CKKS Homomorphic Encryption Part 1 - The Original Scheme

Notes by Linus Tang.

These notes have not been thoroughly reviewed. Any errors below are my own responsibility.

Thanks to Brian Gu for helping me fix a typo!

Sources:
- [CKKS'16] The original paper by Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song
  - https://eprint.iacr.org/2016/421.pdf

Assumed background knowledge:
- Know what (fully) homomorphic encryption is
  - https://en.wikipedia.org/wiki/Homomorphic_encryption
  - Familiarity with at least one FHE scheme may help. I strongly recommend reading about BFV first.
    - https://docs.google.com/viewerng/viewer?url=https://www.mit.edu/~linust/files/BFV_Homomorphic_Encryption_Part_1.pdf
- Some familiarity with polynomial rings, complex numbers, and roots of unity

What these notes cover:

Part 1: Covers the original scheme which achieves leveled homomorphic encryption. There are a lot of overlapping ideas with BFV.

Part 2: Covers the bootstrapping procedure of to achieve fully homomorphic encryption. There are many really nice ideas in the bootstrapping!

Part 3: Covers some optimizations to the bootstrapping procedure.

Details about choosing parameters and analyzing security are outside the scope of these notes. We will go into some detail about the noise analysis but not prove everything rigorously.

Contents

Part 1: Introduction and Leveled Homomorphic Encryption

1. Approximate Arithmetic

Don't worry too much if this section doesn't make a lot of sense now; it will hopefully make more sense after you read further.

CKKS is a fully homomorphic scheme which performs *approximate* arithmeic. In particular, plaintexts are effectively rounded to some precision before they are encrypted. Each operation also incurs some noise, and decryption does not remove noise (unlike schemes for exact arithmetic such as BFV, in which decryption removes errors less than some tolerance $\frac{\Delta}{2}$.)

In homomorphic encryption schemes for exact arithmetic, noise is allowed to grow within some threshold, and the bootstrapping operation decreases the noise of a ciphertext that gets too close to the threshold. In contrast, in CKKS, noise is not removed by decryption, so parameters are set in a way that keeps the overall error of the entire computation within a desired level of precision.

In BFV, each ciphertext multiplication amplifies the noise of a ciphertext by a large factor, so the noise of the output of a circuit is exponential in its depth. Bootstrapping reduces the noise of a ciphertext from just below the tolerance $\frac{\Delta}{2}$ to a constant level.

In CKKS, there is no mechanism for noise reduction, so it is unacceptable for noise to grow exponentially with circuit depth. Each ciphertext multiplication is done in a way that adds a constant term to the error rather than multiplying it by a constant factor, so that the noise is linear instead of exponential in circuit depth. This "noise-efficient" multiplication requires decreasing the ciphertext modulus, so bootstrapping in CKKS involves increasing the modulus back to a larger value.

2. Notation

- We will use sans letters to denote keys, ciphertexts, and their components (which are polynomials). We use bold letters to denote vectors of numbers. Unfortunately, default typesetting will be used for both numbers and polynomials.
- For an integer a, we will use $\mathbb{Z}_a$ to denote the set of a integers lying in $\left(-\frac{a}{2}, \frac{a}{2}\right]$, not the ring $\mathbb{Z}/a\mathbb{Z}$.
  - For any integer r, we let $[r]_a$ denote the residue of r belonging to $\mathbb{Z}_a$. For a polynomial r, we let $[r]_a$ denote the same operation applied coefficient-wise. For a vector $\mathsf{R} = (r_1, r_2)$ of polynomials, we let $[\mathsf{R}]_a$ denote the same operation applied component-wise, i.e. $[\mathsf{R}]_a = \left([r_1]_a, [r_2]_a\right)$.
  - For real r we will let $\lfloor r \rceil$ denote r rounded to the nearest integer (this operation will typically be applied to a quotient in order to denote more general rounding). Similarly, this operation also applies coefficient-wise to polynomials and component-wise to vectors of polynomials.
- We will use $\langle \cdot, \cdot \rangle$ to denote the dot product of two vectors.

3. Encoding and Decoding

The overall goal of this section is to encode a vector $\boldsymbol{z}$ of complex numbers as a polynomial $m_{\Delta \boldsymbol{z}}$ in $\mathbb{Z}[X]/(X^N + 1)$ such that evaluating $m_{\Delta \boldsymbol{z}}$ on the roots of $X^N + 1$ approximately recovers $\boldsymbol{z}$.

Let N be a power of 2 and R_N be the set of roots of $X^N + 1$. We can write $R_N = \{\zeta, \zeta^3, \zeta^5, \dots, \zeta^{2N-1}\}$, where $\zeta = e^{i\pi/N}$.

The message space is $\mathbb{C}^{N/2}$. In particular, a message will be a vector $\boldsymbol{z} = \left(z_1, \dots, z_{N/2}\right)$ of complex numbers, and addition and multiplication of messages will be component-wise.

For the purposes of encoding, a message $\boldsymbol{z}$ is associated with a function $f_{\boldsymbol{z}} : R_N \to \mathbb{C}$ such that $f_{\boldsymbol{z}}(w) = \overline{f_{\boldsymbol{z}}(\overline{w})}$ for all $w \in R_N$. In particular, we define $f_{\boldsymbol{z}}$ so that its N evaluations are given by

$$\begin{aligned} z &= (f_z(\zeta^1), f_z(\zeta^3), f_z(\zeta^5), ..., f_z(\zeta^{N-1})) \\ &= \left(\overline{f_z\left(\overline{\zeta^1}\right)}, \overline{f_z\left(\overline{\zeta^3}\right)}, \overline{f_z\left(\overline{\zeta^5}\right)}, ..., \overline{f_z\left(\overline{\zeta^{N-1}}\right)}\right). \end{aligned}$$

Lagrange interpolation can be used to find the unique polynomial $p_z(X) \in \mathbb{C}[X]/(X^N + 1)$ that restricts to f_z, i.e.

$$p_z(w) = f_z(w), \qquad w \in R_N.$$

Note that the above evaluation of a polynomial p_z in a quotient ring $\mathbb{C}[X]/(X^N + 1)$ is a well-defined operation precisely because we are only evaluating it on roots $x \in R_N$ of the polynomial modulus $X^N + 1$.

The property that f_z commutes with conjugation implies that the coefficients of p_z are real (mathematical exercise!), so we have $p_z(X) \in \mathbb{R}[X]/(X^N + 1)$. Since we will eventually want to be working with integer polynomials, we can round each coefficient of p_z to the nearest integer to get

$$m_z(X) = \lfloor p_z(X) \rceil \in \mathbb{Z}[X]/(X^N + 1).$$

Actually, rounding to the nearest integer is a rather large precision loss if z is small, so we multiply messages by a large scaling factor Δ before interpolating and rounding. So, the encoding of z is $m_{\Delta z}$.

To decode a message from its encoding $m_{\Delta z}$, simply evaluate it on the roots of $X^N + 1$, and divide by Δ. In particular,

$$\Delta z \approx (m_{\Delta z}(\zeta), m_{\Delta z}(\zeta^3), m_{\Delta z}(\zeta^5), ..., m_{\Delta z}(\zeta^{N-1})),$$

so

$$z \approx \frac{1}{\Delta}(m_{\Delta z}(\zeta), m_{\Delta z}(\zeta^3), m_{\Delta z}(\zeta^5), ..., m_{\Delta z}(\zeta^{N-1})).$$

The error in the first approximation can be upper-bounded independently of Δ and z', so choosing a large Δ makes the encoding error in the second approximation small.

Example

Suppose that $N = 4$ and $\Delta = 64$ and we want to encrypt the message $z = (3 + 4i, 2 - i)$. So, we want to find the polynomial $p_z(X) \in \mathbb{Z}[X]/(X^8 + 1)$ such that

$$(p_z(\zeta), p_z(\zeta^3), p_z(\zeta^5), p_z(\zeta^7)) = (3 + 4i, 2 - i, 2 + i, 3 - 4i),$$

where $\zeta = 2^{i\pi/4}$. This polynomial turns out to be

$$p_z(X) = \frac{5}{2} + \sqrt{2}X + \frac{5}{2}X^2 + \frac{\sqrt{2}}{2}X^3$$

So we have

$$p_{\Delta z}(X) = \Delta p_z(X) = 160 + 64\sqrt{2}X + 160X^2 + 32\sqrt{2}X^3.$$

Rounding each coefficient to the nearest integer gives us

$$m_{\Delta z}(X) = \lfloor p_{\Delta z}(X) \rceil = 160 + 91X + 160X^2 + 45X^3.$$

This decodes to $\frac{1}{\Delta}(m_{\Delta z}(\zeta), m_{\Delta z}(\zeta^3)) \approx (3.0082 + 4.0026i, 1.9918 - 0.9974i)$, which is indeed close to $z = (3 + 4i, 2 - i)$.

Summary

- A message is a vector of $N/2$ complex numbers.
- To encode a message $\boldsymbol{z} \in \mathbb{C}^{N/2}$, find the unique polynomial $p_{\Delta \boldsymbol{z}} \in \mathbb{C}[X]/(X^N + 1)$ such that

$$\begin{aligned} \boldsymbol{z} &= (p_{\Delta \boldsymbol{z}}(\zeta^1), p_{\Delta \boldsymbol{z}}(\zeta^3), p_{\Delta \boldsymbol{z}}(\zeta^5), \dots, p_{\Delta \boldsymbol{z}}(\zeta^{N-1})) \\ &= \left(\overline{p_{\Delta \boldsymbol{z}}\left(\overline{\zeta^1}\right)}, \overline{p_{\Delta \boldsymbol{z}}\left(\overline{\zeta^3}\right)}, \overline{p_{\Delta \boldsymbol{z}}\left(\overline{\zeta^5}\right)}, \dots, \overline{p_{\Delta \boldsymbol{z}}\left(\overline{\zeta^{N-1}}\right)}\right). \end{aligned}$$

- It is guaranteed that $p_{\Delta \boldsymbol{z}}$ has real coefficients, so each coefficient can be rounded to the nearest integer to get the encoding $m_{\Delta \boldsymbol{z}} \in \mathbb{Z}[x]/(x^N + 1)$.
- In order to approximately recover $\boldsymbol{z}$ from its encoding $m_{\Delta \boldsymbol{z}} \in \mathbb{Z}[x]/(x^N + 1)$, use

$$\boldsymbol{z} \approx \frac{1}{\Delta}(m_{\Delta \boldsymbol{z}}(\zeta), m_{\Delta \boldsymbol{z}}(\zeta^3), m_{\Delta \boldsymbol{z}}(\zeta^5), \dots, m_{\Delta \boldsymbol{z}}(\zeta^{N-1})).$$

- We call $\boldsymbol{z}$ the message and $m_{\Delta \boldsymbol{z}}$ the plaintext.

Since components of message vectors are encoded as evaluations of plaintext polynomials, working with the plaintexts results in component-wise addition and multiplication of messages. In particular,

$$m_{\Delta \boldsymbol{z}^{(1)}} + m_{\Delta \boldsymbol{z}^{(2)}} = m_{\Delta (\boldsymbol{z}^{(1)} + \boldsymbol{z}^{(2)})}$$

and

$$\frac{1}{\Delta} \cdot m_{\Delta \boldsymbol{z}^{(1)}} m_{\Delta \boldsymbol{z}^{(2)}} = m_{\Delta (\boldsymbol{z}^{(1)} \odot \boldsymbol{z}^{(2)})},$$

where $\odot$ represents componentwise multiplication.

In sections 4 and 5 on key generation, encryption, decryption, and homomorphic operations, we can, for the most part, forget about messages $\boldsymbol{z}$ and instead think directly in terms of the polynomial encodings m.

4. Setup

Since we will use the RLWE assumption to securely encrypt polynomials, we will need to work in a large modulus q. Let $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_q$ be the set of polynomials in $\mathcal{R}$ with all coefficients in $\mathbb{Z}_q$. All additions and multiplications on keys and ciphertext polynomials will be done in $\mathcal{R}$, but we often end computations by taking $[\cdot]_q$.

Key generation

The secret key $\mathsf{SK} = (1, s) \in \mathcal{R}^2$ where s is a random small polynomial in $\mathcal{R}$. (We will see later that adding the dummy first component of 1 leads to notational convenience, because decryption becomes taking a dot product with SK.)

- At risk of overcomplicating this exposition:
  - We use the *canonical embedding norm* $\|v\|_\infty^{\text{can}} = \max_{x \in R_N} |v(x)|$ as our main measure of the size of a polynomial. This choice of norm can be motivated by the fact that polynomials decode by evaluating them on elements of R_N, but it is mainly useful for analyzing the noise growth of homomorphic operations due to the three following useful properties:
  - The norm $\| \cdot \|_\infty^{\text{can}}$ is subadditive, i.e. $\|u\|_\infty^{\text{can}} + \|v\|_\infty^{\text{can}} \leq \|u + v\|_\infty^{\text{can}}$ for all $u, v \in \mathcal{R}$.
  - The norm $\| \cdot \|_\infty^{\text{can}}$ is submultiplicative, i.e. $\|u\|_\infty^{\text{can}} \|v\|_\infty^{\text{can}} \leq \|uv\|_\infty^{\text{can}}$ for all $u, v \in \mathcal{R}$.
  - If a polynomial has a small canonical embedding norm, then each of its coefficients is not too large either. (We don't spell out this out formally but refer the dedicated reader to 2.2 of [CKKS'16].)

- When we say "random small polynomial," in practice we mean that we generate a polynomial in $\mathcal{R}$ with random bounded coefficients and keep track of the implied upper bound on its canonical embedding norm.
- The important thing here is that if the result of a computation is off from a target by a small polynomial or product of small polynomials, we know how to bound the discrepancy between the result and target in a useful and relatively tight manner.

The public key is

$$\mathsf{PK} = (\mathsf{PK}_1, \mathsf{PK}_2) = [(-a \cdot s + e, a)]_q \in \mathcal{R}_q^2,$$

where a is a random polynomial in $\mathcal{R}_q$ and e is a random small polynomial in $\mathcal{R}$.

- The Ring Learning With Errors (RLWE) assumption states that efficient adversaries (without knowledge of s) cannot distinguish $(\mathsf{PK}_1, \mathsf{PK}_2)$ as defined above from a two independent uniform random draws from $\mathcal{R}_q$. The security of the CKKS scheme relies on the RLWE assumption.
- Formal security proofs are outside the scope of these notes, but a helpful intuition is that if $(\mathsf{PK}_1, \mathsf{PK}_2)$ looks indistinguishable from uniform randomness to the server, then the public key gives the server no information about the underlying secret key SK.

There is also a public evaluation key EK for ciphertext multiplication and some other public bootstrapping keys used for bootstrapping (part 2), each of which will be discussed later when it can be better motivated.

Encryption
A ciphertext of a polynomial m is

$$\mathsf{C} = (\mathsf{C}_1, \mathsf{C}_2) = [u \cdot \mathsf{PK} + (m + e_1, e_2)]_q \in \mathcal{R}_q^2,$$

where u, e_1, e_2 are random small polynomials.

Decryption
The key property that will be used for decryption is that

$$\begin{aligned}\langle \mathsf{C}, \mathsf{SK} \rangle &\equiv m + (e \cdot u + e_1 + e_2 \cdot s) \pmod{q} \\ &= m + v,\end{aligned}$$

where v is small because e, u, e_1, e_2, s are small. In fact, a more careful analysis shows that

$$\langle \mathsf{C}, \mathsf{SK} \rangle = m + v + qr$$

where r is also small. We call v the noise of the ciphertext C.

A holder of the secret key can approximately decrypt a ciphertext through $m \approx m + v = [\langle \mathsf{C}, \mathsf{SK} \rangle]_q$ but cannot remove the small noise v. Here, we must assume that $|m + v| < \frac{q}{2}$ so that our decryption is not off by a nonzero multiple of q. In practice, this means that the modulus we use must be large enough that the coefficients of our intermediate computations do not overflow it.

Rescaling
Recall from earlier that when we want to multiply messages $z^{(1)}, z^{(2)}$ component-wise, we multiply their plaintexts $m^{(1)}, m^{(2)}$ and divide by Δ because

$$\frac{1}{\Delta} \cdot m_{\Delta z^{(1)}} m_{\Delta z^{(2)}} = m_{\Delta \left(z^{(1)} \odot z^{(2)}\right)}.$$

We now discuss some of the implications of having to divide a plaintext by Δ.

To divide a plaintext by Δ, we divide each component of its ciphertext by Δ (and round to the nearest integer). In doing so, we need to also divide the modulus q of the ciphertext by Δ, in order to get a deterministic result. We call this rescaling. The rescaling operation approximately divides the noise and modulus by Δ, but slightly increases the worst-case noise-to-modulus ratio due to the step of rounding to the nearest integer.

In order to support multiplication of ciphertexts, we want our starting modulus to be divisible by a large power of Δ. We will let q_0 be some parameter and let $q_\ell = q_0 \cdot \Delta^\ell$ for all ℓ. Then we will choose some parameter L and let $q = q_L$ be the modulus of each freshly-generated ciphertext. If a ciphertext has modulus q_ℓ, we will say that ℓ is the *level* of the ciphertext. In particular, the level of a ciphertext starts at L and decreases by 1 every time it participates in a homomorphic multilication, due to rescaling.

Summary

- Parameters N, q_0, and L are chosen. Below, $q_\ell = q_0 \cdot \Delta^\ell$ and $q = q_L$.
- We use the canonical embedding norm $\|v\|_\infty^{\text{can}} = \max_{x \in R_N} |v(x)|$ as our main measure of the size of a polynomial, which is convenient for noise analysis because it is submultiplicative.
- The user generates a secret key
$$\mathsf{SK} = (1, s) \in \mathcal{R}^2$$
where $s \in \mathcal{R}$ is a small random polynomial.
- The user generates a public key
$$\mathsf{PK} = (\mathsf{PK}_1, \mathsf{PK}_2) = [(-a \cdot s + e, a)]_q \in \mathcal{R}_q^2,$$
where a is random over $\mathcal{R}_q$ and e is a small random polynomial, and sends it to the server.
- The user generates and sends an evaluation key EK and some other keys, which will be used for multiplication and bootstrapping. These will be discussed later.
- One way to securely encrypt a polynomial m is to commpute
$$\mathsf{C} = (\mathsf{C}_1, \mathsf{C}_2) = [u \cdot \mathsf{PK} + (m + e_1, e_2)]_q,$$
where u, e_1, e_2 are random polynomials in $\mathcal{R}$ with small coefficients. The user encrypts all inputs to the computation in this fashion and sends the ciphertexts to the server.
- A ciphertext C with level ℓ can be decrypted by noting that
$$\langle \mathsf{C}, \mathsf{SK} \rangle \equiv m + v \pmod{q_\ell}$$
with v small. Assuming that $|m + v| < \frac{q_\ell}{2}$ (i.e. our message does not overflow our modulus), we have $m \approx [\langle \mathsf{C}, \mathsf{SK} \rangle]_{q_\ell}$.
  - From now on, we will assume that parameters are chosen such that the message never overflows the modulus in the above sense, even when this assumption is not explicitly stated.

In the next section, we will see how the server can take ciphertexts for two messages and obtain ciphertext for their sum or product, with slightly increased noise, thus achieving leveled homomorphic encryption.

5. Homomorphic Operations

Ciphertext Tags

From now on, for notational clarity, we will tag a ciphertext with information about its level, size, and noise. In particular, a full ciphertext will be of the form $(\mathsf{C}, \ell, \nu, B)$, and the meaning of each tag will become clear below.

For $\mathsf{C} \in \mathcal{R}^2$ with level ℓ, and for m in $\mathcal{R}_{q_\ell}$, we say that C encrypts m with noise at most B if

$$\langle \mathsf{C}, \mathsf{SK} \rangle \equiv m + v \pmod{q_\ell}$$

for some polynomial v with $\|v\|_\infty^{\text{can}} \leq B$. For any real number ν such that $\|m\|_\infty^{\text{can}} \leq \nu$, we say that

$$\mathsf{C} \in \text{Enc}_{(\ell,\nu,B)}(m).$$

If we tag a ciphertext C as $(\mathsf{C}, \ell, \nu, B)$, we are guaranteeing that $\mathsf{C} \in \text{Enc}_{(\ell,\nu,B)}(m)$, where m is the message that C is meant to encrypt.

For example, in this new notation, we have so far developed the following pieces of the scheme:

$$\text{Encrypt} : m \mapsto (\mathsf{C}, L, \nu, B_{\text{clean}}) \text{ for some } \nu \geq \|m\|_\infty^{\text{can}}$$

$$\text{Decrypt} : (\mathsf{C}, \ell, \nu, B) \mapsto [\langle \mathsf{C}, \mathsf{SK} \rangle]_{q_\ell}$$

$$\text{Rescale} : (\mathsf{C}, \ell, \nu, B) \mapsto (\lfloor \mathsf{C}/\Delta \rceil, \ell - 1, \nu/\Delta, B + B_{\text{scale}})$$

for some constants B_{clean} and B_{scale} depending on the parameters of the scheme.

We keep track of the noise bound B so that we can ultimately get guarantees on how much precision our enryption scheme offers on a given computation. We keep track of an upper bound ν on the size of the encrypted polynomial because it should not overflow the modulus and also because it is involved in the expression that updates our noise bound after a multiplication.

In particular, in this section, we will construct the following functions:

$$\text{Add} : \left(\left(\mathsf{C}^{(1)}, \ell, \nu^{(1)}, B^{(1)}\right), \left(\mathsf{C}^{(2)}, \ell, \nu^{(2)}, B^{(2)}\right)\right) \mapsto \left(\mathsf{C}_{\text{add}}, \ell, \nu^{(1)} + \nu^{(2)}, B^{(1)} + B^{(2)}\right)$$

$$\text{Mult} : \left(\left(\mathsf{C}^{(1)}, \ell, \nu^{(1)}, B^{(1)}\right), \left(\mathsf{C}^{(2)}, \ell, \nu^{(2)}, B^{(2)}\right)\right) \mapsto \left(\mathsf{C}_{\text{mult}}, \ell - 1, \nu^{(1)}\nu^{(2)}, \frac{\nu^{(1)}B^{(2)} + \nu^{(2)}B^{(1)} + B^{(1)}B^{(2)} + B_{\text{mult}}}{\Delta}\right)$$

Of course, C_{add} will encrypt $m^{(1)} + m^{(2)}$ and C_{mult} will encrypt $\frac{1}{\Delta} \cdot m^{(1)}m^{(2)}$, where $m^{(1)}$ and $m^{(2)}$ are the plaintexts encrypted by the input ciphertexts $C^{(1)}$ and $C^{(2)}$.

We will also construct a function that will help us add or multiply ciphertexts that were originally on different levels.

$$\text{ModDecrease}_{\ell \to \ell'} : (\mathsf{C}, \ell, \nu, B) \mapsto \left([\mathsf{C}]_{q_{\ell'}}, \ell', \frac{\nu}{\Delta^{\ell-\ell'}}, \frac{B}{\Delta^{\ell-\ell'}} + B_{\text{scale}}\right)$$

where $\ell' < \ell$. Above, B_{mult} and B_{scale} are constants that depend on the parameters of the scheme.

Adding two ciphertexts

Suppose ciphertexts $\left(\mathsf{C}^{(1)}, \ell, \nu^{(1)}, B^{(1)}\right)$ and $\left(\mathsf{C}^{(2)}, \ell, \nu^{(2)}, B^{(2)}\right)$ encrypt $m^{(1)}$ and $m^{(2)}$, respectively. If we want a ciphertext that encodes $m^{(1)} + m^{(2)}$, we add the ciphertexts component-wise:

$$\mathsf{C}_{\text{add}} = \mathsf{C}^{(1)} + \mathsf{C}^{(2)}$$

encrypts $m^{(1)} + m^{(2)}$, with size and noise bounds given by the following proposition:

$$\mathsf{C}_{\text{add}} \in \text{Enc}_{(\ell, \nu^{(1)}+\nu^{(2)}, B^{(1)}+B^{(2)})}\big(m^{(1)} + m^{(2)}\big).$$

The proof is straightforward. The main idea is that the decryption function is linear, so the noise of the sum of two ciphertexts is the sum of the noises.

For $i \in \{1, 2\}$ we have

$$\langle \mathsf{C}^{(i)}, \mathsf{SK} \rangle \equiv m^{(i)} + v^{(i)} \pmod q$$

for some $\|v^{(i)}\|_\infty^{\text{can}} \le B^{(i)}$. Thus,

$$\begin{aligned} \langle \mathsf{C}^{(1)} + \mathsf{C}^{(2)}, \mathsf{SK} \rangle &= \langle \mathsf{C}^{(1)}, \mathsf{SK} \rangle + \langle \mathsf{C}^{(2)}, \mathsf{SK} \rangle \\ &\equiv \big(m^{(1)} + m^{(2)}\big) + \big(v^{(1)} + v^{(2)}\big) \pmod q \end{aligned}$$

and $\|v^{(1)} + v^{(2)}\|_\infty^{\text{can}} \le \|v^{(1)}\|_\infty^{\text{can}} + \|v^{(2)}\|_\infty^{\text{can}} \le B^{(1)} + B^{(2)}$.

So, $\mathsf{C}^{(1)} + \mathsf{C}^{(2)}$ encrypts $m^{(1)} + m^{(2)}$ with noise at most $B^{(1)} + B^{(2)}$.

Also, $\|m^{(1)} + m^{(2)}\|_\infty^{\text{can}} \le \|m^{(1)}\|_\infty^{\text{can}} + \|m^{(2)}\|_\infty^{\text{can}} \le \nu^{(1)} + \nu^{(2)}$, so

$$C^{(1)} + C^{(2)} \in \text{Enc}_{(\ell, \nu^{(1)}+\nu^{(2)}, B^{(1)}+B^{(2)})}\big(m^{(1)} + m^{(2)}\big),$$

as desired.

Multiplying two ciphertexts and rescaling

Suppose ciphertexts $\big(\mathsf{C}^{(1)}, \ell, \nu^{(1)}, B^{(1)}\big)$ and $\big(\mathsf{C}^{(2)}, \ell, \nu^{(2)}, B^{(2)}\big)$ encrypt $m^{(1)}$ and $m^{(2)}$, respectively, and we want a ciphertext that encrypts $\frac{1}{\Delta} \cdot m^{(1)} m^{(2)}$.

Note the expansion

$$\begin{aligned} \big(m^{(1)} + v^{(1)}\big)\big(m^{(2)} + v^{(2)}\big) &= \big[\langle \mathsf{C}^{(1)}, \mathsf{SK} \rangle \langle \mathsf{C}^{(2)}, \mathsf{SK} \rangle\big]_{q_\ell} \\ &= \Big[\Big\langle \Big(\mathsf{C}_1^{(1)}, \mathsf{C}_2^{(1)}\Big), (1, s) \Big\rangle \Big\langle \Big(\mathsf{C}_1^{(2)}, \mathsf{C}_2^{(2)}\Big), (1, s) \Big\rangle\Big]_{q_\ell} \\ &= \Big[\Big\langle \Big(\mathsf{C}_1^{(1)}\mathsf{C}_1^{(2)}, \mathsf{C}_1^{(1)}\mathsf{C}_2^{(2)} + \mathsf{C}_2^{(1)}\mathsf{C}_1^{(2)}, \mathsf{C}_2^{(1)}\mathsf{C}_2^{(2)}\Big), (1, s, s^2) \Big\rangle\Big]_{q_\ell}. \end{aligned}$$

Now let $(\mathsf{C}'_1, \mathsf{C}'_2, \mathsf{C}'_3) = \Big[\Big(\mathsf{C}_1^{(1)}\mathsf{C}_1^{(2)}, \mathsf{C}_1^{(1)}\mathsf{C}_2^{(2)} + \mathsf{C}_2^{(1)}\mathsf{C}_1^{(2)}, \mathsf{C}_2^{(1)}\mathsf{C}_2^{(2)}\Big)\Big]_{q_\ell}$ so that

$$\langle (\mathsf{C}'_1, \mathsf{C}'_2, \mathsf{C}'_3), (1, s, s^2) \rangle = \big(m^{(1)} + v^{(1)}\big)\big(m^{(2)} + v^{(2)}\big).$$

This gives us a three-component "ciphertext" $(\mathsf{C}'_1, \mathsf{C}'_2, \mathsf{C}'_3)$ whose mod q dot product with $\big(1, s, s^2\big)$ is approximately $m^{(1)} m^{(2)}$. However, what we want is a ciphertext of two polynomials whose mod q dot product with $(1, s)$ is $m^{(1)} m^{(2)}$.

To help the server reduce the three components back down to two, the user can provide the server with an evaluation key

$$\mathsf{EK} = (\mathsf{EK}_1, \mathsf{EK}_2) = \Big(\big[-(a \cdot s + e) + P \cdot s^2\big]_{P \cdot q}, a\Big),$$

where P is an integer parameter (which should be about as large as q to achieve low noise), a is a random polynomial in $\mathcal{R}_{P \cdot q}$, and e is a random small polynomial.

Essentially, the evaluation key is an encryption of $P \cdot s^2$ in the sense that

$$[\langle \mathsf{EK}, \mathsf{SK}\rangle]_{P\cdot q} \approx P \cdot s^2.$$

As a result,

$$\begin{aligned}\left[\left\langle (\mathsf{C}_1', \mathsf{C}_2') + \left\lfloor \frac{\mathsf{C}_3' \cdot q_\ell}{P \cdot q} \cdot \mathsf{EK}\right\rceil, (1, s)\right\rangle\right]_{q_\ell} &\approx [\langle(\mathsf{C}_1', \mathsf{C}_2', \mathsf{C}_3'), (1, s, s^2)\rangle]_{q_\ell} \\ &= \left(m^{(1)} + v^{(1)}\right)\left(m^{(2)} + v^{(2)}\right) \\ &\approx m^{(1)} m^{(2)},\end{aligned}$$

so $\left[(\mathsf{C}_1', \mathsf{C}_2') + \left\lfloor \frac{\mathsf{C}_3' \cdot q_\ell}{P \cdot q} \cdot \mathsf{EK}\right\rceil\right]_{q_\ell}$ is a ciphertext for $m^{(1)} m^{(2)}$.

Recalling that we actually want a ciphertext for $\frac{1}{\Delta} \cdot m^{(1)} m^{(2)}$, we rescale by dividing the ciphertext components and the modulus by Δ.

In particular we let $\mathsf{C}_{\text{mult}} = \left[\left\lfloor \frac{\mathsf{C}_1', \mathsf{C}_2'}{\Delta} + \frac{\mathsf{C}_3' \cdot q_{\ell-1}}{P \cdot q} \cdot \mathsf{EK}\right\rceil\right]_{q_{\ell-1}}$, recalling that

$$(\mathsf{C}_1', \mathsf{C}_2', \mathsf{C}_3') = \left[\left(\mathsf{C}_1^{(1)} \mathsf{C}_1^{(2)}, \mathsf{C}_1^{(1)} \mathsf{C}_2^{(2)} + \mathsf{C}_2^{(1)} \mathsf{C}_1^{(2)}, \mathsf{C}_2^{(1)} \mathsf{C}_2^{(2)}\right)\right]_q.$$

The new size $\|\mathsf{C}_{\text{mult}}\|_\infty^{\text{can}}$ is upper bounded by $\nu^{(1)} \nu^{(2)}$ and the new noise is upper bounded by

$$\frac{\nu^{(1)} B^{(2)} + \nu^{(2)} B^{(1)} + B^{(1)} B^{(2)} + B_{\text{mult}}}{\Delta}.$$

While we don't show a proof of the noise bound, we point out that the first three terms in the numerator come from the approximation $\left(m^{(1)} + v^{(1)}\right)\left(m^{(2)} + v^{(2)}\right) \approx m^{(1)} m^{(2)}$ while the B_{mult} comes from the the noise of the evaluation key and rounding operations. The denominator of Δ comes from the rescaling.

Note:

Adding and multiplying ciphertexts on different levels

Suppose we want to add two ciphertexts $\left(\mathsf{C}^{(1)}, \ell, \nu^{(1)}, B^{(1)}\right)$ and $\left(\mathsf{C}^{(2)}, \ell', \nu^{(2)}, B^{(2)}\right)$ on different levels, so say $\ell' < \ell$. Then we simply apply $\text{ModDecrease}_{\ell \to \ell'}$ to $\mathsf{C}^{(1)}$. In particular, we replace $\mathsf{C}^{(1)}$ with $\left[\mathsf{C}^{(1)}\right]_{q_{\ell'}}$ and so that we have two ciphertexts on the same level ℓ' and can now add them.

The same applies to multiplying two ciphertexts on different levels.

Adding and multiplying by constants

If instead of having ciphertexts for both $m^{(1)}$ and $m^{(2)}$ we have a ciphertext $\mathsf{C}^{(1)}$ for $m^{(1)}$ and direct access to $m^{(2)}$ (e.g. if $m^{(2)}$ is a public input), then we can compute a ciphertext for $m^{(1)} + m^{(2)}$ by first generating a ciphertext $\mathsf{C}^{(2)}$ for $m^{(2)}$, then performing addition of two ciphertexts $\mathsf{C}^{(1)}$ and $\mathsf{C}^{(2)}$.

Our ciphertext $\mathsf{C}^{(2)}$ for $m^{(2)}$ need not have noise because $m^{(2)}$ is public; in particular, we can simply set $\mathsf{C}^{(2)} = \left(m^{(2)}, 0\right)$.

The same applies to multiplying a ciphertext by a public input.

Recap

We now know how to add plaintexts homomorphically, by adding their ciphertexts component-wise. We also saw how to multiply plaintexts homomorphically and divide them by Δ.

Each time we perform a multiplication, our modulus gets divided by Δ. Thus, we begin with a modulus q divisible by Δ^L and are able to perform levelled homomorphic encryption on circuits of depth $\leq L$.

Stay tuned for part 2, where we will show a method for bootstrapping (i.e. refreshing the modulus q_ℓ of a ciphertext back to a larger one), which will help us achieve fully homomorphic encryption!