Post-quantum cryptography

Why Post-Quantum Cryptography?

- Modern cryptography such as RSA and ECC relies on mathematical problems that are hard for classical computers.
- Quantum computers introduce new algorithms (for example Shor’s algorithm) capable of solving these problems efficiently.
- Post-quantum cryptography aims to develop cryptographic systems resistant to both classical and quantum attacks.
- NIST has standardized ML-KEM, a key encapsulation mechanism based on CRYSTALS-Kyber, as part of its post-quantum cryptography standards.

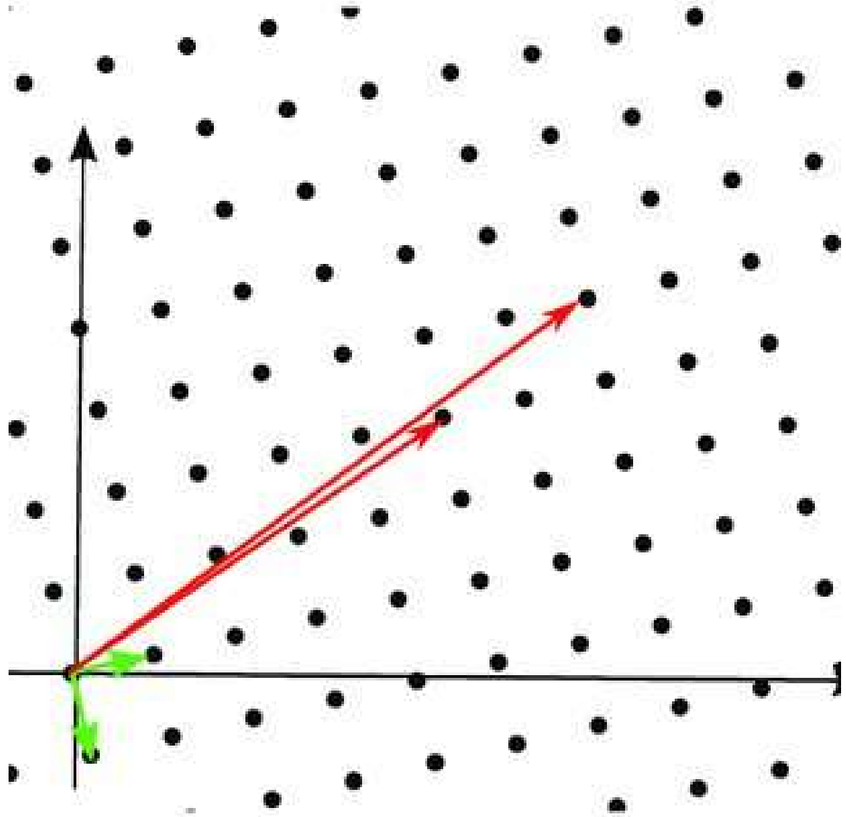
Lattice-based hard problems

Lattice-based problems involve finding solutions to problems within a grid-like structure in *n*-dimensional space.

These problems are believed to be hard even against quantum algorithms.

Main hard lattice-based problems are:

- **Shortest Vector Problem (SVP):** Find the shortest non-zero vector in a lattice.
- **Closest Vector Problem (CVP):** Find the closest lattice vector to a target point.



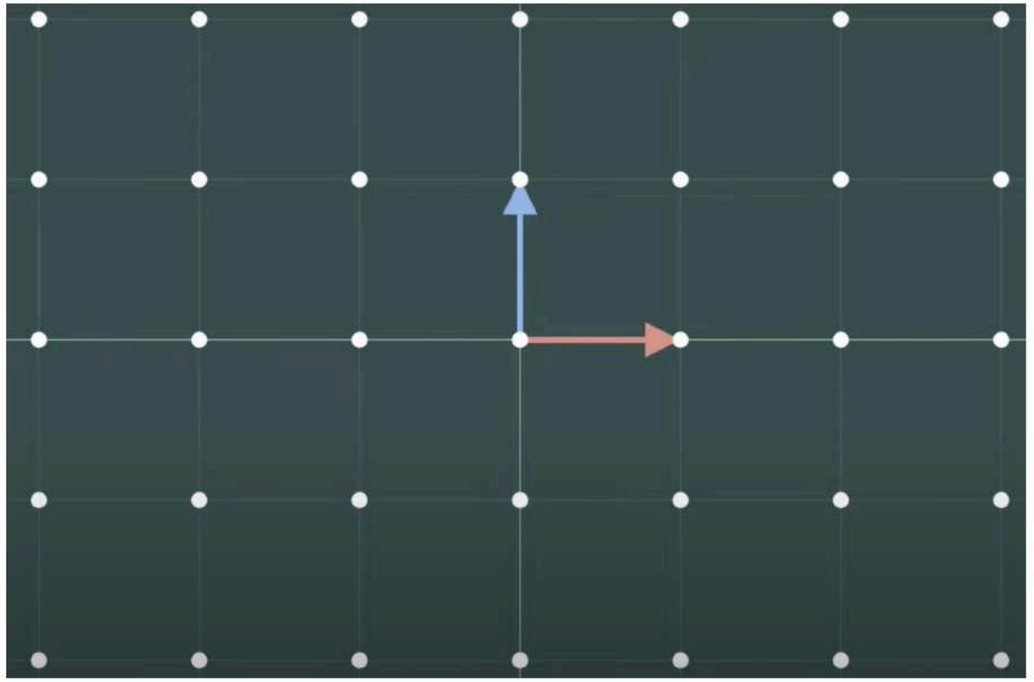
Lattice-based hard problems

A lattice $\mathcal{L}$ of $\mathbb{R}^n$ is by definition a discrete subgroup of $\mathbb{R}^n$. We will consider only integer lattices i.e. $\mathcal{L} \subset \mathbb{Z}^n$.

A basis of $\mathcal{L}$ is an ordered set $\mathbb{B} = (b_1, b_2, ..., b_n)$ such that

$$\mathcal{L} = \{\sum_{i=1}^{n} c_i b_i \ : \ c_i \in \mathbb{Z}\}.$$

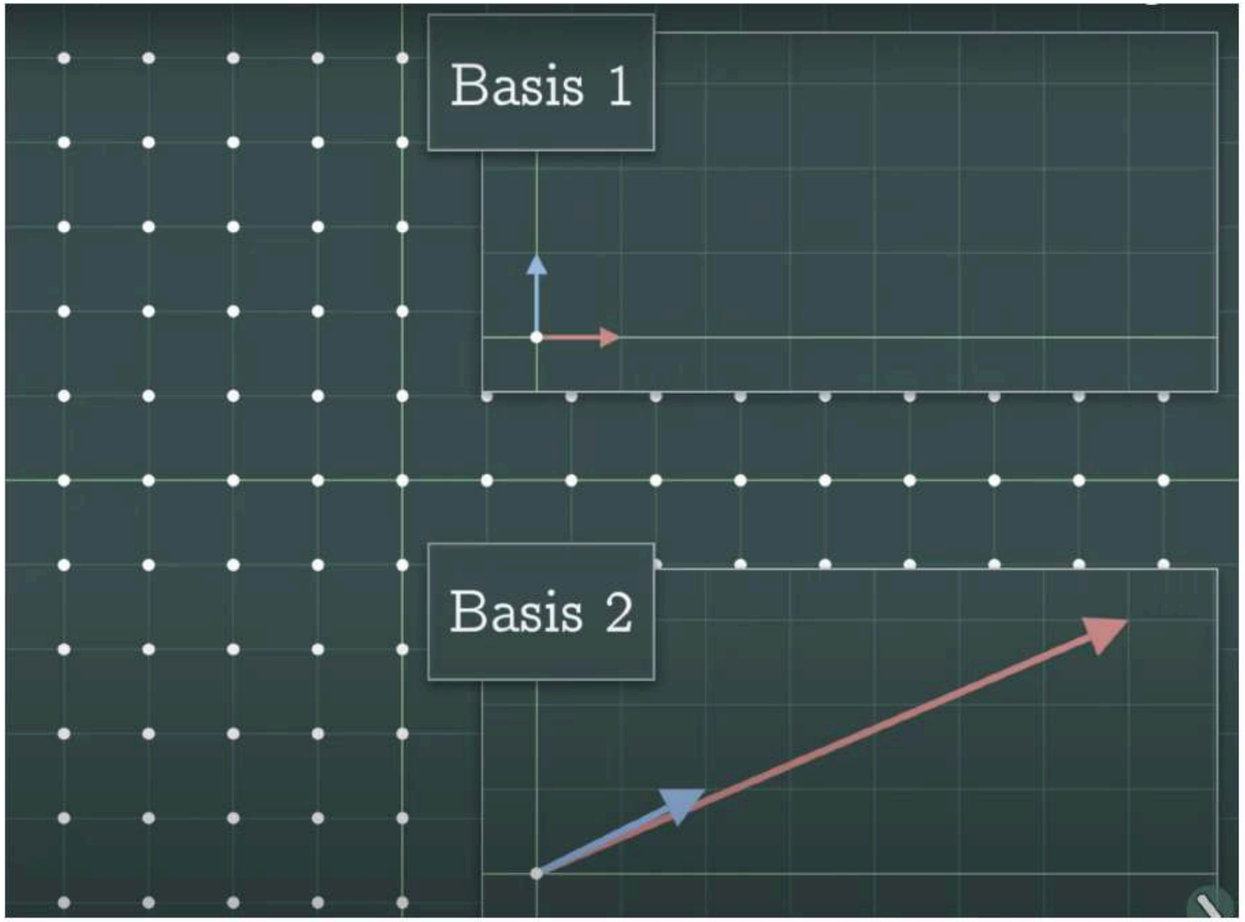
The picture shows lattice in a two-dimensional space whose base is made up of vectors marked in red and blue. By linear combination with integer coefficients of the base vectors, we get the lattice shown in the picture.



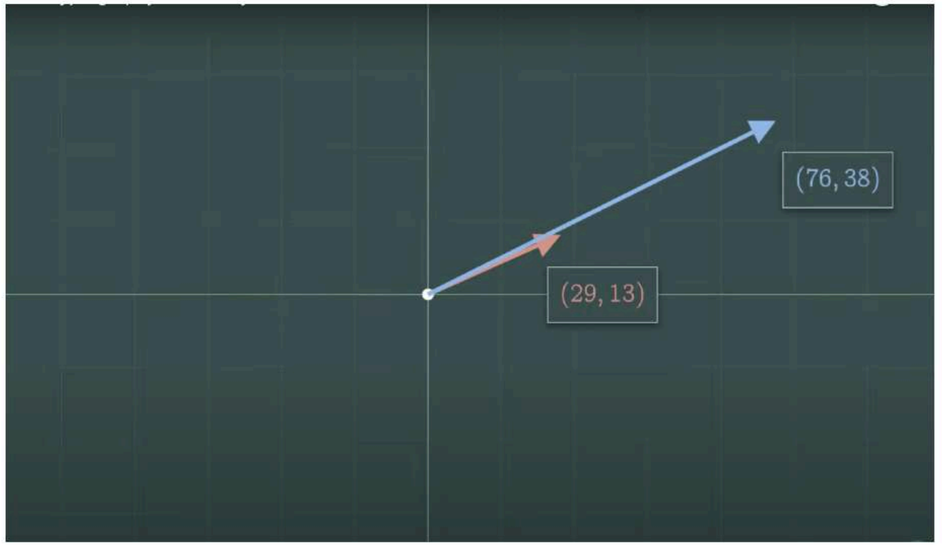
What kind of lattice will we get if we change the base vectors?

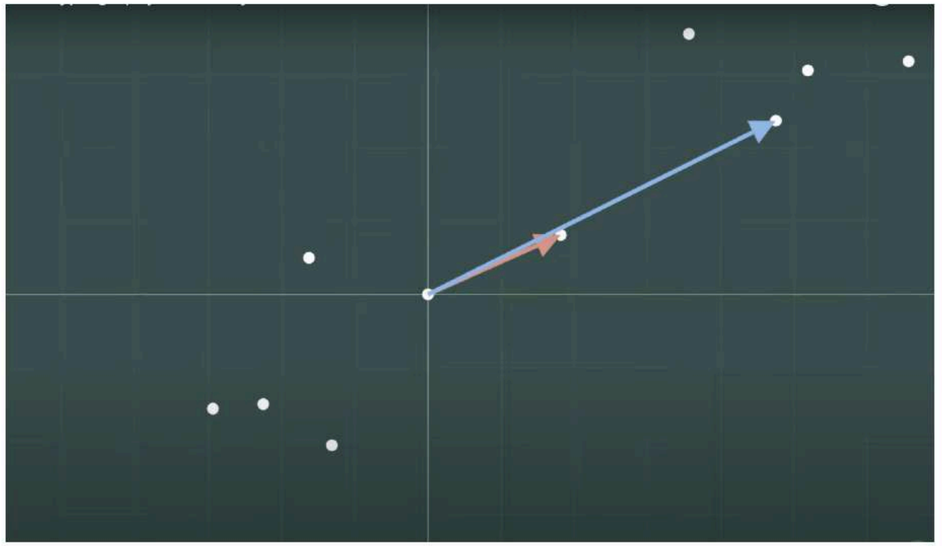
Lattice-based hard problems

Two different bases can generate the same lattice.



Let the basis vectors be given: $(29, 13)$ and $(76, 38)$. What does the lattice of which these two vectors are base vectors look like? By adding or subtracting the base vectors, we get the points that make up the lattice.

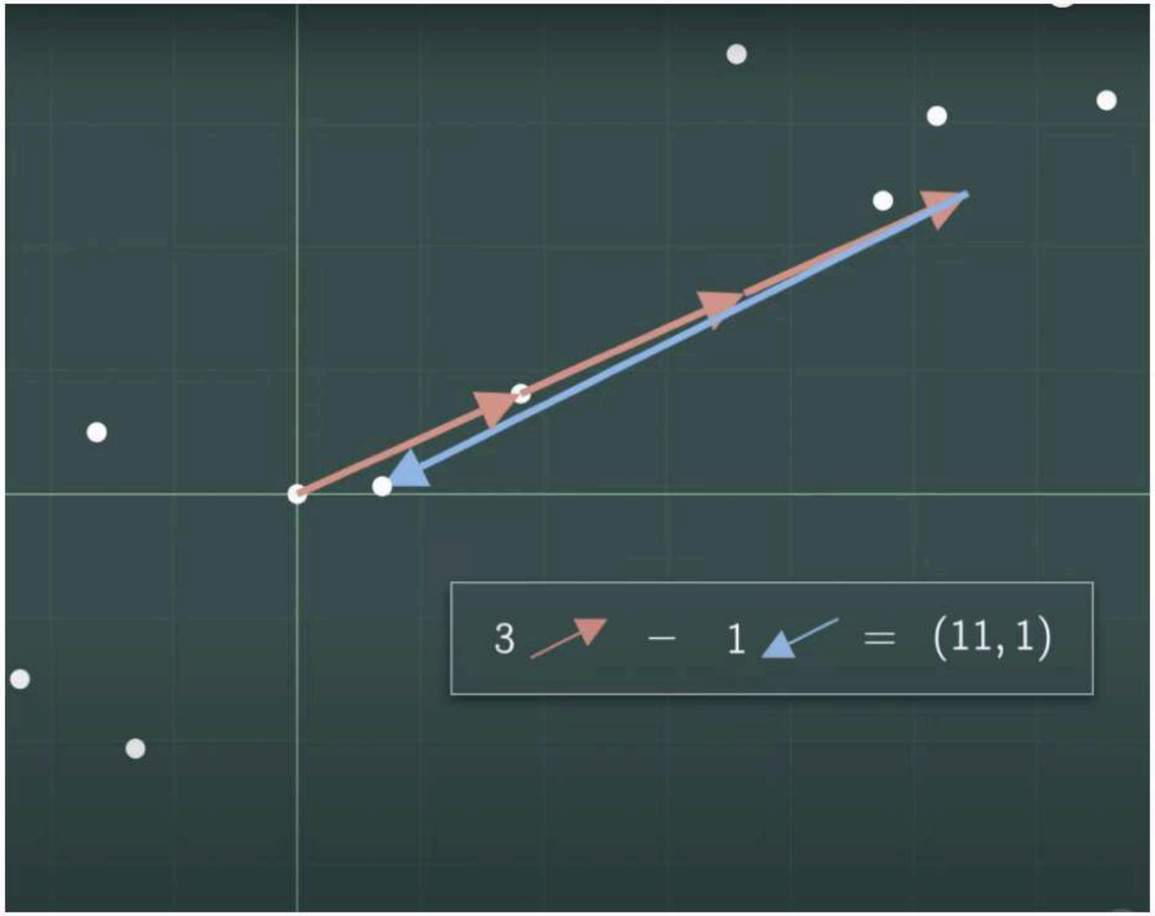




However, here it is not clear which points make up the lattice, unlike the previous situation when we had a clear picture of what the lattice looks like.

Let us now consider a different problem. We are interested in which non-zero lattice point is closest to the coordinate origin.

If we extend the red vector three times and subtract the blue one, we get a grid point with coordinates $(11, 1)$. Is this point closest to the coordinate origin?



We are certainly not sure of the answer. With a bit of luck, we can notice that if we add the red vector 8 times and subtract the blue vector three times, we get a point with coordinates $(4, -10)$. This point is certainly closer to the origin than the point $(11, 1)$.

Shortest Vector Problem and Closest Vector Problem

We see that the problem of determining the lattice point closest to the coordinate origin is not an easy one, even in two dimensions.

"Find the lattice point which is closest to the origin"

is called the "Shortest Vector Problem". In higher dimensions this problem becomes a very difficult problem.

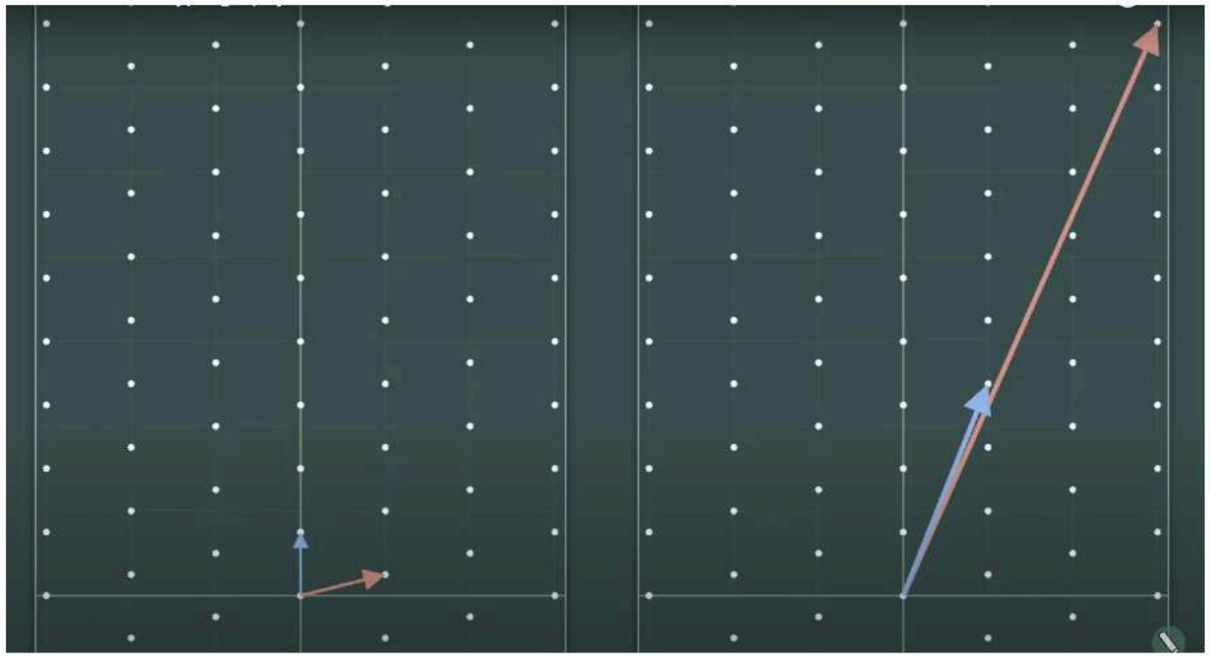
The "Shortest Vector Problem" is similar to "Closest Vector Problem":

"Find the lattice point which is closest to particular point in space."

These problems belong to "lattice problems" which experts believe that are hard problems, especially in hundred of dimesions. It would take a computer a very long time to solve the shortest and the closest vector problems. Computer scientist believe that it would be very difficult for quantum computer to solve these problems too.

Closest Vector Problem - encryption

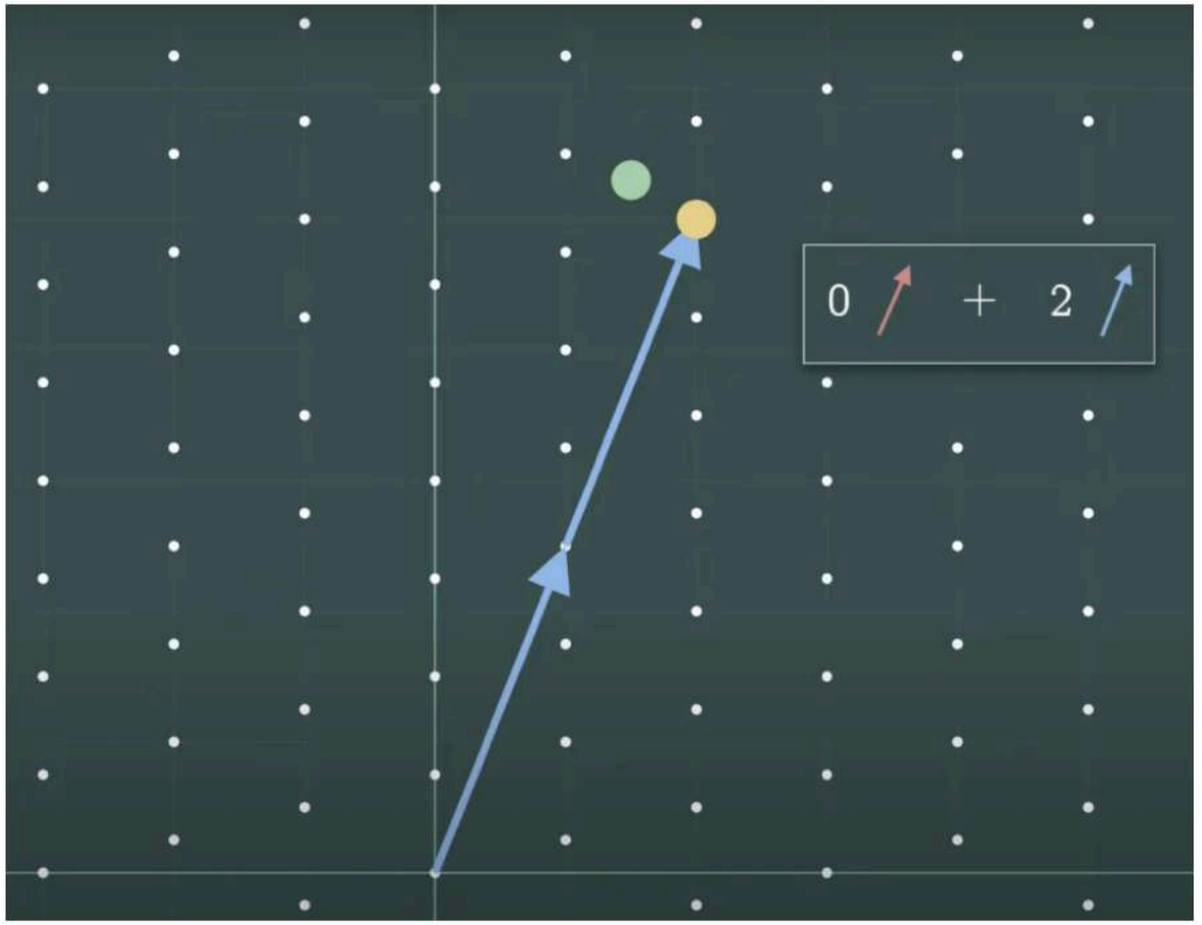
Bob wants to send Alice a message. Alice need to generate lattice with two different bases: a "good" basis and a "bad" basis.



The closest vector problem is more difficult to solve in a bad basis than in a good one.

Alice keeps the good base as a secret. The good basis vectors represent Alice's private key. Her public key is bad basis vectors.

Bob picks a point on the lattice (YELLOW) which represents his message. Then he moves a little bit away from that lattice point to create a new point (GREEN).



Bob sends Alice the coordinates of the green point. Alice knows the secret good basis, and in this basis it is easy to find the closest lattice point (yellow) and recover the message. In a bad basis, it is much more difficult to find the closest point to the green, especially with the increasing dimensions of the lattice. In a sufficiently large dimension, that problem is hard for both classical and quantum computers.

Learning With Errors (LWE)

Find $s \in \mathbb{Z}_q^n$ given noisy inner products

$$a_1 \in \mathbb{Z}_q^n \ : \ b_1 =< a_1, s > +e_1$$

$$a_2 \in \mathbb{Z}_q^n \ : \ b_2 =< a_2, s > +e_2$$

$$\vdots$$

$$a_m \in \mathbb{Z}_q^n \ : \ b_m =< a_m, s > +e_m$$

where e_i are from Gaussian distribution over $\mathbb{Z}$.

You can find more information on these links:

[Link 1 >]
[Link 2 >]

Without Errors - example

Bob has secret vector $s = (10, 82, 50, 5)$. This secret vector is his private key. To obtain public key Bob will multiply his private key with a public matrix A to obtain his public key b i.e. $b = As$:

$$77 * 10 + 7 * 82 + 28 * 50 + 23 * 5 = 2859$$

$$21 * 10 + 19 * 82 + 30 * 50 + 48 * 5 = 3508$$

$$4 * 10 + 24 * 82 + 33 * 50 + 38 * 5 = 3848$$

$$8 * 10 + 20 * 82 + 84 * 50 + 61 * 5 = 6225$$

$$6 * 10 + 53 * 82 + 1 * 50 + 86 * 5 = 4886$$

$$42 * 10 + 86 * 82 + 31 * 50 + 8 * 5 = 9062$$

$$5 * 10 + 24 * 82 + 79 * 50 + 27 * 5 = 6103$$

$$16 * 10 + 7 * 82 + 35 * 50 + 21 * 5 = 2589$$

$$56 * 10 + 18 * 82 + 25 * 50 + 58 * 5 = 3576$$

$$4 * 10 + 55 * 82 + 73 * 50 + 13 * 5 = 8265.$$

So, Bob's public key is

$$b = (2859, 3508, 3848, 6225, 4886, 9062, 6103, 2589, 3576, 8265).$$

Learning Without Errors - example

Since we have only 4 unknowns, we can choose only 4 equations (note that the equations are linear) and solve this system:

$$77 * x + 7 * y + 28 * z + 23 * t = 2859$$

$$21 * x + 19 * y + 30 * z + 48 * t = 3508$$

$$4 * x + 24 * y + 33 * z + 38 * t = 3848$$

$$8 * x + 20 * y + 84 * z + 61 * t = 6225.$$

Solving the system will give us Bob's private key, which means that this algorithm is not secure. For this reason, we add errors.

Learning With Errors - example

Let Bob has the same private key, as in the previous example, $s = (10, 82, 50, 5)$. To obtain his public key, Bob will multiply his private key with same public matrix A and add private error vector e:

$$b = As + e$$

i.e.

$$77 * 10 + 7 * 82 + 28 * 50 + 23 * 5 \sim 2859 + (-3)$$

$$21 * 10 + 19 * 82 + 30 * 50 + 48 * 5 \sim 3508 + 2$$

$$4 * 10 + 24 * 82 + 33 * 50 + 38 * 5 \sim 3848 + (-1)$$

$$8 * 10 + 20 * 82 + 84 * 50 + 61 * 5 \sim 6225 + 0$$

$$6 * 10 + 53 * 82 + 1 * 50 + 86 * 5 \sim 4886 + 4$$

$$42 * 10 + 86 * 82 + 31 * 50 + 8 * 5 \sim 9062 + (-1)$$

$$5 * 10 + 24 * 82 + 79 * 50 + 27 * 5 \sim 6103 + (-2)$$

$$16 * 10 + 7 * 82 + 35 * 50 + 21 * 5 \sim 2589 + 2$$

$$56 * 10 + 18 * 82 + 25 * 50 + 58 * 5 \sim 3576 + 0$$

$$4 * 10 + 55 * 82 + 73 * 50 + 13 * 5 \sim 8265 + (-1).$$

Now we have a system of 4 unknowns, but which usually has no solution because errors have been added, and the number of equations is greater than the number of unknowns.

Learning With Errors - example

$$77x + 7y + 28z + 23t = 2856$$
$$21x + 19y + 30z + 48t = 3510$$
$$4x + 24y + 33z + 38t = 3847$$
$$8x + 20y + 84z + 61t = 622$$
$$6x + 53y + 1z + 86t = 4890$$
$$42x + 86y + 31z + 8t = 9061$$
$$5x + 24y + 79z + 27t = 6101$$
$$16x + 7y + 35z + 21t = 2591$$
$$56x + 18y + 25z + 58t = 3576$$
$$4x + 55y + 73z + 13t = 8264$$

The error-adding procedure made the problem difficult. To make this problem even more difficult, we use modular arithmetic (for example we use mod 89):

$$77x + 7y + 28z + 23t = 2856 \bmod 89 = 8$$
$$21x + 19y + 30z + 48t = 3510 \bmod 89 = 39$$
$$4x + 24y + 33z + 38t = 3847 \bmod 89 = 20$$
$$8x + 20y + 84z + 61t = 6225 \bmod 89 = 84$$
$$6x + 53y + 1z + 86t = 4890 \bmod 89 = 84$$
$$42x + 86y + 31z + 8t = 9061 \bmod 89 = 72$$
$$5x + 24y + 79z + 27t = 6101 \bmod 89 = 49$$
$$16x + 7y + 35z + 21t = 2591 \bmod 89 = 10$$
$$56x + 18y + 25z + 58t = 3576 \bmod 89 = 16$$
$$4x + 55y + 73z + 13t = 8264 \bmod 89 = 76.$$

Module-LWE

Let $R = \mathbb{Z}[X]/(X^n + 1)$ for n a power of 2 and let $R_q = R/qR$, for odd prime number q. Notice that:

1) elements of R_q are polynomials with degree less than n and with the coefficients less than q;

2) operations in R_q are very efficient using Number Theoretic Transform (NTT).

Module-LWE

The Module-LWE search problem is to recover

$$s \in R_q^k$$

from

$$t = As + e,$$

where

$$A \in R_q^{k \times k}, \qquad s, e, t \in R_q^k.$$

The entries of A, s, e, and t are polynomials in R_q, and e is sampled from a small error distribution.

For $k > 1$, this is Module-LWE; for $k = 1$, it becomes Ring-LWE.

Addition and multiplication - examples

$$\text{Let } q = 5, n = 4, k = 2. \quad \text{Let } f(x) := \begin{bmatrix} x^2+x+1 \\ 3x^2+4x+1 \end{bmatrix} \text{ and } q(x) := \begin{bmatrix} 2x^2+2x+1 \\ 2x^3+4x+1 \end{bmatrix}.$$

$$f(x)+q(x) = \begin{bmatrix} x^2+x+1 \\ 3x^2+4x+1 \end{bmatrix} + \begin{bmatrix} 2x^2+2x+1 \\ 2x^3+4x+1 \end{bmatrix} = \begin{bmatrix} 3x^2+3x+2 \\ 2x^3+3x^2+8x+2 \end{bmatrix} =$$

$$= \begin{bmatrix} 3x^2+3x+2 \\ 2x^3+3x^2+3x+2 \end{bmatrix}.$$

Addition and multiplication - examples

$$\text{Let } q = 5, n = 4, m = 2. \text{ Let } A(x) := \begin{bmatrix} x^2 + x + 1 & 3x^2 + 4x + 1 \\ 2x^2 + 2x + 1 & x + 2 \end{bmatrix} \text{ and } s(x) := \begin{bmatrix} 2x^2 + 1 \\ 2x^3 + 2x \end{bmatrix}.$$

$$A(x) \cdot s(x) = \begin{bmatrix} (x^2 + x + 1)(2x^2 + 1) + (3x^2 + 4x + 1)(2x^3 + 2x) \\ (2x^2 + 2x + 1)(2x^2 + 1) + (x + 2)(2x^3 + 2x) \end{bmatrix} =$$

$$= \begin{bmatrix} 6x^5 + 10x^4 + 10x^3 + 11x^2 + 3x + 1 \\ 6x^4 + 8x^3 + 6x^2 + 6x + 1 \end{bmatrix} =$$

$$= \begin{bmatrix} -6x - 10 + 10x^3 + 11x^2 + 3x + 1 \\ -6 + 8x^3 + 6x^2 + 6x + 1 \end{bmatrix} = \begin{bmatrix} 10x^3 + 11x^2 - 3x - 9 \\ 8x^3 + 6x^2 + 6x - 5 \end{bmatrix} =$$

$$= \begin{bmatrix} x^2 + 2x + 1 \\ 3x^3 + x^2 + x \end{bmatrix}$$

Module-LWE vs Ring-LWE

Ring-LWE is a special case of the Module-LWE problem where the dimension of the matrices A_i over the ring R_q is 1×1. Varying the hardness (and security) of an Ring-LWE scheme therefore requires to change the dimension of the ring, whereas in Module-LWE, the ring is always the same and the dimension of the module is being varied. One advantage of Module-LWE is that we only need to have one good implementation for operations over the ring; varying the dimension of the module simply involves doing more (or fewer) of the same ring operations. Changing the ring, on the other hand, would require completely reimplementing all the operations. Another advantage of working with a constant-degree "small" ring is that it enables more fine-grained trade-offs between performance and security. The simplest and most efficient way of implementing Ring-LWE is to work over rings $\mathbb{Z}[X]/(X^n + 1)$ where n is a power of 2. Since n is the only parameter that determines the efficiency and security of Ring-LWE schemes, limiting it to powers of 2 may require overshooting the needed security bound. For example, the dimension of Kyber768 is not reachable, because 768 is not a power of 2.

Kyber

Kyber is a quantum-safe Key Encapsulation Mechanism (KEM). Kyber's security is based on the hardness of the Module-LWE problem, which is reducible to approximate shortest vector problem (SVP). So recovering a message from a ciphertext (or a private from a public key), should be as hard as solving a large SVP instance. This should be computationally infeasible, even for a quantum computer.

Kyber comes in three security levels, with different key size (in bytes):

Version	Security Level	Private Key Size	Public Key Size	Ciphertext Size
Kyber512	AES128	1632	800	768
Kyber768	AES192	2400	1184	1088
Kyber1024	AES256	3168	1568	1568

You can find more information on these links:

Link 1 >
Link 2 >

Kyber

Kyber parameter set is defined by the variables:

- n: maximum degree of the used polynomials;
- k: number of polynomials per vector;
- q: modulus for numbers;
- η_1, η_2: control how big coefficients of "small" polynomials can be;
- d_u, d_v: control how much $(\mathbf{u}, v)$ get compressed;
- δ: shows the probability of a decryption yielding a wrong result.

The parameter sets running for standardization use the following variables:

Name	**n**	**k**	**q**	$\boldsymbol{\eta_1}$	$\boldsymbol{\eta_2}$	$\mathbf{d_u}$	$\mathbf{d_v}$	$\boldsymbol{\delta}$
Kyber512	256	2	3329	3	2	10	4	2^{-139}
Kyber768	256	3	3329	2	2	10	4	2^{-164}
Kyber1024	256	4	3329	2	2	11	5	2^{-174}

Kyber - example

- $n = 4$
- $k = 2$
- $q = 17$

Key Generation

In our example, we use the private key:

$$\mathbf{s} = (-x^3 - x^2 + x,\ -x^3 - x)$$

A Kyber *public key* consists of two elements: a matrix of random polynomials $\mathbf{A}$ and a vector of polynomials $\mathbf{t}$. For our example, we use:

$$\mathbf{A} = \begin{pmatrix} 6x^3 + 16x^2 + 16x + 11 & 9x^3 + 4x^2 + 6x + 3 \\ 5x^3 + 3x^2 + 10x + 1 & 6x^3 + x^2 + 9x + 15 \end{pmatrix}$$

To calculate $\mathbf{t}$, we need an additional error vector $\mathbf{e}$. This error vector consists of polynomials with small coefficients, just like the private key. In our example, we use the error vector:

$$\mathbf{e} = (x^2,\ x^2 - x)$$

Now, we calculate $\mathbf{t}$ by matrix multiplication and addition:

$$\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$$

$\mathbf{t}$ will be a vector of polynomials:

$$\mathbf{t} = (16x^3 + 15x^2 + 7,\ 10x^3 + 12x^2 + 11x + 6)$$

Encryption

The encryption procedure uses error terms $\mathbf{e}_1$ and e_2, and a randomizer $\mathbf{r}$. These values are freshly generated for every encryption. The polynomials within $\mathbf{e}_1$, e_2, and $\mathbf{r}$ have small coefficients, just like the polynomials in $\mathbf{s}$.

In our example, we use:

$$\mathbf{r} = (-x^3 + x^2, x^3 + x^2 - 1)$$

$$\mathbf{e}_1 = (x^2 + x, x^2)$$

$$e_2 = -x^3 - x^2$$

Now, to encrypt a message, we first turn it into a polynomial. We do so by using the message's binary representation. Every bit of the message is used as a coefficient. In our example, we want to encrypt the number 11. Eleven has binary representation 1011. Therefore, our message polynomial is:

$$m_b = 1x^3 + 0x^2 + 1x + 1 = x^3 + x + 1.$$

Before encryption, we upscale m_b by multiplying it with

$$\left\lceil \frac{q}{2} \right\rceil.$$

Since in our example $q = 17$, we have

$$\left\lceil \frac{q}{2} \right\rceil = 9.$$

This is done because the coefficients of the message polynomial need to be far from 0, so that they can still be recovered after small noise is added. Thus, the scaled message is:

$$m = 9x^3 + 9x + 9.$$

We encrypt m using the public key $(\mathbf{A}, \mathbf{t})$. The encryption procedure calculates two values $(\mathbf{u}, v)$:

$$\mathbf{u} = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1,$$

$$v = \mathbf{t}^T\mathbf{r} + e_2 + m.$$

In our example, these values are:

$$\mathbf{u} = (11x^3 + 11x^2 + 10x + 3,\ 4x^3 + 4x^2 + 13x + 11),$$

$$v = 8x^3 + 6x^2 + 9x + 16.$$

Kyber ciphertexts consist of these two values:

$$(\mathbf{u}, v).$$

Decryption

First, we compute a noisy version of the message:

$$m_n = v - \mathbf{s}^T\mathbf{u}.$$

This result is noisy because the computation does not yield the original scaled message m exactly. We can see this by substituting the expressions for $\mathbf{u}$, v, and $\mathbf{t}$:

$$m_n = v - \mathbf{s}^T\mathbf{u} = \mathbf{t}^T\mathbf{r} + e_2 + m - \mathbf{s}^T(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1).$$

Since

$$\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e},$$

we have

$$\mathbf{t}^T = \mathbf{s}^T\mathbf{A}^T + \mathbf{e}^T.$$

Therefore,

$$m_n = (\mathbf{s}^T\mathbf{A}^T + \mathbf{e}^T)\mathbf{r} + e_2 + m - \mathbf{s}^T(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1).$$

After cancellation, we obtain

$$m_n = m + \mathbf{e}^T\mathbf{r} + e_2 - \mathbf{s}^T\mathbf{e}_1.$$

Now it becomes apparent why we needed to scale the message. All other terms in the equation are small error terms. Therefore, the coefficients of m_n are either close to

$$\left\lceil \frac{q}{2} \right\rceil = 9,$$

which represents a binary coefficient 1, or close to 0 modulo q, which represents a binary coefficient 0.

In our example, we get:

$$m_n = 8x^3 + 14x^2 + 8x + 6.$$

We recover the original scaled message by checking whether each coefficient of m_n is closer to 9 or to 0 modulo 17. Equivalently, being close to 17 also means being close to 0 modulo 17.

- 8 is closer to 9 than to 0, so we round it to 9.
- 14 is closer to 17, i.e. to 0 modulo 17, so we round it to 0.
- 8 is closer to 9 than to 0, so we round it to 9.
- 6 is closer to 9 than to 0, so we round it to 9.

Thus, we recover the scaled message polynomial:

$$9x^3 + 0x^2 + 9x + 9.$$

Finally, we recover the original binary message polynomial by mapping coefficients close to 9 back to 1, and coefficients close to 0 back to 0:

$$m_b = x^3 + x + 1.$$

Kyber - public key size optimization

- We'll consider the Kyber768 parameters $(q = 3329, n = 256, k = 3, \eta_1 = 2, \eta_2 = 2)$.
- The bitlength of an integer in $\mathbb{Z}_q$ is $\lceil \log_2 3329 \rceil = 12$ bits.
- The size of an encryption key (A, t) is

$$(9 \times 256 \times 12) + (3 \times 256 \times 12) \text{ bits, or } 4608 \text{ bytes.}$$

- **Idea:** Generate A from a random (and public) 256-bit seed ρ.
- The polynomials in A can be generated by first selecting $\rho \in_R \{0,1\}^{256}$, and then generating the coefficients of the polynomials by hashing ρ with a counter.
- The encryption key is (ρ, t) instead of (A, t).
- Anyone who knows ρ can generate A.

Kyber - ciphertext size optimization

- The size of a ciphertext $c = (u, v)$ is

$$(3 \times 256 \times 12) + (256 \times 12) \text{ bits, or } 1536 \text{ bytes.}$$

- **Idea:** Discard the "low order" bits of the coefficients of all polynomials in the ciphertext $c = (u, v)$.
- Let $1 \leq d \leq \lfloor \log_2 q \rfloor$, and define:
  - For $x \in [0, q-1]$, $\text{Compress}_q(x, d) = \lfloor (2^d/q) \cdot x \rfloor \mod 2^d$.
  - For $y \in [0, 2^d - 1]$, $\text{Decompress}_q(y, d) = \lfloor (q/2^d) \cdot y \rfloor \mod q$.
- **Fact:** Let $x \in [0, q-1]$ and $x' = \text{Decompress}_q(\text{Compress}_q(x, d), d)$. Then $\|x' - x\|_\infty \leq \lceil q/2^{d+1} \rceil$.
- The ciphertext components u and v are replaced by $c_1 = \text{Compress}_q(u, d_u)$ and $c_2 = \text{Compress}_q(v, d_v)$.
- Kyber768 has $d_u = 10$ and $d_v = 4$.
- So, the size of the compressed ciphertext is

$$3 \times 256 \times 10 + 256 \times 4 \text{ bits, or } 1088 \text{ bytes.}$$

Example

- Let $q = 3329$, $d = 10$.
  - $\text{Compress}_q(223 + 1438x + 3280x^2 + 798x^3, 10) = 69 + 442x + 1009x^2 + 245x^3$.
  - $\text{Decompress}_q(69 + 442x + 1009x^2 + 245x^3) = 224 + 1437x + 3280x^2 + 796x^3$.
  - The error polynomial is $-1 + x + 2x^3$.