Intuitive Understanding of Quantum Computation and Post-Quantum Cryptography (Chapter 2)

Nguyen Thoi, Minh Quan *

Contents

2 Lattice-based cryptography

Lattice [1], [2] is a rare double-edged sword in cryptography. It can be used to build cryptographic protocols as well as to break them. Furthermore, lattice-based cryptography is assumed to be safe against quantum computers, while ECDH key exchange and RSA aren't. As quantum computers are more powerful than classical computers, can we deduce that lattice-based cryptography is safe against classical computers? No, we can't. It's because we start with a security assumption (not fact) which might turn out to be wrong. Like many other things in cryptography, lattice-based cryptography's security is an assumption, no one knows for sure. There is no evidence that lattice-based cryptography is safer than ECDH against classical computers. Therefore, it's better to deploy lattice-based key exchange in hybrid mode together with ECDH where the shared key is derived from both lattice and ECDH. Finally, lattice-based cryptography has extraordinary properties that no other cryptographic constructions have. It is used to build famous fully homomorphic encryption (FHE).[1]

Lattice-based cryptography often relies on the hardness of finding something small or solving equations with errors. Every time you see the words small and errors in the context of vectors, equations, polynomials, you know that you land in the realms of lattice-based cryptography.

Before studying lattice, let's warm up by solving Fermat's last equation $x^n + y^n = z^n$. I found a beautiful solution $z = \sqrt[n]{x^n + y^n}$. Sorry, I forgot that x, y, z must be integers :) All right, I found an integer solution as well $x = 0, z = y$:) What's wrong with me! I missed the conditions $x, y, z > 0$. The lessons learned are to pay attention to both the variables' domains and their constraints.

*https://www.linkedin.com/in/quan-nguyen-a3209817, https://scholar.google.com/citations?user=9uUqJ9IAAAAJ, msuntmquan@gmail.com

[1] I copied some paragraphs from my previous article [3] because I'm lazy :)

2.1 Lattice definitions

Definition Lattice [2] is a set of points $L = \{\sum_{i=1}^{i=n} a_i v_i | a_i \in \mathbb{Z}\}$ where v_i are linearly independent vectors in $\mathbb{R}^n$ and $B = \{v_1, \cdots, v_n\}$ is a base of L. We often write v_i in the form of $n \times 1$ column vector and $B = (v_1, \cdots, v_n)$ is a $n \times n$ matrix.

To understand this abstract definition, let's take a look at a concrete example where $v_1 = (2, 0), v_2 = (0, 2)$, i.e., v_1 and v_2 are twice unit vectors in x-axis and y-axis. The lattice L is the set of points $\{a_1 v_1 + a_2 v_2 = (2a_1, 2a_2) | a_1, a_2 \in \mathbb{Z}\}$, i.e., all even integer points in 2-dimensional space. This gives us a figure of lattice, but we haven't gone far in understanding the definition. One way to move forward is to question the details of the definition.

What if $\{v_1, \cdots, v_n\}$ are dependent vectors, e.g., $v_1 = v_2 = (0, 2)$? In this case, a single vector v_1 is enough to generate L, i.e., the linearly independent condition is to remove redundancy in $\{v_1, \cdots, v_n\}$.

What if a_i are real numbers, instead of integers? In this case, L is the whole 2-dimensional space regardless of the base $B = \{v_1, v_2\}$, i.e., L loses all its interesting math structure.

If we change $\{v_1, \cdots, v_n\}$, will L change? Not always. L is defined as a set of points, i.e., the points are the essence of L, not vectors $\{v_1, \cdots, v_n\}$ that are used to generate L. The vectors $\{v'_1 = (2, 0), v'_2 = (2, 2)\}$ ($v'_1 = v_1, v'_2 = v_1 + v_2$) also generate all even integer points in 2-dimensional space, i.e., the base $B = \{v_1, \cdots, v_n\}$ of L is not unique. In fact, if U is an integer matrix with determinant ± 1 then $B' = BU$ is another base of L. In our example, $U = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$, $det(U) = 1$ and $(v'_1, v'_2) = (v_1, v_1 + v_2) = (v_1, v_2) \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$.

We've made significant progress but we can go further by playing with the elements in the definition. If $a_i = 0$ then $0 = 0v_1 + \cdots + 0v_n$ is in L. If $x = a_1 v_1 + \cdots a_n v_n$ is in L then $-x = (-a_1)v_1 + \cdots + (-a_n)v_n$ is in L. If $x = a_1 v_1 + \cdots + a_n v_n$ and $x' = a'_1 v_1 + \cdots + a'_n v_n$ are in L then $x + x' = (a_1 + a'_1)v_1 + \cdots + (a_n + a'_n)v_n$ is also in L. The above properties make L an additive subgroup of $\mathbb{R}^n$. The question is whether any additive subgroup of $\mathbb{R}^n$ a lattice? When showing the above properties we haven't used the fact that a_i are integers at all. As mentioned above if a_i are real numbers then L contains all points in 2-dimensional space and the fact that a_i are integers make the lattice look discrete and not dense. Put everything together, we have an equivalent definition of lattice.

Definition Lattice is a discrete additive subgroup of $\mathbb{R}^n$.

You may wonder why I made an effort to introduce the 2nd definition. What's the point? In practical problems, no one is going to tell us that there is a lattice structure. We ourselves have to recognize the lattice structure hidden in the problem we're trying to solve. The 1st definition is not useful in this regard because we're not even sure yet there is a lattice, let alone know its basis vectors. The 2nd definition gives us signals to detect hidden lattice structure.

Let's look at the set of integer solutions of the following equation $2x_1 + 3x_2 = 0$. It's obvious that $(0, 0)$ is a solution. If (x_1, x_2) and (x'_1, x'_2) are solutions then $(-x_1, -x_2)$ and $(x_1 + x'_1, x_2 + x'_2)$ are solutions because $2(-x_1) + 3(-x_2) = -(2x_1 + 3x_2) = 0$ and $2(x_1 + x'_1) + 3(x_2 + x'_2) = (2x_1 + 3x_2) + (2x'_1 + 3x'_2) = 0 + 0 = 0$. The above properties make the set of solutions additive subgroup of $\mathbb{R}^n$ and the integer condition makes it discrete. Therefore, the set of solutions is a lattice. In conclusion, we know that the set of solutions forms a lattice without solving the equation, is it amazing? While this example is somewhat trivial, we'll use this observation in more complicated problems in later sections.

2.1.1 Successive minima

We briefly introduce a few concepts that we'll need in later sections.

$\lambda_1(L)$ denotes the length of a nonzero shortest vector in L. As 0 is always in L, in the definition of $\lambda_1(L)$, we have to add the nonzero condition to avoid trivial cases.

[2] Lattice has a deep theory which is out of my depth. If you want to understand lattice in depth, I recommend studying it by excellent formal sources [1], [2].

The ith successive minima $\lambda_i(L)$ is the smallest r such that L has i linearly independent vectors whose lengths are at most r.

2.2 Hard lattice computational problems

Public-key cryptography is based on hard computational problems. For instance, ECDH is based on hardness of discrete log problems while RSA is based on hardness of factoring problems. Similarly, lattice-based cryptography is based on hard lattice computational problems. Therefore, we briefly introduce the following hard lattice computational problems, mostly to introduce the terminologies.

Shortest Independent Vectors Problem (SIVP_γ): Given lattice $L(B)$ and small $\gamma > 0$, find n linearly independent lattice vectors whose length is at most $\gamma\lambda_n(L)$.

Bounded Distance Decoding Problems (BDD_γ): Given a lattice $L(B)$, small $\gamma > 0$ and a target point t that is guaranteed to be close to L (i.e. $\text{distance}(t, L) < d = \lambda_1(L)/\gamma$), find a unique lattice vector v such that $||v - t|| < d$.

For the purpose of this article, it's fine to just remember the following: for lattice, it's difficult to find short lattice vectors or to find a lattice vector close to a given target point.

2.3 Short Integer Solution (SIS)

SIS Problem Given m random vector $a_i \in \mathbb{Z}_q^n$ (i.e. a_i is a n-dimensional vector where each coordinate is integer $\mod q$), find nonzero solution $(z_1, \cdots, z_m) \in \{-1, 0, 1\}$ of system of linear equations $z_1a_1 + \cdots + z_ma_m = 0$. In matrix form $Az = 0$ where $A = (a_1, \cdots, a_m)$ is a $n \times m$ matrix.

Note that if there is no condition $z_i \in \{-1, 0, 1\}$ then we know how to solve this system of linear equations using Gaussian elimination.

Let's take a look at the set of solutions $\{(z_1, \cdots, z_m)\}$. $(0, \cdots, 0)$ is a trivial solution. If $(z_1, \cdots, z_m)$ and $(z'_1, \cdots, z'_m)$ are solutions then $(-z_1, \cdots, -z_m)$ and $(z_1 + z'_1, \cdots, z_m + z'_m)$ are solutions because $(-z_1)a_1 + \cdots + (-z_m)a_m = -(z_1a_1 + \cdots + z_ma_m) = 0$ and $(z_1 + z'_1)a_1 + \cdots + (z_m + z'_m)a_m = (z_1a_1 + \cdots + z_ma_m) + (z'_1a_1 + \cdots + z'_ma_m) = 0 + 0 = 0$. The above properties make the set of solutions $\{(z_1, \cdots, z_m\}$ an additive group and integer condition makes it discrete. Therefore, the set of integer solutions $\{(z_1, \cdots, z_m)\}$ is a lattice. How's about the condition $z_i \in \{-1, 0, 1\}$? This condition forces the vector $(z_1, \cdots, z_m)$ to be small. Consequently, solving SIS problems is similar to finding small vectors in the lattice of integer solutions. In other words, solving SIS is a hard problem. It's pretty cool as whenever we see a hard problem, we have hope to use it to build cryptographic protocols. In the next section, we'll use the hardness of SIS problem to construct collision-resistant hash function.

2.3.1 Collision-resistant hash functions based on SIS

Given a random matrix $A \in Z_q^{n \times m}$, the function $f_A(z) = Az$ where $z \in \{0, 1\}^m$ is a collision-resistant hash function.

We'll use proof by contradiction. If $f_A(z) = Az$ is not a collision-resistant hash function then we can find $z_1, z_2 \in \{0, 1\}^m$ such that $f_A(z_1) = f_A(z_2)$. This implies $Az_1 = Az_2$ or $A(z_1 - z_2) = 0$. As $z_1, z_2 \in \{0, 1\}^m$, we have $z_1 - z_2 \in \{-1, 0, 1\}^m$. It means that we found $z = z_1 - z_2 \in \{-1, 0, 1\}^m$ such that $Az = 0$. In other words, we can solve SIS problems (contradicts the fact that SIS problems are hard).

2.4 Learning With Errors (LWE)

Who came up with the name "learning with errors"[4]? Learning with accurate information is hard, let alone learning with misinformation and errors :) Maybe, it's the intention to make the attacker's life miserable.

LWE Problem Generate m random vectors $a_i \in Z_q^n$, small random errors e_i, a random secret $s \in Z_q^n$ and computes $b_i = \langle a_i, s \rangle + e_i \mod q \in Z_q$ [3]. Given (a_i, b_i) , find s.
In the matrix form, given $(A, b^t = s^t A + e^t \mod q)$ [4], find s where $A = (a_1 \cdots a_m)$ is an $n \times m$ matrix and $b = (b_1 \cdots b_m) \in Z_q^m$. A closely related problem is to not find s, but to distinguish $(A, b^t = s^t A + e^t \mod q)$ from random distribution.

Note that if there are no errors e_i then $b^t = s^t A$ is a system of linear equations which we know how to solve for s using Gaussian elimination.

Now, let's try to find our lattice in this LWE problem. If we remove errors e_i then if we fix matrix A, the set $L = \{s^t A \mod q\}$ forms a lattice. Why? $0 = 0A$ is in L. If $x_1 = s_1^t A$ and $x_2 = s_2^t A$ are in L then $-x_1$ and $x_1 + x_2$ are in L because $-x_1 = -s_1^t A = (-s_1^t)A$ and $x_1 + x_2 = s_1^t A + s_2^t A = (s_1^t + s_2^t)A$. The above properties make L additive group and the integer condition makes it discrete. Therefore, L is a lattice. In our LWE problem, $b^t = s^t A + e^t$ is a point close to the lattice point $s^t A$ because the difference between them is the small error vector e^t. Our task is given b^t, find the lattice point $s^t A$ close to it (note that knowing $s^t A$ is enough to find s^t using Gaussian elimination). This is the Bounded Distance Decoding (BDD_γ) problem which is hard to solve.

2.4.1 Ring-LWE

Define $R = Z_q[x]/(x^n + 1)$, i.e., polynomials of degree n where coefficients are integers $\mod q$ and operations on polynomials are $\mod x^n + 1$ (i.e. we can replace x^n with -1). The definition of Ring-LWE is similar to LWE, except instead of using Z_q^n, we use R.

Ring-LWE Problem Generate m random polynomials $a_i \in R$, small random error polynomials e_i, a random secret $s \in R$, computes $b_i = a_i s + e_i$. Given (a_i, b_i), find s.

What's the relationship between element of R with our familiar vector Z_q^n? Let's write down a polynomial $p(x)$ in R

$$p(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

The coefficients $(a_{n-1}, \cdots, a_1, a_0)$ is a vector in Z_q^n. One one hand, to a certain extent, we can cast Ring-LWE problems into LWE problems. On the other hand, polynomial $\mod (x^n+1)$ has more structure than Z_q^n. Let's briefly take a look at $p(x)x \mod x^n + 1$

$$\begin{aligned} p(x)x &= (a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0)x \\ &= a_{n-1}x^n + a_{n-2}x^{n-1} + \cdots + a_1 x^2 + a_0 x \\ &= a_{n-1}(-1) + a_{n-2}x^{n-1} + \cdots + a_1 x^2 + a_0 x \\ &= a_{n-2}x^{n-1} + \cdots + a_1 x^2 + a_0 x - a_{n-1} \end{aligned}$$

I.e. multiply $p(x)$ with x corresponds to transform from $(a_{n-1}, \cdots, a_1, a_0)$ to $(a_{n-2}, \cdots, a_1, a_0, -a_{n-1})$. Properties like this make implementation of Ring-LWE more efficient than LWE. However, whenever you have extra math structure, it might later come back and help cryptanalysis. There is an on-going debate about the trade-off between efficiency of Ring-LWE and its extra math structure to security.

2.5 Regev's LWE public key cryptosystem

In this section, we'll take a look at Regev's public key cryptosystem [4] that is based on the hardness of LWE problem. The protocol only encrypts a single bit μ but it can be generalized to encrypt arbitrary data. Note that Regev's encryption is only semantically secure, i.e., the ciphertext is indistinguishable from random distribution and it's safe against eavesdropper. It's not safe against active adversary, i.e., it's not safe against chosen ciphertext attack.

[3] The notation $\langle x, y \rangle$ denotes the inner product of 2 vectors $x = (x_1, \cdots, x_n), y = (y_1, \cdots, y_n)$, i.e., $x_1 y_1 + \cdots + x_n y_n$

[4] The notation s^t means the transpose of s

To build any public key cryptosystem, the first step is to construct private/public key pair. LWE's description gives us a hint on how to do it: Alice's private key is s and her public key is $(A, b^t = s^tA + e^t)$. As $(A, b^t = s^tA + e^t)$ is indistinguishable from random distribution, the public key (A, b^t) doesn't leak any information about the private key s. The question is how to use the public key (A, b^t) for encryption?

As semantic security concerns indistinguishability from random, we'll try to generate random alike traffic to confuse eavesdroppers. We need another math property: if $x \in \{0,1\}^m$ then (A, Ax) is indistinguishable from random distribution. Therefore, using Alice's public key $(A, b^t = s^tA + e^t)$, Bob can generate $x \in \{0,1\}^m$ and send (Ax, b^tx) to Alice. This is safe because (Ax, b^tx) is indistinguishable from random distribution. What can Alice do? Note that $b^tx = (s^tA + e^t)x = s^tAx + e^tx$ and Alice knows the private key s^t, so she can compute $b^tx - s^tAx = e^tx$. I.e., Alice can compute the noise e^tx (e^t and x are small), but the eavesdropper can't. Awesome, Bob can generate random-alike traffic (Ax, b^tx) and Alice can denoise it. We're happy, we make the eavesdropper confused, except we confuse ourselves as well because we haven't encrypted anything yet :)

We're close to the final protocol. The last piece of the puzzle is how to include the bit μ into the traffic. What will happen if Bob sends $(Ax, b^tx + \mu)$? Alice uses the same computation to get $b^tx - s^tAx = e^tx + \mu$. However, Alice doesn't know x, so while she can compute the noise $e^tx + \mu$, she can't extract μ from it. The idea to solve this puzzle is to encode the "meaning" of μ, instead of μ itself into the traffic: if $\mu = 0$, we doesn't include it into the traffic, but if $\mu = 1$ then we include large number $\mu\lfloor\frac{q}{2}\rceil$ into traffic. When Alice receives it, she computes $b^tx - s^tAx$ and if the result is small, she knows that μ was 0, otherwise if $b^tx - s^tAx$ is large, she knows that μ was 1. The complete protocol is shown in the below figure.

Alice		**Bob**
Secret s $\leftarrow_\$ Z_q^n$		$x \leftarrow_\$ \{0,1\}^m$
$b^t = s^tA + e^t$	$\xrightarrow{(A, b^t)}$	
		$u = Ax$
		$u' = b^tx + \mu\lfloor\frac{q}{2}\rceil$
	$\xleftarrow{(u, u')}$	
$u' - s^tu \approx \mu\lfloor\frac{q}{2}\rceil$		

To encrypt a bit μ, choose random $x \in \{0,1\}^m$, compute ciphertext as follow ($u = Ax, u' = b^tx + \mu\lfloor\frac{q}{2}\rceil$).

To decrypt, compute $u' - s^tu = b^tx + \mu\lfloor\frac{q}{2}\rceil - s^tAx = (s^tA + e^t)x + \mu\lfloor\frac{q}{2}\rceil - s^tAx = e^tx + \mu\lfloor\frac{q}{2}\rceil \approx \mu\lfloor\frac{q}{2}\rceil$ and test whether it is closer to 0 ($\mu = 0$ case) or $\lfloor\frac{q}{2}\rceil \mod q$ ($\mu = 1$ case).

2.6 Lattice cryptanalysis of subset sum problem

LLL algorithm [5] is a celebrated algorithm, there is a dedicated book just for LLL algorithm and its applications. For the purpose of the article, we'll use only one property of the LLL algorithm. Given a lattice $L(B)$ in $\mathbb{R}^n$ where B is a base of L, LLL algorithm will find a relatively short vector b: $||b|| < \beta^n\alpha_1(L)$ where $1 < \beta < 2$.

Using the above property, to solve certain equations, we'll prove the following:

- The solutions form a lattice.
- The solutions are short vectors.

After that we'll let LLL algorithm do its job to find short vectors (solutions) in the above lattice for us. Let's apply the described method to solve the following subset sum problem [6]:

given n positive numbers a_i and a positive number M, find $x_i \in \{0,1\}$ such that $a_1x_1 + \cdots + a_nx_n = M$.
Let's consider the following vectors

$$\begin{aligned} b_1 &= (1, 0, \cdots, 0, -a_1) \\ b_2 &= (0, 1, \cdots, 0, -a_2) \\ &\vdots \\ b_n &= (0, 0, \cdots, 1, -a_n) \\ b_{n+1} &= (0, 0, \cdots, 0, M) \end{aligned}$$

We have

$$\begin{aligned} & x_1b_1 + x_2b_2 + \cdots + x_nb_n + 1b_{n+1} \\ &= (x_1, 0, \cdots, 0, -a_1x_1) \\ &+ (0, x_2, \cdots, 0, -a_2x_2) \\ &\vdots \\ &+ (0, 0, \cdots, x_n, -a_nx_n) \\ &+ (0, 0, \cdots, 0, M) \\ &= (x_1, x_2, \cdots, x_n, -a_1x_1 - a_2x_2 + \cdots - a_nx_n + M) \\ &= (x_1, x_2, \cdots, x_n, 0) \end{aligned}$$

I.e., $(x_1, x_2, \cdots, x_n, 0)$ is a vector in the lattice with basis vectors $b_1, \cdots, b_{n+1}$. Furthermore, $(x_1, x_2, \cdots, x_n, 0)$ is a short vector because $x_i \in \{0,1\}$. Therefore, LLL algorithm will help find the short vector (solution) $(x_1, x_2, \cdots, x_n, 0)$ for us.

2.7 Lattice-based homomorphic encryption

Let's assume we have data that we store in a cloud. To protect our data, we encrypt them and keep the key to ourselves. On the other hand, we want to take advantage of cloud's computing power, so we want the cloud to compute on our ciphertexts without knowing what our plaintexts are. Homomorphic encryption is a special type of encryption that achieves the previous goal. In this section, we'll describe a simple lattice-based homomorphic encryption [7].

We'll use $\mathbb{Z}_q[x]/(x^{2^k}+1)$ (q is a prime number), i.e., polynomials whose coefficients are in $\mathbb{Z}_q$ and all operations are $\mod x^{2^k}+1$. We also use a small modulus t that is much smaller than q. Note that everything in this section including secret key, message, ciphertext are polynomials.

The secret key is a polynomial s in $\mathbb{Z}_q[x]/(x^{2^k}+1)$.

To encrypt a message polynomial $m \in \mathbb{Z}_t[x]/(x^{2^k}+1)$, we randomly generate polynomial a, small error polynomial e and the ciphertext is simply $c = Enc(m) = (c_0, c_1) = (-a, as+m+et)$.

To decrypt a ciphertext $c = (c_0, c_1)$, we compute $c_1 + c_0s \mod t = as + m + et - as \mod t = m + et \mod t = m$.

To see how this encryption is additive homomorphic, let's take a look at two encryptions of m and m': $c = Enc(m) = (c_0, c_1) = (-a, as + m + et)$ and $c' = Enc(m') = (c'_0, c'_1) = (-a', a's + m' + e't)$. If we add (c_0, c_1) and (c'_0, c'_1) together, we have:

$$\begin{aligned} (c_0, c_1) + (c'_0, c'_1) &= (c_0 + c'_0, c_1 + c'_1) \\ &= (-a - a', as + m + et + a's + m' + e't) \\ &= (-(a + a'), (a + a')s + (m + m') + (e + e')t \end{aligned}$$

If we denote $a'' = a + a'$, $m'' = m + m'$, $e'' = e + e'$, then we see that $c'' = (c''_0, c''_1) = (c_0, c_1) + (c'_0, c'_1)$ is the encryption of $m'' = m + m'$ with error $e'' = e + e'$. To recap, what we've done is to add 2 ciphertexts together without knowing the messages, but the result corresponds to the sum of the messages. It's pretty cool, right?

Another nice property is that if you multiply a polynomial p to the encryption of m then the result corresponds to encryption of pm. To see why it's the case, let's take a look at polynomial p and encryption of m: $(c_0, c_1) = (-a, as + m + et)$. We have:

$$\begin{aligned} p(c_0, c_1) &= (pc_0, pc_1) \\ &= (p(-a), p(as + m + et)) \\ &= (-pa, pas + pm + pet) \end{aligned}$$

If we denote $a' = -pa$, $m' = pm$, $e' = pe$, then we see that $(c'_0, c'_1) = p(c_0, c_1)$ is the encryption of $m' = pm$ with error $e' = pe$.

The final note is that the error increases in both cases. For lattice-based cryptography to work, the error must be small. Therefore, various techniques have been designed to reduce the error over time. We won't discuss error reduction techniques here, instead, we'll take a look at an awesome application of the previous homomorphic encryption in the next section.

2.8 Lattice-based private information retrieval

Let's say a server has a public database (e.g. movies, songs, lyrics, stories, books) with n items $x_1, \cdots, x_n$. A user wants to see a single item x_i at index i from the database without revealing to the server what item has been downloaded. This is to protect the user's privacy. An obvious solution is the user downloads all n items from the database. This has perfect privacy, but it costs significant bandwidth and user's local storage. We'll trade CPU with bandwidth and storage using the above homomorphic encryption [8]. The basic protocol works as follows.

The user forms a sequence of 0 and 1 where only at index i, it's 1 while the remaining numbers are 0: $0, \cdots, 0, \underbrace{1}_{\text{index i}}, 0, \cdots, 0$. The user uses homomorphic encryption to encrypt the above sequence, i.e., $c_1 = Enc(0), \cdots, c_{i-1} = Enc(0), c_i = Enc(1), c_{i+1} = Enc(0), \cdots, c_n = Enc(0)$. The user sends $c_1, \cdots, c_n$ to the server.

The server computes $x = x_1c_1 + \cdots + x_nc_n$ without knowing what i is and sends x to the user.

The user decrypts x and the result is x_i. Why's that? By homomorphic property, $x = x_1c_1 + \cdots + x_nc_n$ corresponds to the encryption of $x_1.0 + \cdots + x_{i-1}.0 + x_i.1 + x_{i+1}.0 + \cdots + x_n.1 = 0 + \cdots + 0 + x_i + 0 + \cdots + 0 = x_i$.

References

[1] The 2nd biu winter school. `https://cyber.biu.ac.il/event/the-2nd-biu-winter-school/`.

[2] Chris Peikert. A decade of lattice cryptography.

[3] Nguyen Thoi Minh Quan. Intuitive advanced cryptography.

[4] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography.

[5] A. K. Lenstra, Jr. Lenstra, H. W., and L. Lovász. Factoring polynomials with rational coefficients.

[6] J.C. Lagarias and A.M. Odlyzko. solving low density subset sum problems.

[7] Simple homomorphic encryption library with lattices (shell) (https://github.com/google/shell-encryption).

[8] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. Xpir : Private information retrieval for everyone.