CKKS

Naive Implementation

Given matrix $A \in \mathbb{Z}_q^{n \times n}$ $s, e \in \mathbb{Z}_q^n$, we publish n public keys as a tuple

$$p = (-A \cdot s + e, A)$$

s is hidden because of e.

Naive Implementation-Keys

Given matrix $A \in \mathbb{Z}_q^{n \times n}$ $s, e \in \mathbb{Z}_q^n$, we publish n public keys as a tuple

$$p = (-A \cdot s + e, A)$$

s and e are private.

Encryption

Given message $\mu \in \mathbb{Z}_q^n$, we can encrypt by adding the public key as

$$(\mu, 0) + p = (\mu - A \cdot s + e, A) = (c_0, c_1)$$

We assume that e is negligible compared to μ

Decyption

$$c_0 + c_1 \cdot s = \mu - A \cdot s + e + A \cdot s = \mu + e$$

Problem

$c_1 \cdot S$ is $O(n^2)$ which is too inefficient.

Solution: Use polynomial rings

We get $a, s, e, \mu \in \frac{\mathbb{Z}_q[X]}{X^N - 1}$ and e a small random polynomial.
Then,
1. We publish n public keys as a tuple

$$p = (-a \cdot s + e, a)$$

Since a is size n and not n^2, the complexity is way lower.
2.

$$(\mu, 0) + p = (\mu - a \cdot s + e, a) = (c_0, c_1)$$

3.

$$c_0 + c_1 \cdot s = \mu - a \cdot s + e + a \cdot s = \mu + e$$

Here, $a \cdot s$ can be done with discrete fourier transform in $O(n \log n)$ time.

Properties of CKKS/Homomorphic Encryption

We can do addition and multiplication on encrypted data then recover that

Addition

Let us add the cypher texts

$$c_{add} = c + c' = (c_0 + c'_0, c_1 + c'_1)$$

Let's try decryption

$$\mu + \mu' = c_0 + c'_0 + (c_1 + c'_1)s = \mu + \mu' + 2e \approx \mu + \mu'$$

So just adding normally does work assuming that 2e is negligible.

Multiplication

Now this is a bit more complicated. We want to do some operation on c and c' so that when we decrypt, we get

$$\mu \cdot \mu' = (c_0 + c_1 \cdot s)(c_0' + c_1' \cdot s) = c_0 c_0' + (c_0 c_1' + c_0' c_1)s + c_1 c_1' s^2$$

So if we define the multiplication operation to produce

$$(c_0 c_0', c_0 c_1' + c_0' c_1, c_1 c_1')$$

where decryption multiplies this by $(1, s, s^2)$, we have our solution. To stop this polynomial from keep growing forever, we introduce Relinearization.

Relinearization

We introduce a new polynomial $P \in \frac{\mathbb{Z}_q[X]}{X^N - 1}$ such that when P is decrypted, we get $c_1 c_1' s^2$.
To be more clear, we want to get a pair (p_1, p_2) that if decrypted makes $c_1 c_1' s^2$.
Here, we can see, given a_0 a random polynomial, we can get an evaluation key pair below

$$evaulation := (-a_0 \cdot s + e + s^2, a_0)$$

as

$$(-a_0 \cdot s + e + s^2) + a_0 \cdot s = e + s^2 \approx s^2$$

Then for P, we can just do

$$P := c_1 c_1' evaluation$$

Relinearization

Now that we have P, which when decrypted becomes $c_1 c_1' s^2$, we can decrypt to obtain

$$\mu \cdot \mu' = (c_0 + c_1 \cdot s)(c_0' + c_1' \cdot s) = c_0 c_0' + (c_0 c_1' + c_0' c_1)s + c_1 c_1' s^2$$

To do this, since we are doing relinearization on $(c_0 c_0', c_0 c_1' + c_0' c_1, c_1 c_1')$, we can simply get the pair

$$(c_0 c_0', c_0 c_1' + c_0' c_1) + P$$

as the first pair handles $c_0 c_0' + (c_0 c_1' + c_0' c_1)s$ part and the P handles $c_1 c_1' s^2$

Problem with P

The problem is with our assumption. Even if for the evaluation key,

$$(-a_0 \cdot s + e + s^2) + a_0 \cdot s = e + s^2 \approx s^2$$

is true, when we multiply the above by $c_1 c_1'$,

$$c_1 c_1' e + c_1 c_1' s^2$$

is not necessarily the same as $c_1 c_1' s^2$ because the error can become too large to ignore. Especially hard since for c_1, we are sampling from $\dfrac{\mathbb{Z}_q[X]}{X^N - 1}$.

Trick solution

The main solution to this problem is to change the valuation key such that, given $a_0 \in \frac{\mathbb{Z}_{pq}[X]}{X^N - 1}$

$$evaulation := (-a_0 \cdot s + e + p \cdot s^2, a_0) \mod pq$$

If we decrypt this, we obtain $p \cdot s^2$. Now, for getting P, we do

$$P := p^{-1} c_1 c_1' evaluation$$

So we thus make the error negligible when decoding by inverting by p. Now we are done with multiplication!

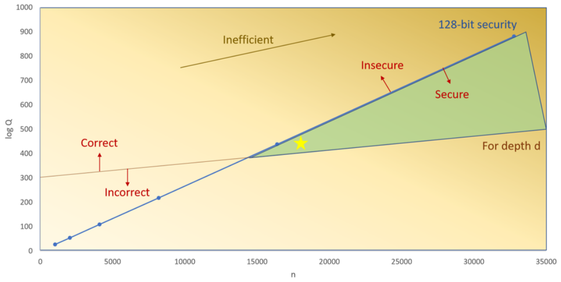
Problems

When we do keep multiplying after a while, the error gets too big to ignore again.
To solve this we need to do a technique called rescaling.

Rescaling

To do this technique we need to

1. Know the amount of multiplications until the error becomes too much. If we make the amount of multiplications allowed too much, the encryption becomes less secure. The hardness of the encryption is based on $\frac{N}{q}$. More variables+higher coefficients=less secure. q will give us the amount of multiplications we can do as $c_1 c_1'$ are both sampled from $a \in \frac{\mathbb{Z}_q[X]}{X^N - 1}$



Rescaling

For our $a, \mu \in \dfrac{\mathbb{Z}_q[X]}{X^N - 1}$, but not the error, we first multiply by $\triangle$ before encryption. So given $c = \triangle z$, when we multiply two cyphers Why do we have $\triangle$?

$$cc' = \triangle^2 zz'$$

We want to

1. Keep this scale constant
2. Reduce the noise.

Rescaling

After L multiplications,

$$q_L = \triangle^L q_0$$

Now in q_0, the $\triangle$ will dictate the amount of precision we want in the decimal part of the gas tank(why?). So given we want 10 bits for integers and 30 bit decimal, $\triangle = 2^{30}$ and $q_0 = 2^{10+30} = 2^{40}$. Note q can be not prime.
As we want to keep the scale cosntant, to rescale from level q_l to level q_{l-1} we can simply divide the cypher text as follows.

$$\frac{q_{l-1}}{q_l} c \mod q_l = \triangle^{-1} c \mod q_l$$

Rescaling

The previous operation accomplishes 2 things1 1. Since we are dividing by $\triangle$, the result of the multiplication of cyphers cc' will just be $\triangle zz'$. 2. Noise is reduced as we are dividing by $\triangle$ which it wasn't scaled by on each level.

Chinese Remainder Theorem

The problem is that these qs become too large very fast as if $\triangle$ is 2^{30} one operation is enough to make it infeasable to fit in a 64 bit register.
To solve this we use the property of the chinese remainder theorem where given $p = \prod_{i=1}^{L} p_i$

$$\mathbb{Z}/p\mathbb{Z} \to \mathbb{Z}/p_1\mathbb{Z} X \cdots \mathbb{Z}/p_L\mathbb{Z}$$

So the operation in the left space is the same as the right space.
Now, we also choose $p_i \approx \triangle$

Chinese Remainder Theorem

So thus instead of $\triangle^L$ we do

$$q_L = \prod_{i=1}^{L} p_i q_0$$

and do the computation with the Chinese Remainder Theorem. To rescale we can just do

$$\frac{q_{l-1}}{q_l} c \mod q_l = p_l^{-1} c \mod q_l$$