PQ Tutorial

Resources:

- https://blog.trailofbits.com/2018/10/22/a-guide-to-post-quantum-cryptography/
- https://cims.nyu.edu/~regev/papers/qcrypto.pdf

Basis for PQ

Systems of linear equations, solvable using linear algebra, like Gauss Elimination.

$$a_{0,0}x_0 + a_{0,1}x_1 + a_{0,n}x_n = y_0 = f^0_{\vec{x_0}}(\vec{a_0})$$

$$a_{1,0}x_0 + a_{1,1}x_1 + a_{1,n}x_n = y_1 = f^1_{\vec{x_1}}(\vec{a_1})$$

$$\dots$$

$$a_{n,0}x_0 + a_{n,1}x_1 + a_{n,n}x_n = y_n = f^n_{\vec{x_n}}(\vec{a_n})$$

Reformulated with Vectors:

$$\vec{a} = a_0, a_1, \dots, a_n$$

$$\vec{x} = x_0, x_1, \dots, x_n$$

$$f_{\vec{x}}(\vec{a}) = a_0x_0 + \dots + a_nx_n$$

When given an $\vec{a}$, we see the result of ax without knowing $\vec{x}$.

Now we introduce and **error** and a prime modulus **q** to generate a noisy function:

$$f_{\vec{x}}(\vec{a}) = a_0x_0 + \dots + a_nx_n + \epsilon \mod q$$

Because of the error term ϵ learning the mystery function is extremely difficult - this is called the **Ring Learning With Errors Problem (LWE)**

Taken from resource:

The reason cryptography based on LWE gets called lattice-based cryptography is because the proof that LWE is hard relies on the fact that finding the shortest vector in something called a lattice is known to be NP-Hard. We won't go into the mathematics of lattices in much depth here, but one can think of lattices as a tiling of n-dimensional space

Cryptosystem from LWE

Public Key is a matrix of vectors $\vec{a}$ from the linear system above:

$$pk = \begin{bmatrix} a_{00} & a_{01} & \dots & a_{0n} & y_0 \\ a_{10} & a_{11} & \dots & a_{1n} & y_1 \\ & & \dots & & \\ a_{n0} & a_{n1} & \dots & a_{nn} & y_n \end{bmatrix}$$

The secret key is a n-dimensional vector

$$sk = (s_0, s_1, \dots, s_n)$$

When ok is multiplied by $(-\vec{sk}, 1)$ the result is the error term, which is about 0.

Encryption and Decryption

Encrypting a bit of information m goes as follows:

1. take the sum of random columns of pk (Matrix A)
2. encode m in the last coordinate of the result by:
    - adding 0 of m is 0
    - q/2 if m is 1

This is the same as picking a random vector x of 0s or 1s and compute:

$$A\vec{x} + (\vec{0}, \mu) = \vec{c}$$

where

$$\mu = m \left\lfloor \frac{q}{2} \right\rceil$$

NOTE: A has $n + 1$ dimensions, as the y-values are contained in it. That means the zero vector $\vec{0}$ must be of dimension n, including μ the dimension becomes $n + 1$.

Example Calculation

$$A = \begin{bmatrix} 1 & 0 & 1 & y_0 \\ 0 & 1 & 1 & y_1 \\ 0 & 0 & 1 & y_2 \\ 1 & 1 & 1 & y_3 \end{bmatrix}$$

$$\vec{x} = (1, 0, 1, 0)$$

In [7]:
```
import numpy as np

q = 11

y0 = 0
y1 = 0
y2 = 0
y3 = 0

pk = 0
pk = np.array([
    [1, 0, 1, 0],
    [0, 1, 1, 0],
    [0, 0, 1, 0],
    [1, 1, 1, 0]
])
x = np.array([1, 0, 1, 0])

r0 = pk.dot(x)
```

In [8]:
```
m = 1
mu = m * q//2
print(mu)
```

```
5
```

In [9]:
```
c = r0 + np.array([0,0,0, mu])
print(c)
```

```
[2 1 1 7]
```

To get the original message m back, one computes:

$$\vec{c} \cdot (-\vec{sk}, 1) \approx A \cdot x \cdot (-\vec{sk}, 1) + m \left\lfloor \frac{q}{2} \right\rceil \approx m \left\lfloor \frac{q}{2} \right\rceil$$

where:

$$(-\vec{sk}, 1) \approx \epsilon \approx 0$$

The recipient can now test if the output is closer to zero or the q/2 mod q and decode the bit.

In [20]:
```
sk = np.array([1,0,1])

sk = -sk
sk = np.append(sk, 1)

r1 = c.dot(sk)
print(r1)
```

```
4
```

In [21]:
```
out = q // 2
if abs(out - r1) < m:
    print("m = 1: {}".format(m))
else:
    print("m = 0: {}".format(m))
```

```
m = 0: 1
```

The result is 1, as expected.

Kyber - rC3 2021

Source: https://www.youtube.com/watch?v=FUb75AUXMvw

Below are my notes / transcript, all credit to above.

Polynomials

Polynomials can be multiplied and added. They can be reduced by modulo operations. The polynomials itself is reduced by another polynomial, the coefficients by modulo with a scalar.

Multiplication / Addition

$$(x^2 + x - 7)(x - 1) = x^3 - 8x + 7$$

Modulo

$$x^17 + 3x^6 + 14x \mod (x^4 + 1)$$

We can use polynomials in matrices and vectors.

$$\begin{bmatrix} x+1 & 3x^2-4 \\ x^2-2 & 2x^2-2x \end{bmatrix} \cdot \begin{bmatrix} 3x^2+2x \\ -x-4 \end{bmatrix} + (-6x - 10, 12x^2 + 3) = (-7x^5 + 6, 4x)$$

Kyber

$$As + e = \begin{bmatrix} x+1 & 3x^2-4 \\ x^2-2 & 2x^2-2x \end{bmatrix} \cdot \begin{bmatrix} 3x^2+2x \\ -x-4 \end{bmatrix} + (-6x - 10, 12x^2 + 3) = (-7x^5 +$$

`e` makes the problem hard, `A` is the public key `s` is the secret, `e` is the error, the result is `t`.

$$As + e = t$$

The public keypair is (A, t) and the private key is s.

Kyber - Encryption

1. transform the letter we want to send to binary

m = 'c' (the letter 'c') = 1000011_2

1. turn this into a polynomial

m = $1 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0 = x^6 + x + 1$ (plaintext)

1. plaintext polynomial is scaled by a factor -> same polynomial, with larger coefficients

$m_s = 1337 \cdot m = 1337 \cdot x^6 + 1337 \cdot x + 1337$

1. add error terms e_0, e_1, e_2 (error terms are more complicated than constants)

$$v = t \cdot e_0 + e_1 + m_s$$

$$u = A \cdot e_0 + e_2$$

v is a polynomial and u is a vector of polynomials.

$$c = (v, u)$$

is the ciphertext.

Kyber - Decryption

1. remove the public key

$$d = v - su = (t \cdot e_0 + e_1 + m_s) - s \cdot (A \cdot e_0 + e_2)$$

$$d = te_0 + e_1 + m_s - Ase_0 + se_2$$

with:

$$As + e = t$$

this becomes:

$$d = (A \cdot s + e) \cdot e_0 + e_1 + m_s - Ase_0 - se_2$$

$$d = A \cdot s \cdot e_0 + e \cdot e_0 + e_1 + m_s - Ase_0 - se_2$$

(the term $A \cdot s \cdot e_0$ gets cancelled out)

$$d = e \cdot e_0 + e_1 + m_s - se_2$$

The error terms are all very small (polynomials with small coefficients) this applies to the secret key as well.

The plaintext on the other hand is large, as we scaled it.

$$d = m_s + (e \cdot e_0 + e_1 - se_2)$$

Noise / Errors are removed by rounding, so we get our scaled plaintext.

$$m_s = 1337 \cdot m = 1337 \cdot x^6 + 1337 \cdot x + 1337$$

Now, we scale down, so all coefficients are divided by the scale factor 1337

$$m = \frac{1337 \cdot m = 1337 \cdot x^6 + 1337 \cdot x + 1337}{1337}$$

We bring back the polynomial by bringing back the zeros in the result polynomial.

$m = \$1 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0 = x^6 + x + 1\$(plainte$

So we have our plaintext back.

$$1000011_2$$

There are three Kyber variants. Kyber512, Kyber768, Kyber1024 - corresponding to a security level of ≈ AES128, AES192, AES256.

In Kyber n is the degree of the polynomials (256 for all variants), $k = 2, 3, 4$ is the size of the vectors of polynomials, that is, how many polynomials are in the vectors.
$q = 3329$ is the modulus for the number used in Kyber.

Compared to Pre-Quantum Crypto:

- Kyber is very fast, faster than Curve25519
- Kyber has larger key sizes than pre-quantum algorithms (RSA3072, Curve25519)