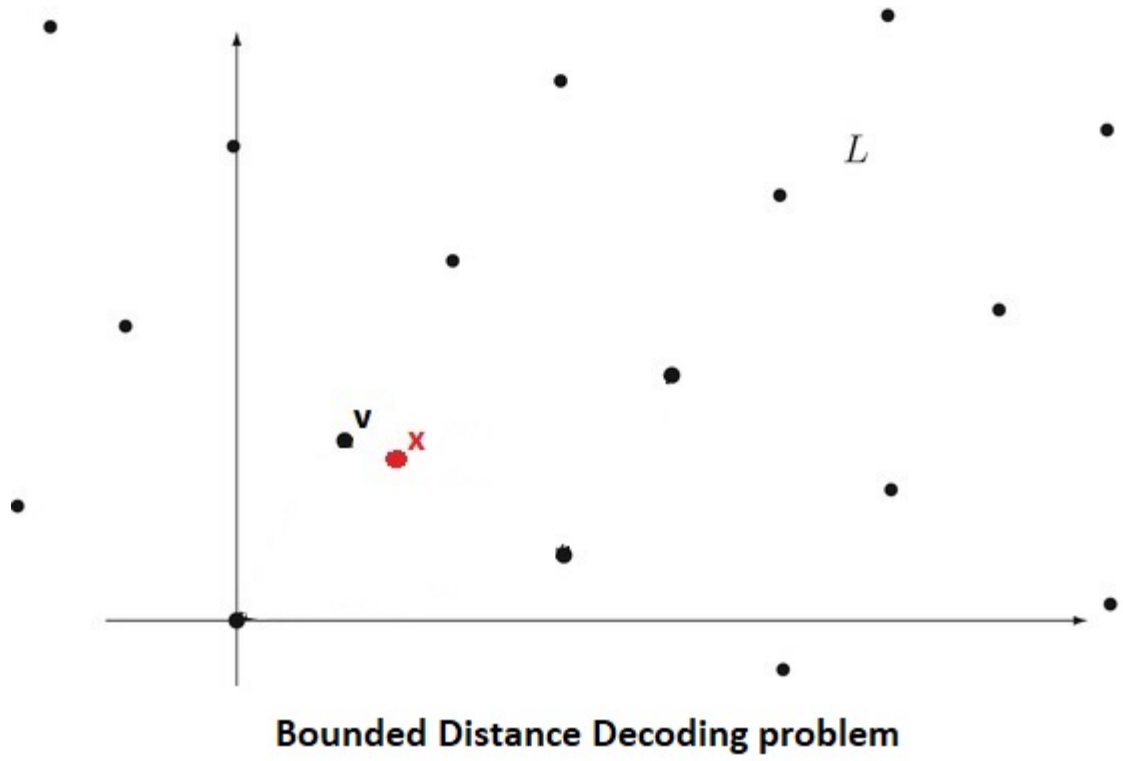
4 The Bounded Distance Decoding Problem

Suppose we are given a basis B that generates an n-dimensional Lattice L, and a challenge point $x \in \mathbb{R}^n$ not necessarily in $\mathcal{L}$, with the guarantee that there exists a unique vector $v \in \mathcal{L}$ such that $\|x, v\| \leq \frac{\|v_{shortest}\|}{2}$; we want to find v. This is called the Bounded Distance Decoding Problem (BDDP). We require the distance between x and v to be less than $\frac{\|v_{shortest}\|}{2}$ since any farther than that then v could be closer to a different lattice point.

BDDP is a Quantum hard problem in general, but like SVP, can be easier to solve if given a good enough basis.

Good Basis: A basis B for a lattice $\mathcal{L}$ is good if it is made up of nearly orthogonal short vectors. That is if every basis vector is of the form $(0, ..., 0, k, 0, ..., 0)$ for some small value $k \in \mathbb{R}$.

Bad Basis: A basis B for a lattice $\mathcal{L}$ is bad if it is made up of non-orthogonal long vectors.



Bounded Distance Decoding problem

Now we see how the BDDP can be turned into a public key cryptographic system; intuitively, We encode a secret message as a lattice point m, then add a small noise e to m, to decode $m + e \notin \mathcal{L}$ and recover the original point m would be easy for someone who holds the good basis of the lattice, but hard for someone who holds the bad basis.

Now how difficult it is to turn a bad basis into a good basis? For euclidean spaces, the Gram-Schmidt orthogonalization process allows one to transform any random basis for a space into an orthogonal basis, the obtained basis can then be normalized to minimize their length. Can such algorithm be devised to solve the BDDP?

The answer is no, because to obtain a good basis one must first figure out the unit length of the good basis vectors, essentially one

must first solve the Shortest Vector Problem. So we see that solving the BDDP is as hard as solving the SVP.

5 Learning with Error

Learning with Error is a generalization of the Learning Parity with Noise problem from machine learning.

Suppose we have a function

$$f(a) = a_0x_0 + \cdots + a_nx_n$$

and we are given some samples $(a, f(a))$, we want to learn what the function f is.

If given enough samples $(a, f(a))$, we can reduce this problem to that of solving a system of linear equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m, \end{aligned}$$

where we can solve for $(x_1, x_2, \cdots, x_n)$ using Gaussian elimination.

Now what if we add some noise e to our function and require our solution to be in $\mathbb{Z}_q$, then our problem becomes learning the form of the function:

$$f(a) = a_0x_0 + \cdots + a_nx_n + e \pmod{q}$$

This problem has been proven to be extremely difficult to solve, even a quantum computer will not make it easier. The intuition is that at each step in the Gaussian elimination procedure, the error term is amplified until it eclipses all useful information about the function we want to learn.

There are two versions of the learning with error problem. Search version and Decision version.

Search Learning With Error Problem: given m independent samples (a_i, b_i), find uniformly random secret s such that $a_i \cdot s + e = b_i$, where e is a small error randomly sampled from a Gaussian distribution.

Decision Learning with Error Problem: Let s be a uniformly random secret and e a small error randomly sampled from a Gaussian distribution. Given m independent samples (a_i, b_i), distinguish which is $(a_i, b_i) = (a_i, a_i \cdot s + e)$ and which is a uniform sample.

It can be proven that having a procedure for solving the search-LWE problem would automatically yield a solution to the decision-LWE problem and vice versa; and solving the search-LWE problem is equivalent to solving the approximate SVP and the Bounded Distance Decoding problem on a certain family of q-ary m-dimensional lattices.[9]

6 Public Key Cryptographic System Based on LWE

In Public Key Cryptographic System, we have two parties Alice and Bobby that want to communicate with each other. Alice and Bobby both select a secret private key and calculate a public key from the private key, then they both publishes their public key If Alice wants to send a message to Bobby, she will encrypt her message using Bobby's public key, once Bobby obtains message, he will decrypt the message using his own private key. If Bobby wants to send a message to Alice, he will encrypt his message using Alice's public key, and once Alice obtains the message, she will decrypt his message using her secret private key.

In the LWE public key cryptographic system, Alice chooses a modulus q, an $m \times n$ matrix A over $\mathbb{Z}_q^{m \times n}$, a secret key $s \in \mathbb{Z}_q^n$ and a "small" error vector $e \in \mathbb{Z}^n$, sampled from a Gaussian distribution. Alice then calculate

$$b^t = s^t A + e^t \tag{1}$$

Alice publishes (A, b, q) as her public key. Here, the dimension of the matrix A is the security parameter that determines the strength of the cryptographic system.

Bobby wants to send a message to Alice. He chooses a secret binary vector $x \in \{0,1\}^m$, then calculate

$$\begin{aligned} u &= Ax \\ v &= b^t x + bit \cdot \frac{q}{2} \end{aligned} \tag{2}$$

where bit is the binary representation of the message. Bobby then send the encrypted message (u, v) to Alice.

When Alice receives (u, v) she computes

$$v - s^t u \tag{3}$$

The key insight here is that the term $e^t x$ is very small compared to $\frac{q}{2}$, therefore

$$v - s^t u \approx bit \cdot \frac{q}{2} \tag{4}$$

Thus if Alice's calculation evaluates to a value close to zero, she will know Bobby sent her a bit of 0; if her calculation evaluates to a value close to $\frac{q}{2}$, she will know Bobby sent her a bit of 1.

For an $m \times n$ matrix A choosen as public key, if m is much smaller than n, then the LWE problem is easy to solve because there are fewer constraints and the solution space is much larger. Therefore in order to guarantee security, we need to choose $m \times n$ matrix for which m is much larger than n; We also need to set q such that $q > \sqrt{n}$.

Suppose we use a 640×256 matrix, and set $q = 4093$, then the public key size of the LWE cryptographic system is calculated to be $640 \cdot 257 \cdot log_2(4093) = 1973586$ bits. Much larger than 3072-bit RSA public key or 256-bit elliptic curve public key.[15]

7 Ring Learning With Error

One disadvantage of public key system based on LWE is that it requires very large key size which is a large $m \times n$ matrix. Now instead of arbitrary randomly sampled matrix, we consider matrix of the form

$$\begin{bmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ -c_1 & c_0 & c_{n-1} & & c_2 \\ \vdots & -c_1 & c_0 & \ddots & \vdots \\ -c_{n-2} & & \ddots & \ddots & c_{n-1} \\ -c_{n-1} & -c_{n-2} & \cdots & -c_1 & c_0 \end{bmatrix} \pmod{p}.$$

This type of matrix is called anti-circulant matrix, it is fully defined by its first row vector, and every row after the first is a cyclic shift of the row above with a special wrapping rule: x wraps to $-x \pmod{p}$. We can represent this matrix with the polynomial $f(x) = c_0 + c_{n-1}x + \cdots + c_1x^{n-1}$. Polynomial representation of circulant matrices have the special property that they can be multiplied very efficiently using Fast Fourier Transform. Operations on such polynomials form the basis of Ring Learning with Error.

With RLWE we can quadratically reduce the key size, since instead of using $n \times n$ matrix as public key, we can just use an nth degree polynomial as the public key. For example, to achieve 128-bits of security, LWE will require a key size of 58,982,400 bits, while RLWE only need a key size of 7680 bits.[14]

With RLWE, instead of vectors in a vector space over finite field $\mathbb{Z}/p\mathbb{Z}$, we perform operations on polynomials $f(x)$ with coefficients from $\mathbb{Z}/p\mathbb{Z}$. A polynomial

$$a_nx^n + a_{n-1}x^{n-1} + \cdots + a_2x^2 + a_1x + a_0$$

can be viewed as a vector $(a_n, a_{n-1}, ..., a_1, a_0)$ that is a linear combination of the basis $x^n, x^{n-1}, ..., x, 1$.

Therefore polynomials with coefficients from $\mathbb{Z}/p\mathbb{Z}$ is a vector space over $\mathbb{Z}/p\mathbb{Z}$, the advantage over the Euclidean vector space is that polynomials can be multiplied in a natural way that is not

true of vectors in a Euclidean vector space. The product of two polynomials is another polynomial, while the dot product of two vectors in the euclidean vector space is a scalar.

In RLWE, instead of using the Euclidean norm, we use another norm called the infinity norm

$$\|f\|_\infty = \max\{|a_1|, \ldots, |a_n|\}$$

where $a^n, a^{n-1}, ..., a_1$ are the coefficients of the polynomial f. In other words, the length of a polynomial f is defined as the largest coefficient of f. Having defined the norm, we can talk about the notion of "small" polynomials.

Now instead of working with an infinite number of polynomials, we want to work with only a finite number of polynomials, so just like we can do arithmetic with only integers modulus some prime number p, we can work with only polynomials modulus some irreducible polynomial. Just like a prime number is a positive integer not divisible by any number other than 1 and itself, an irreducible polynomial is only the product of 1 and itself. In RLWE, we generally work with polynomials modulus irreducible polynomial $x^n + 1$, where n is a power of 2 . It can be shown that the set of polynomials with coefficients from $\mathbb{Z}/p\mathbb{Z}$ modulus $x^n + 1$ denoted by $\mathbf{F}_p[x]/(x^n+1)$, contains p^n polynomials with degree no greater than n.

8 Public Key Cryptographic System Based On Ring Learning With Error

Having defined Ring Learning with Error, we can now describe a public key cryptographic system based on RLWE.[11]

Key Generation Alice Selects the ring $R = \mathbf{F}_p[x]/(x^n + 1)$such that n is a power of 2. Choose a polynomial $a \in R_p$ such that the coefficients of a are uniformly random in $\mathbb{Z}_p$, as well as two random "small" polynomial $s, e \in R$ where the coefficients of s, e are sampled from a discrete Gaussian error distribution χ. Here ,

s is Alice's the secret private key. Alice then publishes

$$(a, b = a \cdot s + e) \in R_p^2$$

as her public key.

Encryption In order to send an n-bit message $m \in \{0,1\}^n$ to Alice, Bobby encodes the message m into the 0-1 coefficients of a polynomial $z \in R_p$. Bobby then chooses 3 random "small" polynomials $r, e_1, e_2 \in R$ from χ.

Bobby computes

$$u = a \cdot r + e_1 \mod p$$

and

$$v = b \cdot r + e_2 + \frac{p}{2} \cdot z \mod p$$

then sends (u, v) as the ciphertext.

Decryption Upon receiving the ciphertext (u, v), Alice computes

$$v - u \cdot s = (r \cdot e - s \cdot e_1 + e_2) + \frac{p}{2} \cdot z \mod p$$

The coefficients of $r \cdot e - s \cdot e_1 + e_2 \in R$ should have magnitude less than $\frac{q}{4}$, therefore each bit of m can be recovered by rounding each coefficient of $v - u \cdot s$ to either 0 or $\frac{q}{2}$, whichever is closest modulo p.

9 Diffie-Hellman Key Exchange Based on RLWE

Looking at the Ring Learning with Error public key crypto system we can see that it in fact resembles the Diffie-Hellman key-agreement protocol and ElGamal cryptosystem. The $a \in R_q$ is analogous to the generator of a (multiplicative) cyclic group, and taking noisy products is analogous to exponentiation. Thus we can

modify the RLWE public key crypto system into a Diffie-Hellman key-exchange algorithmm as follows:

Alice and Bobby agree on a polynomial $a \in R_q$. Alice chooses small secret s_1 and s_0, and calculate $b = a \cdot s_1 + s_0$ and send b to Bobby.

Bobby chooses small secret e_0, e_1, and calculate $u = e_0 \cdot a + e_1$ and send u to Alice.

Alice can now calculate the shared secret $S = w = u \cdot s_1$ and Bobby picks a small secret e_2 and calculate the shared secret $S' = v = e_0 \cdot b + e_2$. However S and S' are only approximately equal because $S - S' = e_0 \cdot a \cdot s_1 + e_1 \cdot s_1 - e_0 \cdot a \cdot s_1 - e_0 \cdot s_0 - e_2 = e_1 \cdot s_1 - e_0 \cdot s_0 - e_2$, which is a small noise. Now We will define several functions that will help us remove the noise and achieve an exact key agreement.

Randomized Rounding Function with randomized rounding function, we partition the coefficients of the polynomial u into four quadrants:

$$\begin{aligned}
\mathbf{I_0} &:= \mathbb{Z}_q \cap [0, q/4) \\
\mathbf{I'_1} &:= \mathbb{Z}_q \cap [q/4, q/2) \\
\mathbf{I'_0} &:= \mathbb{Z}_q \cap [q/2, 3q/4) \\
\mathbf{I_1} &:= \mathbb{Z}_q \cap [3q/4, q)
\end{aligned}$$

then probabilistically distribute the coefficients on the boundaries of the quadrants such that the numbers of coefficients in $\mathbf{I_0} \cup \mathbf{I'_0}$ and $\mathbf{I_1} \cup \mathbf{I'_1}$ balances out. We do it for two separate cases:

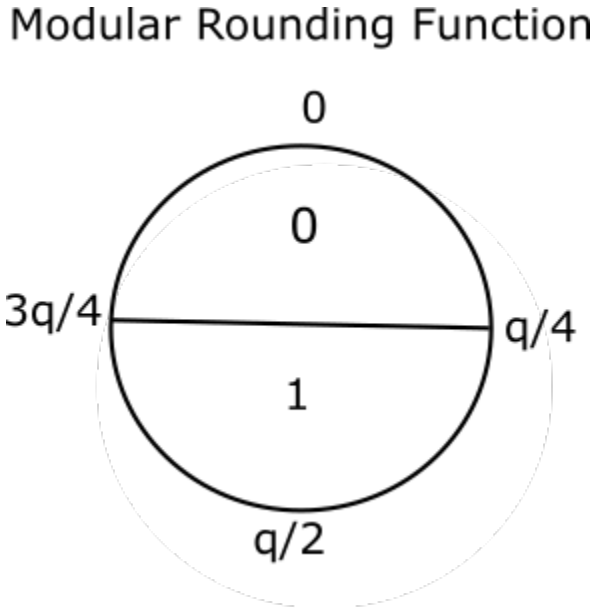
$q \equiv 1 \pmod 4$: if the coefficient is 0, we flip a uniformly random coin and map 0 to either itself or $q - 1$ depending on the result of the coin toss. Independently, if the coefficient is $(q-1)/4$, we flip a uniformly random coin and map $(q-1)/4$ to either itself or $(q+3)/4$ depending on the result of the coin toss.

$q \equiv 3 \pmod 4$: if the coefficient is 0, we flip a uniformly random coin and map 0 to either itself or $q-1$ depending on the result of the coin toss. Independently, if the coefficient is $(3q-1)/4$, we flip a uniformly random coin and map $(3q-1)/4$ to either itself or $(3q+3)/4$ depending on the result of the coin toss.

Modular Rounding Function We define the Modular Rounding function $Mod_2 : \mathbb{Z}_q \to \mathbb{Z}_2$ as

$$Mod_2(\bar{v}) := \lfloor \frac{2}{q} \cdot \bar{v} \rceil \pmod 2$$

where $\lfloor \cdot \rceil$ means rounding to the nearest integer and $\bar{v}$ is the randomize rounded coefficients of the polynomial v. Essentially, the Mod_2 function generates a secret key stream using $\bar{v}$; for every coefficient c in $\bar{v}$, if $c \in (q/4, 3q/4)$, then it is represented by a bit of 1, else it is represented by a bit of 0.

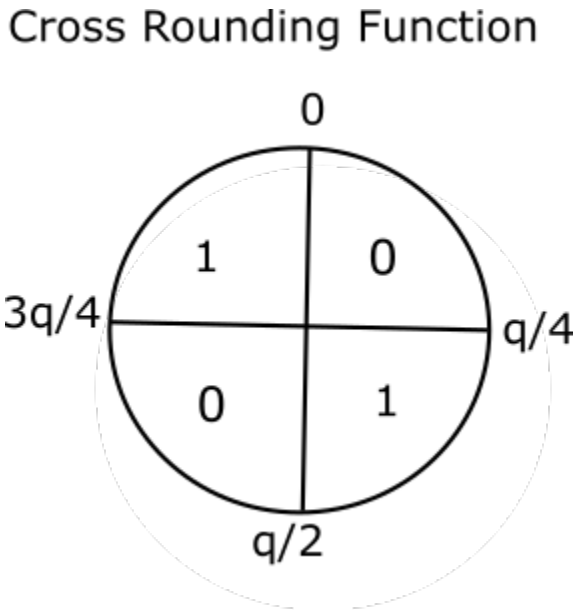


We will use $Mod_2(\bar{v})$ to generate Bobby's shared secret key stream.

Cross Rounding Function We define the Cross Rounding function $CoR_2 : \mathbb{Z}_q \to \mathbb{Z}_2$ as

$$Cor_2(\bar{v}) := \lfloor \frac{4}{q} \cdot \bar{v} \rfloor \pmod 2$$

where $\lfloor \cdot \rfloor$ is the floor function and $\bar{v}$ is the randomize rounded coefficients of the polynomial v. Essentially, the CoR_2 function generates an array of masking bits using $\bar{v}$; for every coefficient c in $\bar{v}$, if $c \in [q/4, q/2) \cup [3q/4, q)$, then it is represented by a masking bit of 1, else it is represented by a masking bit of 0.



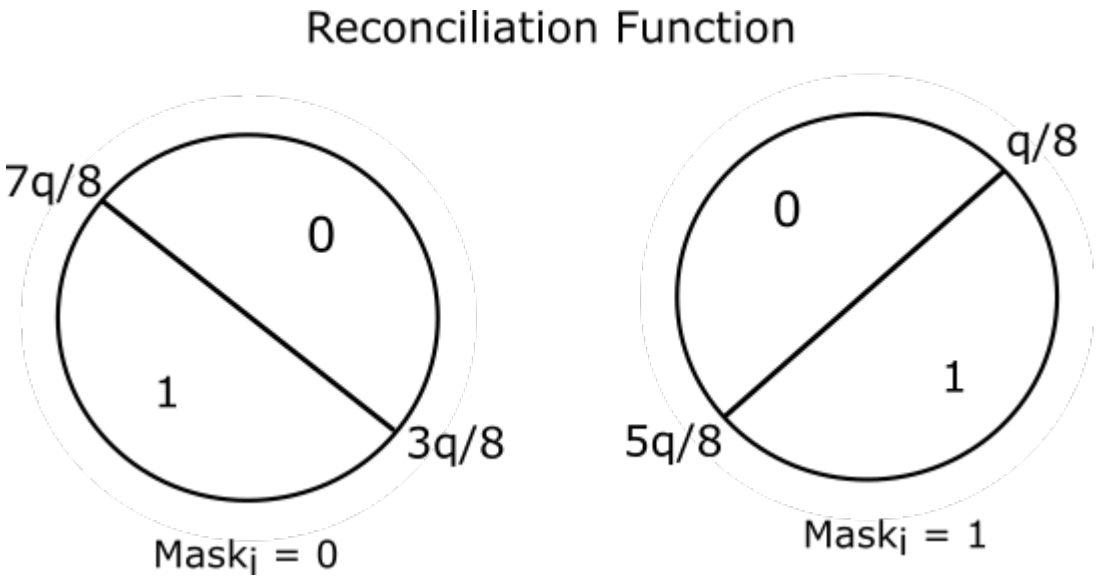
We will use the Cross Rounding function to generate a hint, called a "Mask", and send it to Alice, to help her correct the error contained in the her shared secret key. The Mask itself does not contain enough information that would allow an attacker to calculate the shared secret.

Reconciliation Function Suppose b is a masking bit of $Cor_2(\bar{v})$ such that $\bar{v} \in \mathbf{I_b} \cup \mathbf{I_b}'$ and $E := [-q/8, q/8)$ and the Reconciliation function is defined as:

$$rec(w, b) := \begin{cases} 0 & \text{if } w \in \mathbf{I_b} + E \pmod{q} \\ 1 & otherwise \end{cases}$$

Essentially, the Reconciliation function will convert an approximate shared secret key w to shared key stream according to the masking bits' value. For every coefficient c of w, if the masking bit is 0, c will be represented by a key of 1 if $c \in [3q/8, 7q/8)$ or a key of 0

otherwise; and if the masking bit is 1, c will be represented by a key of 1 if $c \in [q/8, 5q/8)$ or a key of 0 otherwise.



Alice will use the Reconciliation function to convert her approximate shared secret key to a shared secret key stream using the hint Mask sent by Bobby.

Now we can describe the complete Ring Learning with Error Diffie-Hellman Key Exchange protocol:

Alice and Bobby will agree on a public key $a \in R$. Alice chooses small secret s_1 and s_0, and calculate the public key $b = a \cdot s_1 + s_0$ and transmit it to Bobby.

Upon receiving public key b, Bobby chooses small secret e_0 and e_1 and calculate public key $u = e_0 \cdot a + e_1$.

Bobby then chooses small secret e_2 and calculate his approximate shared secret $v = e_0 \cdot b + e_2$. Bobby apply the Randomized Rounding function to v to obtain $\bar{v}$, and calculate $key = Mod_2(\bar{v})$ to obtain his shared secret key stream.

Bobby then calculate the hint $Mask = Cor_2(\bar{v})$, then send both u and Mask to Alice.

Upon receiving u and Mask, Alice calculates her approximate shared secret $w = u \cdot s_1$. Then Alice use the Reconciliation function $Rec(w, Mask)$ to obtain her shared secret key stream.

10 Python Implementation of LWE

```
import numpy as np

q = 21
bit1 = 1
bit0 = 0

print("\nQuantum-Resistant Lattice Cryptography\n")
print("1-bit LWE\n")

print("Alice selects Modulus q:")
print(q)
print()
A = np.random.randint(0, q, (21, 4))
print("Alice selects Matrix A:")
print(A)
print()
s = np.array([0, 1, 2, 3])
print("and secret key s:")
print(s)

#random error vector sampled from Gaussian distribution
e = np.random.normal(loc=0, scale=2, size=21).astype(int

print()
print("and secret error e:")
print(e)

b = np.matmul(A, s)%q
b = np.add(b, e)%q
print()
print("Alice computes b:")
print(b)
print()
```

```
print("Alice publishes (A, b, q) as her public key \n\n"
#input("press enter to continue...")
print()
x = np.random.randint(0, q, (21))%2
print("Bobby selects secret binary key x:")
print(x)
print()


print("Bobby wants to send to Alice " + str(bit1))
print()

u = np.matmul(x.T, A)%q
v = (np.matmul(b.T, x)+int(bit1*q/2))%q


print("Bobby sends ciphertext (u, v):")
print(u)
print(v)
print()

print("Alice decrypts:")
msg = np.absolute((v - np.matmul(s, u)%q))


if (msg < q/4):
    print(0)
else:
    print(1)


print()
print()
print("Bobby wants to send to Alice " + str(bit0))
print()
```

```
u = np.matmul(x.T, A)%q
v = (np.matmul(b.T, x)+int(bit0*q/2))%q


print("Bobby sends ciphertext (u, v):")
print(u)
print(v)
print()

print("Alice decrypts:")
msg = np.absolute((v - np.matmul(s, u)%q))


if (msg < q/4):
    print(0)
else:
    print(1)
```

11 Python Implementation of RLWE

```
import numpy as np
from numpy.polynomial import polynomial as p
import json
import base64

q = 255
n=8
xN_1 = [1] + [0] * (n-1) + [1]

# character to send
c = "K"
```

```
print("\nQuantum-Resistant Lattice Cryptography\n")
print("Ring Learning with Error\n")

# convert polynomial to centered representation
def center(f):
    center_f = []
    for c in f:
        if c >= 0 and c <= (q-1)/2:
            center_f.append(c)
        elif c > q/2 and c <= q-1:
            center_f.append(c-q)
    return center_f

def str2bin(text, encoding='utf-8', errors='surrogatepas
    bits = bin(int.from_bytes(text.encode(encoding, erro
    return bits.zfill(8 * ((len(bits) + 7) // 8))

def bin2str(bits, encoding='utf-8', errors='surrogatepas
    n = int(bits, 2)
    return n.to_bytes((n.bit_length() + 7) // 8, 'big').



a = np.random.randint(0, q, (9))
s = [0,1,2,3,4,5,6,7,8]
e = [2,0,-1,0,1,0,-1]

b = p.polymul(a, s)%q
b = p.polyadd(b, e)%q
b = p.polydiv(b, xN_1)[1].astype(int)%q

# while True:
#     c = input('Enter a character: ')
#     if len(c) == 1:
#         break
#     print('Please enter only one character')
```

```
print()
print("public key (a, b)")
print("a")
print(a)
print("b")
print(b)


m = [int(bit) for bit in str2bin(c)]
print()
print("message to encrypt")
print(c)
print()
print("binary of message to send")
print(np.asarray(m))

r = [1,0,-1,0,1,0,1]
e1 = [1,0,1,0,-1,0,1]
e2 = [2,0,1,0,1,0,-1]


# r = np.floor(np.random.normal(0, size=(5)))
# e1 = np.floor(np.random.normal(0, size=(5)))
# e2 = np.floor(np.random.normal(0, size=(5)))



print()
u = p.polymul(a, r)%q
u = p.polyadd(u, e1)%q
u = p.polydiv(u, xN_1)[1]%q
u = u.astype(int)


print()
z = [int(np.round(q/2+0.1))*i for i in m]
```

```
v = p.polymul(b, r)%q
v = p.polyadd(v, e2)%q
v = p.polydiv(v, xN_1)[1]%q
v = p.polyadd(v, z)%q
v = p.polydiv(v, xN_1)[1]%q
v = v.astype(int)
print("ciphertext (u, v)")
print(u)
print(v)
print()

print("base64: ")

uv = u.tolist()+v.tolist()
uvstr = json.dumps(uv)
base64str = base64.b64encode(uvstr.encode('utf-8'))
print(base64str)
print()

dm = p.polymul(u, s)%q
dm = p.polydiv(dm, xN_1)[1]%q


dm = center(p.polysub(v, dm)%q)


dm = [int(np.round((t*2)/q+0.1)) for t in dm]
dm = np.absolute(dm)



print("Decryption ")
print(dm)

print(bin2str("".join(str(bit) for bit in dm)))
```

12 Python Implementation of RLWE Diffie-Hellman Key Exchange

```
import numpy as np
from numpy.polynomial import polynomial as p


q = 4095
n=128
xN_1 = [1] + [0] * (n-1) + [1]


def randomized_round(f, q):
    round_f = [];
    lenf = len(f)
    for i in range(lenf):
        coin = np.random.randint(0,2)
        if f[i] == 0:
            if coin == 0:
                round_f.append(0)
            else:
                round_f.append(q-1)
        elif q%4 ==3 and f[i] == (3*q-1)/4:
            if coin == 0:
                round_f.append(f[i])
            else:
                round_f.append((3*q+3)/4)
        elif f[i] == (q-1)/4:
            if coin == 0:
                round_f.append(f[i])
            else:
                round_f.append((q+3)/4)
        else:
            round_f.append(f[i])
```

```
    return round_f

def ModularRound(v, q):
    return np.round(p.polymul(2/q, v))%2

def CrossRound(v, q):
    return np.floor(p.polymul(4/q, v))%2


def rec(w,b):
    res = []
    index = min(len(w), len(b))
    for i in range(index):
        if b[i] == 1:
            if w[i] >= q/8 and w[i] < 5*q/8:
                res.append(1)
            else:
                res.append(0)
        elif b[i] == 0:
            if w[i] >= 3*q/8 and w[i] < 7*q/8:
                res.append(1)
            else:
                res.append(0)
    return res


print("\nQuantum-Resistant Lattice Cryptography\n")
print("Ring Learning with Error Diffie-Hellman Key Excha

print("Lattice Dimension: " + str(n))
print()
print("q: " +str(q))
print("q is of the form 2^k-1, q needs to be at least 10
print()
print()
```

```
a = np.random.randint(0, q, (n))
# a = [89, 13,  6, 73, 28, 30,  0,  1]

print("a:" + str(a))
print()

#sample random errors from Gaussian distribution
s1 = np.floor(np.random.normal(0, size=(n)))
s0 = np.floor(np.random.normal(0, size=(n)))

b = p.polymul(a, s1)%q
b = p.polyadd(b, s0)%q
b = p.polydiv(b, xN_1)[1]%q
print("Public Key b:")
print(b.astype(int))
print()


#sample random errors from Gaussian distribution
e0 = np.floor(np.random.normal(0, size=(n)))
e1 = np.floor(np.random.normal(0, size=(n)))
e2 = np.floor(np.random.normal(0, size=(n)))

u = p.polymul(e0, a)%q
u = p.polyadd(u, e1)%q
u = p.polydiv(u, xN_1)[1]%q
# print("u")
# print(u.astype(int))
# print()

v = p.polymul(e0, b)%q
v = p.polyadd(v, e2)%q
v = p.polydiv(v, xN_1)[1]%q
# print("v")
# print(v.astype(int))
```

```
# print()

v_r = randomized_round(v, q)

keystream_a = ModularRound(v_r, q).astype(int)

mask = CrossRound(v_r, q)
print("mask: "+str(mask.astype(int)))
print()
print()

w = p.polymul(u, s1)%q
w = p.polydiv(w, xN_1)[1]%q

keystream_b = rec(w, mask)
keystream_b = np.array(keystream_b).astype(int)


print("Alice's shared secret: " + str(keystream_a))
digest_a = ''.join([str(elem) for elem in keystream_a])
print("{0:0>4X}".format(int(digest_a, 2)))
print()

print("Bobby's shared secret: " + str(keystream_b))
digest_b = ''.join([str(elem) for elem in keystream_b])
print("{0:0>4X}".format(int(digest_b, 2)))
print()

comparison = keystream_a == keystream_b
equal_arrays = comparison.all()
if equal_arrays == True:
    print("Alice and Bobby's Shared Secret is Equal.")
else:
    print("Error!")
```

References

[1] National Security Agency *Commercial National Security Algorithm Suite*, https://apps.nsa.gov/iaarchive/programs/iad-initiatives/cnsa-suite.cfm, 2015.

[2] National Institute of Standards and Technology *Post-Quantum Cryptography Standardization*, https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization, 2017

[3] National Institute of Standards and Technology *Post-Quantum Cryptography, Round 2 Submissions*, https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions, 2020

[4] Neal Koblitz and Alfred J. Meneze *A Riddle Wrapped In An Enigma*, IEEE Security Privacy, vol. 14, no. 6, pp. 34-42, Dec. 2016.

[5] Jeremy Hsu *How the United States Is Developing Post-Quantum Cryptography*, IEEE Spectrum, https://spectrum.ieee.org/tech-talk/telecom/security/how-the-us-is-preparing-for-quantum-computings-threat-to-end-secrecy, 2019.

[6] Jeffery Hoffstein, Jill Pipher and Joseph H. Silverman *An Introduction To Mathematical Cryptography*, Springer, 2014.

[7] Stefano Ottolenghi *Homomorphic Signatures over Lattices*, Universit'a di Genova, 2019

[8] Lyubashevsky, Vadim *Towards practical lattice-based cryptography*, 2008

[9] Dong Pyo Chi, Jeong Woon Choi, Jeong San Kim and Taewan Kim *Lattice Based Cryptography for Beginners*, 2008

[10] Douglas Stebila *Introduction to post-quantum cryptography and learning with errors*, 2018

[11] Vadim Lyubashevsky, Chris Peikert, Oded Regev *On Ideal Lattices and Learning with Errors Over Rings*, 2013

[12] Martin R. Albrecht, Emmanuela Orsini, Kenneth G. Paterson,Guy Peer,Nigel P. Smart *Tightly Secure Ring-LWE Based Key Encapsulation With Short Ciphertexts*, 2017

[13] Vikram Singh *A Practical Key Exchange for the Internet using Lattice Cryptography*, 2008

[14] Vikram Singh, Arjun Chopra *Even More Practical Key Exchanges for the Internet using Lattice Cryptography*

[15] Steven D. Galbraith *Space-efficient variants of cryptosystems based on learning with errors*, University of Auckland