Thanks to https://blog.openmined.org/ckks-explained-part-3-encryption-and-decryption/

Basic encryption scheme:

Apparently lattice algorithms are robust to quantum attacks for some reason. I should investigate this later.

1. 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

$$p = (-A \cdot s + e, A) \tag{1}$$

It'll be easy to recover s if we don't have e there by just multiplying both sides by $-A^{-1}$.

We'll assume here that $\mu$ is negligible compared to $\mu$.

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

$$(\mu, 0) + p = (\mu - A \cdot s + e, A) = (c_0, c_1) \tag{2}$$

3. Now to decrypt, we do

$$c_0 + c_1 \cdot s = \mu - A \cdot s + e + A \cdot s = \mu + e \tag{3}$$

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

Solution: Using polynomial rings:

We get $a, s, e, \mu \in \dfrac{\mathbb{Z}_q(X)}{X^N - 1}$. Then,

1. We publish n public keys as a tuple

$$p = (-a \cdot s + e, a) \tag{4}$$

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

2.
$$(\mu, 0) + p = (\mu - a \cdot s + e, a) = (c_0, c_1) \tag{5}$$

3.
$$c_0 + c_1 \cdot s = \mu - a \cdot s + e + a \cdot s = \mu + e \tag{6}$$

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

Thus the size is $O(n)$ and speed is $O(n \log n)$now.

We are doing homomorphic encryption so that we can still do operations on the encrypted data.

Addition:

Let us add the cypher texts

$$c_a dd = c + c' = (c_0 + c_0', c_1 + c_1') \tag{7}$$

Let's try decryption

$$c_0 + c_0' + (c_1 + c_1')s = \mu + \mu' + 2e \approx \mu + \mu' \tag{8}$$

So just adding normally does work.

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

$$(c_0 + c_1 \cdot s)(c_0' + c_1' \cdot s) = c_0c_0' + (c_0c_1' + c_0'c_1)s + c_1c_1's^2 \tag{9}$$

So if we define the multiplication operation to produce

$$(c_0c_0', c_0c_1' + c_0'c_1, c_1c_1') \tag{10}$$

where decryption multiplies this by $(1, s, s^2)$, we have our solution.

Now, to stop this polynomial vector from growing forever, we introduce relinearization.

For relinearlization, we introduce a new polynomial $P \in \dfrac{\mathbb{Z}_q(X)}{X^N - 1}$ such that when P is decrypted, we get $c_1c_1's^2$.

Once we have this key we can do

$$c_{mult} = (c_0c_0', c_0c_1' + c_0'c_1) + P \tag{11}$$

now we can do relinearlization by first getting an evaluation key

$$(-a_0 \cdot s + e + s^2, a_0) \tag{12}$$

If this is decrypted, we get $e + s^2 \approx s^2$.

Now, we can't just multiply $c_1c_1'$ to this because then $ec_1c_1'$ won't be negligible as $c_1c_1'$ is large. So we can make a new key as

$$(-a_0 \cdot s + e + ps^2, a_0) \mod p \cdot q \tag{13}$$

where p is a big integer. Then, we find P we do

$$p^{-1}c_1c_1'(-a_0 \cdot s + e + ps^2, a_0) \mod p \cdot q \tag{14}$$

where we divide $c_1c_1'$ and round to the nearest integer $\mod q$

Now we can do multiplication!

Problem: Each time we do a computation, noise is present. So after a while, it'll become too large to ignore. To work with this, we can scale the noise to the right amount by knowing the amount of computation we need to do beforehand. Although no noise is insecure.

Now let's say we have a scale $\triangle$. The higher the scale the higher the precision. The general idea is during encryption, we multiply by this scale, and during decryption we divide by this scale. This helps in reducing noise at high scales.

The main bottleneck/cause of errors exploding is multiplication as we multiply 2 integers to $\triangle$ like

$$z\triangle^2 \tag{15}$$

while with addition we just keep adding more es like

$$z\triangle \tag{16}$$

for some integer z.

Now, to have a good q to accomodate for L multiplications, we do

$$q = q_0 \triangle^L \tag{17}$$

Now once we do one multiplication, we can decrease the level by doing

$$\frac{q_{l-1}}{q_l} c = \triangle^{-1} c \tag{18}$$

which would keep the scale S $\triangle$ instead of $\triangle^2$. This also reduces the noise. And now we are done with multiplication!

Now one final point. As $q_0 \triangle^L$ can become uncomputably large, we use chinese remainder theorem on L prime numbers where $p_l \approx \triangle$ and we can do

$$q_l = q_0 \prod_{i=0}^{l} p_i \tag{19}$$

Now given $p = \prod_{i=0}^{l} p_i$, we can do

$$x \mod p = (x \mod p_0) \times (x \mod p_1).... \tag{20}$$

So rescaling is now just

$$\frac{q_{l-1}}{q_l} c = p_l^{-1} c \tag{21}$$