CKKS Homomorphic Encryption Part 2 - Bootstrapping

Notes by Linus Tang.

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

Sources:
- [CHKKS'18] The first bootstrapping for the CKKS scheme, by Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song
  - https://eprint.iacr.org/2018/153.pdf

Assumed background knowledge:
- Read and understand CKKS, either from Part 1 or from the original paper
  - https://docs.google.com/viewerng/viewer?url=https://www.mit.edu/~linust/files/CKKS_Homomorphic_Encryption_Part_1.pdf
  - https://eprint.iacr.org/2016/421.pdf

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 2: Bootstrapping to Achieve Fully Homomorphic Encryption

6. Big Picture and Review of Part 1

CKKS is a fully homomorphic scheme which performs *approximate* arithmeic.

- Encoding a message involves multiplying by Δ and rounding to the nearest integer. So, plaintexts have a small error before they are even encrypted. (In practice, if you want plaintexts encrypted with b bits of precision, choose Δ slightly larger than 2^b.)
- Each homomorphic operation incurs an increase in noise, and there is no mechanism for noise reduction. Parameters can be chosen to control this noise and achieve a desired level of precision in the output. (Although we did not spell this out explicitly in Part 1, messages should have absolute value $O(1)$ in order to keep noise in control. This fact is connected to the noise bound on ciphertext multiplication.)

Each ciphertext has a *level* indicating the size of its modulus. Multiplication decreases levels.

- In particular, when we multiply two ciphertexts, we divide the polynomials and modulus by Δ; this is called rescaling.
- For ease of reference, we let the possible moduli be $q_0, q_0 \cdot \Delta, q_0 \cdot \Delta^2, ..., q_0 \cdot \Delta^L = q$. Then a ciphertext with modulus $q_0 \cdot \Delta^\ell$ is said to have level ℓ.
- When we multiply two ciphertexts with levels ℓ_1 and ℓ_2, we first bring both down to level $\min(\ell_1, \ell_2)$, and after rescaling, the product has level $\min(\ell_1, \ell_2) - 1$.
- This means that we have achieved levelled homomorphic encryption, where we can evaluate circuits of multiplicative depth at most L. (The cost of making L large is that q becomes very large and basic multiplications mod q become very expensive.)

The bootstrapping operation has the effect of increasing the level of a ciphertext.

- If we have a magic box that allows us to put in a ciphertext with low level and get out a ciphertext (of the same message, with only slightly increased noise) with high level, then we are no longer constrained to evaluating low-depth circuits.
- This magic box is precisely what we will construct in these notes (Part 2).

7. Tools for Bootstrapping

Key switching

Suppose we have a ciphertext $\mathsf{C} = (\mathsf{C}_1, \mathsf{C}_2)$ encrypting plaintext m under secret key $\mathsf{SK} = (1, s)$. In particular,

$$[\langle \mathsf{C}, \mathsf{SK} \rangle]_{q_\ell} \approx m.$$

We want to obtain a ciphertext C' encrypting the same plaintext m under a new key $\mathsf{SK}' = (1, s')$. In order to perform this operation, we clearly need some additional information that relates s and s', without revealing either key to us. It turns out that it suffices to have an encryption of s under SK'. Let

$$\mathsf{KSK}_{\mathsf{SK}'}(s) = \Big([-(a \cdot s' + e) + P \cdot s]_{P \cdot q}, a \Big),$$

where a is a random polynomial in $\mathcal{R}_{P \cdot q}$, and e is a random small polynomial. We call this the *key-switching key.* In particular, we have

$$[\langle \mathsf{KSK}_{\mathsf{SK}'}(s), \mathsf{SK}' \rangle]_{P \cdot q} \approx s$$

$$\left[\left\langle \left\lfloor \frac{\mathsf{KSK}_{\mathsf{SK}'}(s) \cdot q_\ell}{P \cdot q} \right\rceil, \mathsf{SK}' \right\rangle \right]_{q_\ell} \approx s.$$

It follows that

$$
\begin{aligned}
m &\approx [\langle \mathsf{C}, \mathsf{SK} \rangle]_{q_\ell} \\
&= [\langle \mathsf{C}, (1, s) \rangle]_{q_\ell} \\
&\approx \left[\left\langle (\mathsf{C}_1, 0) + \left\lfloor \frac{\mathsf{KSK}_{\mathsf{SK}'}(s) \cdot q_\ell}{P \cdot q} \cdot \mathsf{C}_2 \right\rceil, (1, s') \right\rangle\right]_{q_\ell} \\
&= \left[\left\langle (\mathsf{C}_1, 0) + \left\lfloor \frac{\mathsf{KSK}_{\mathsf{SK}'}(s) \cdot q_\ell}{P \cdot q} \cdot \mathsf{C}_2 \right\rceil, \mathsf{SK}' \right\rangle\right]_{q_\ell} .
\end{aligned}
$$

Thus, $(\mathsf{C}_1, 0) + \left\lfloor \frac{\mathsf{KSK}_{\mathsf{SK}'}(s) \cdot q_\ell}{P \cdot q} \cdot \mathsf{C}_2 \right\rceil$ is an encryption of m under the new secret key SK' with a slightly increased noise bound. The increase in the noise bound comes from the randomness and rounding of the key-switching key.

To summarize: Key switching is an operation whose inputs are
- ciphertext C encrypting plaintext m under secret key $\mathsf{SK} = (1, s)$
- an encryption $\mathsf{KSK}_{\mathsf{SK}'}(s)$ of s under a new secret key $\mathsf{SK}' = (1, s')$

and whose output is a ciphertext C' encrypting the same plaintext m under the new key SK'.

Rotation and conjugation

We will see here that key switching directly gives us a way to perform certain cyclic permutations (rotations) on the plaintext slots.

For $\mathsf{C} = (\mathsf{C}_1, \mathsf{C}_2) = (\mathsf{C}_1(X), \mathsf{C}_2(X))$, let $\mathsf{C}(X^k)$ denote $(\mathsf{C}_1(X^k), \mathsf{C}_2(X^k))$. Define the notation $\mathsf{SK}(X^k)$ similarly.

Recall that N is a power of 2 and assume $N \geq 4$. The plaintext slots essentially correspond to the evaluations of m on roots $\zeta, \zeta^3, \zeta^5 ..., \zeta^{2N-1}$ of the polynomial modulus $X^N + 1$. Actually, we only care about half of the evaluations, because m has real coefficients and thus commutes with conjugation (in particular, $m(\zeta^i) = \overline{m(\zeta^{2N-i})}$).

In Part 1, we naively chose the representatives $\zeta, \zeta^3, \zeta^5 ..., \zeta^{N-1}$ of the $\frac{N}{2}$ pairs of conjugates to be evaluated/interpolated. It turns out that a more convenient choice is $\zeta, \zeta^5, \zeta^9, ..., \zeta^{2N-3}$, i.e. the 1 mod 4 exponents of ζ. This is because $\{1, 5, 9, ..., 2N-3\}$ is a cyclic multiplicative group (mod $2N$) generated by the element 5. (Mathematical exercise: the generators are precisely the 5 mod 8 elements.) This requires a (conceptually inconsequential) change to our encoding and decoding method. In particular, we will now say that we decode plaintexts by

$$
\boldsymbol{z} \approx \frac{1}{\Delta}\left(m_{\Delta \boldsymbol{z}}(\zeta), m_{\Delta \boldsymbol{z}}(\zeta^5), m_{\Delta \boldsymbol{z}}(\zeta^9), ..., m_{\Delta \boldsymbol{z}}(\zeta^{2N-3})\right)
$$

and modify our encoding method accordingly.

Why is this useful? Observe that if $m(X)$ decodes to $\boldsymbol{z}$, then $m(X^5)$ decodes to $\pi(\boldsymbol{z})$, where π is some cyclic permutation on the slots (components) of $\boldsymbol{z}$. More generally, $m\left(X^{5^j}\right)$ decodes to $\pi^j(\boldsymbol{z})$.

Now, if you have a ciphertext C encrypting message m under secret key $\mathsf{SK} = (1, s)$, then

$$
\begin{aligned}
m(X) &\approx [\langle \mathsf{C}(X), \mathsf{SK}(X) \rangle]_{q_\ell} \\
&\approx \left[\left\langle \mathsf{C}'(X), \mathsf{SK}\left(X^{5^{-j}}\right) \right\rangle\right]_{q_\ell},
\end{aligned}
$$

where C' can be obtained by a key switch using the key-switching key $\mathsf{KSK}_{\mathsf{SK}\left(X^{5^{-j}}\right)}(s)$. Plugging in $X \mapsto X^{5^j}$ tells us that

$$m\left(X^{5^j}\right) \approx \left[\left\langle \mathsf{C}'\left(X^{5^j}\right), \mathsf{SK}(X)\right\rangle\right]_{q_\ell}.$$

So, we have a way to take an encryption of $m(X) = m_{\Delta \boldsymbol{z}}(X)$ and produce an encryption $\mathsf{C}'\left(X^{5^j}\right)$ of $m\left(X^{5^j}\right) \approx m_{\Delta \pi^j(\boldsymbol{z})}(X)$ (under the same key).

Conjugation is essentially the same process as rotation, except with an exponent of -1 instead of 5^j, so we discuss it briefly. Given C encrypting m and the appropriate key-switching key, we can find C' such that

$$\begin{aligned} m(X) &\approx [\langle \mathsf{C}(X), \mathsf{SK}(X)\rangle]_{q_\ell} \\ &\approx \left[\left\langle \mathsf{C}'(X), \mathsf{SK}\left(X^{-1}\right)\right\rangle\right]_{q_\ell}, \end{aligned}$$

and consequently,

$$m\left(X^{-1}\right) \approx \left[\left\langle \mathsf{C}'\left(X^{-1}\right), \mathsf{SK}(X)\right\rangle\right]_{q_\ell}.$$

So we have taken an encryption of $m(X) = m_{\Delta \boldsymbol{z}}(X)$ and produced an encryption $\mathsf{C}'\left(X^{-1}\right)$ of $m\left(X^{-1}\right) \approx m_{\Delta \overline{\boldsymbol{z}}}$.

To summarize: Given certain key-switching keys, we are able to perform a rotation operation, which applies a certain cyclic permutation to the message $\boldsymbol{z}$. We can also perform the conjugation operation, which transforms $m(X)$ to $m(X^-1)$ and thus transforms $\boldsymbol{z}$ to $\overline{\boldsymbol{z}}$. (Like all other homomorphic operations, these are approximate and add to any previous error.)

Linear transformations

Note that every $\mathbb{R}$-linear transformation $\mathbb{C}^{N/2} \to \mathbb{C}^{N/2}$ (i.e. from the message space to itself) can be represented as $\boldsymbol{z} \mapsto A \cdot \boldsymbol{z} + B \cdot \overline{\boldsymbol{z}}$, where A and B are $N/2 \times N/2$ matrices.

Suppose that we have a ciphertext for $m_{\Delta \boldsymbol{z}}$ and public matrices A and B, and we would like to obtain a ciphertext for $m_{\Delta (A \cdot \boldsymbol{z} + B \cdot \overline{\boldsymbol{z}})}$.

We have constructed homomorphic operations which do the following (in the message space).

- Addition:
  - $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)} \mapsto \boldsymbol{z}^{(1)} + \boldsymbol{z}^{(2)}$.
- Componentwise multiplication:
  - $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)} \mapsto \boldsymbol{z}^{(1)} \odot \boldsymbol{z}^{(2)}$.
- Rotation:
  - $\boldsymbol{z} \mapsto \pi^j(\boldsymbol{z})$
- Conjugation:
  - $\boldsymbol{z} \mapsto \overline{\boldsymbol{z}}$

We encourage the reader to find a simple way to chain together these operations to achieve arbitrary linear transformations $\boldsymbol{z} \mapsto A \cdot \boldsymbol{z} + B \cdot \overline{\boldsymbol{z}}$, before reading on.

It suffices to demonstrate how to multiply $\boldsymbol{z}$ by an arbitrary matrix A, since the overall transformation consists of a conjugation, two matrix multiplications, and an addition.

Note that componentwise multiplication is multiplication by a diagonal matrix. What we mean is that if $\boldsymbol{a} = \left(a_1, a_2, ..., a_{N/2}\right)$, then

$$\boldsymbol{a} \odot \boldsymbol{z} = \begin{pmatrix} a_1 & & & \\ & a_2 & & \\ & & \ddots & \\ & & & a_{N/2} \end{pmatrix} \cdot \boldsymbol{z}.$$

Furthermore, $\boldsymbol{a} \odot \pi^j(\boldsymbol{z})$ is equivalent to multiplication by another sparse matrix, with entries in locations given by π^j, say

$$\boldsymbol{a} \odot \pi^j(\boldsymbol{z}) = \begin{pmatrix} & & a_1 & \\ & & & a_2 \\ & & \vdots & \\ a_{N/2} & & & \end{pmatrix} \cdot \boldsymbol{z}.$$

Because the permutation π is a cycle of order $N/2$, it follows that the nonzero positions of the sparse matrices corresponding to $\pi^0, \pi^1, ..., \pi^{N/2-1}$ cover all $N/2 \times N/2$ positions.

In particular, we can construct vectors $\boldsymbol{a}_0, \boldsymbol{a}_1, ..., \boldsymbol{a}_{N/2-1}$ such that

$$\boldsymbol{a}_0 \odot \pi^0(\boldsymbol{z}) + \boldsymbol{a}_1 \odot \pi^1(\boldsymbol{z}) + \cdots + \boldsymbol{a}_{N/2-1} \odot \pi^{N/2-1}(\boldsymbol{z}) = A \cdot \boldsymbol{z}.$$

Specifically, the i-th entry of $\boldsymbol{a}_j$ is the $\pi^j(i)$-th entry of the i-th row of A. This solves the problem of multiplying a message by an arbitrary public matrix A.

To spell things out completely:

Suppose that we have a ciphertext C for $m_{\Delta \boldsymbol{z}}$ and public matrices A and B, and we would like to obtain a ciphertext C' for $m_{\Delta(A \cdot \boldsymbol{z} + B \cdot \overline{\boldsymbol{z}})}$.

We construct vectors $\boldsymbol{a}_0, ..., \boldsymbol{a}_{N/2-1}$ from the entries of A, and similarly construct $\boldsymbol{b}_0, ..., \boldsymbol{b}_{N/2-1}$ from the entries of B. It holds that

$$\left(\sum_{j=0}^{N/2-1} \boldsymbol{a}_j \odot \pi^j(\boldsymbol{z}) \right) + \left(\sum_{j=0}^{N/2-1} \boldsymbol{a}_j \odot \pi^j(\overline{\boldsymbol{z}}) \right) = A \cdot \boldsymbol{z} + B \cdot \overline{\boldsymbol{z}}.$$

In order to perform the computation on the left side homomorphically (over ciphertexts), we need:

- A ciphertext C for $m_{\Delta \boldsymbol{z}}$
- An ciphertext $\mathsf{C}_{\boldsymbol{a}_j}$ for an encoding of each $\boldsymbol{a}_j$ (which can be a noiseless encryption $\left(m_{\Delta \boldsymbol{a}_j}, 0\right)$ since $\boldsymbol{a}_j$ is public)
- Ditto for $\boldsymbol{b}_j$
- Key-switching keys $\mathsf{KSK}_{\mathsf{SK}\left(X^{5^{-j}}\right)}(s)$ and $\mathsf{KSK}_{\mathsf{SK}(X^{-1})}(s)$ for rotation and conjugation.

8. Bootstrapping Procedure

Modulus switching

If we have a ciphertext C for message m at level 0 (i.e. modulus q_0), it cannot participate in any further multiplications, so we need to bootstrap it. The output of bootstrapping is a ciphertext C_{boot} which encrypts the same message m but at a higher level $L - \ell_{\text{boot}}$ (i.e. modulus $q_{L-\ell_{\text{boot}}} = q_0 \cdot \Delta^{L-\ell_{\text{boot}}}$).

The first step is to raise $(\mathsf{C}, 0, \nu, B)$ to a ciphertext $(\mathsf{C}_{\text{raise}}, L, \nu_{\text{raise}}, B)$ of level L.

The new ciphertext $\mathsf{C}_{\text{raise}}$ will have the same polynomial components $(\mathsf{C}_1, \mathsf{C}_2)$ as C. However, note that the new ciphertext no longer encrypts the same message, because decryption is mod q_L instead of q_0. Specifically, if C encrypts $m \approx [\langle \mathsf{C}, \mathsf{SK} \rangle]_{q_0}$, then $\mathsf{C}_{\text{raise}}$ encrypts

$$m + q_0 t \approx [\langle \mathsf{C}, \mathsf{SK} \rangle]_{q_L}$$

for some polynomial t. Since $\mathsf{SK} = (1, s)$ where s is small, t must also be small.

The rest of the bootstrapping procedure consists of a circuit which recovers m from $m + q_0 t$. Since this circuit has some multiplicative depth ℓ_{boot}, the final ciphertext for m will have level $L - \ell_{\text{boot}}$.

From here onwards, in our notation, we ignore the fact that homomorphic operations incur slowly increasing amounts of error, and use "$=$" even where "$\approx$" would be more appropriate. It should become clear that because we are applying homomorphic operations that each only increase the noise by a small amount, the overall result is also just small increase in noise.

Erasing the multiple of q_0

Our goal is now to homomorphically evaluate a circuit that inputs $m' = m + q_0 t$, (where m and t are small polynomials) and outputs m. Since the circuit needs to be evaluated homomorphically, its gates should be additions, multiplications, rotations, conjugations, and linear transformations (since these are the operations that we know how to perform homomorphically).

If we let $m'(X) = m'_0 + m'_1 X + \dots + m'_{N-1} X^{N-1}$ and $m(X) = m_0 + m_1 X + \dots + m_{N-1} X^{N-1}$, then we see that $m_i = [m'_i]_{q_0}$ for each i. In particular, the operation we are trying to compute consists of performing the same operation over the N coefficients. CKKS excels at parallel computation, because each homomorphic operation as a parallel operation over $N/2$ message slots. Therefore, the overall plan will be as follows:

1. **Move coefficients to slots:** Transform m' into two polynomials p'_1 and p'_2, such that the N coefficients of m' get stored in the $N/2$ slots each of p'_1 and p'_2. Formally,
$$(m'_0, \dots, m'_{N-1}) = (p'_1(\zeta), p'_1(\zeta^5), \dots, p'_1(\zeta^{2N-3}), p'_2(\zeta), p'_2(\zeta^5), \dots, p'_2(\zeta^{2N-3})).$$
2. **Erase the multiple of q_0:** Suppose that we have a circuit whose gates are multiplications and additions (i.e. a polynomial circuit) such that when its input can be written as $m_i + q_0 t_i$ for small m_i and small t_i, the output is approximately m_i. Then we run this circuit homomorphically on the ciphertexts of p'_1 and p'_2, resulting in p_1 and p_2. Now,
$$\left([m'_0]_{q_0}, \dots, [m'_{N-1}]_{q_0}\right) = (p_1(\zeta), p_1(\zeta^5), \dots, p_1(\zeta^{2N-3}), p_2(\zeta), p_2(\zeta^5), \dots, p_2(\zeta^{2N-3}))$$
.
3. **Move slots to coefficients:** Finally, perform the inverse of the first step in order to put $\left([m'_0]_{q_0}, \dots, [m'_{N-1}]_{q_0}\right) = (m_0, \dots, m_{N-1})$ back in the coefficients of the polynomial.

If we can fill in the details of the three above steps, then we can turn our level 0 encryption of m into a level L encryption of $m' = m + q_0 t$ (modulus switching), and turn that into a level $L - \ell_{\text{bootstrap}}$ encryption of m (erasing the multiple of q_0).

This would achieve the goal of bootstrapping, which is to begin with a level 0 encryption of a plaintext and end with a high level encryption of the same plaintext.

Steps 1 and 3, moving between coefficients and slots, is easy given our current technology. This is because they are both linear transformations!

Let's dive into the specifics of this transformation.

Say p'_1 encodes $z'_1 = \frac{1}{\Delta} \cdot \left(m'_0, m'_1, \dots, m'_{\frac{N}{2}-1}\right)$ and p'_2 encodes $z'_2 = \frac{1}{\Delta} \cdot \left(m'_{\frac{N}{2}}, m'_{\frac{N}{2}+1}, \dots, m'_{N-1}\right)$.

Meanwhile, m' encodes $z' = \frac{1}{\Delta} \cdot (m'(\zeta), m'(\zeta^5), \dots, m'(\zeta^{2N-3}))$.

Therefore, the two linear transformations that we want to evaluate when moving coefficients to slots are the ones which compute these two maps:

$$(m'(\zeta), m'(\zeta^5), ..., m'(\zeta^{2N-3})) \mapsto \left(m'_0, m'_1, ..., m'_{\frac{N}{2}-1}\right)$$
$$(m'(\zeta), m'(\zeta^5), ..., m'(\zeta^{2N-3})) \mapsto \left(m'_{\frac{N}{2}}, m'_{\frac{N}{2}+1}, ..., m'_{N-1}\right)$$

The components on the left are evaluations of the polynomial m', and the components on the right are coefficients. So, the relevant matrices are L_1^{-1} and L_2^{-1}, where

$$L_1 = \begin{pmatrix} 1 & \zeta & \zeta^2 & \dots & \zeta^{\frac{N}{2}-1} \\ 1 & \zeta^5 & \zeta^{5\cdot 2} & \dots & \zeta^{5\cdot(\frac{N}{2}-1)} \\ 1 & \zeta^9 & \zeta^{9\cdot 2} & \dots & \zeta^{9\cdot(\frac{N}{2}-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta^{2N-3} & \zeta^{(2N-3)\cdot 2} & \dots & \zeta^{(2N-3)\cdot(\frac{N}{2}-1)} \end{pmatrix}$$

and

$$L_2 = \begin{pmatrix} 1 & \zeta^{\frac{N}{2}} & \zeta^2 & \dots & \zeta^{\frac{N}{2}-1} \\ 1 & \zeta^{5\cdot\frac{N}{2}} & \zeta^{5\cdot(\frac{N}{2}+1)} & \dots & \zeta^{5\cdot(\frac{N}{2}-1)} \\ 1 & \zeta^{9\cdot\frac{N}{2}} & \zeta^{9\cdot(\frac{N}{2}+1)} & \dots & \zeta^{9\cdot(\frac{N}{2}-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta^{2N-3} & \zeta^{(2N-3)\cdot 2} & \dots & \zeta^{(2N-3)\cdot(\frac{N}{2}-1)} \end{pmatrix}.$$

To spell things out, we have

$$z'_1 = L_1^{-1} \cdot z' \quad \text{and} \quad z'_2 = L_2^{-1} \cdot z'.$$

We know how to compute on a ciphertext so as to multiply its message by a public matrix, so we can complete step 1.

Naturally, step 3, moving slots back to coefficients, uses the inverse linear transformations L_1 and L_2. Specifically, if $[m']_{q_0}$, p_1, and p_2 encode z, z_1, and z_2, then

$$z = L_1 \cdot z_1 + L_2 \cdot z_2.$$

So, all that's left to do is to perform step 2. Recall that the goal is to write a polynomial F such that whenever m_i and t_i are small, $F(m_i + q_0 t_i) \approx m_i$. Then, we write the polynomial F as a circuit with addition and multiplication gates, and evaluate this circuit homomorphically on the ciphertexts of p'_1 and p'_2 to get p_1 and p_2. Desired properties of this circuit are that it has not too many multiplication gates and that intermediate ciphertexts are not too large (because ciphertext size affects noise growth).

To give the size bound a name, we say that $t_i \in [-K, K]$. In practice, the bound K that we can guarantee will mainly depend on the degree N of our polynomials and the size of the coefficients of the secret key.

The first key idea is that it suffices to get a polynomial approximation of a sine wave (scaled horizontally and vertically to fit our needs). Specifically, we want to approximate $f(x) = \frac{q_0}{2\pi} \cdot \sin\left(\frac{2\pi x}{q_0}\right)$ on the interval $x \in [-Kq_0, Kq_0]$.

When $m_i \ll q_0$ (which we can easily guarantee by choosing q_0 to be sufficiently large), we have that $f(m_i + q_0 t_i)$ is a very good approximation of m_i; in particular, we can show that it's off by about $\frac{m_i^3}{q_0^2}$ by considering the Taylor series of the sine function.

For readability and intuition, we scale our variables so that we can pretend we are simply attempting to find a polynomial approximation for $\sin(x)$ which is accurate on the interval $x \in [-2K\pi, 2K\pi]$.

As a starting point, we can obtain a polynomial approximation of $\sin(x)$ by its n-th degree Taylor series for some odd n,

$$S_n(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \pm \frac{x^n}{n!}.$$

But in order for this approximation to be good on the interval $[-2K\pi, 2K\pi]$ (i.e. for $2K$ oscillations), we need $n = \Theta(K)$. An issue with this is that we run into problems of numerical stability. In particular, if we aren't careful about how we write S_n as a circuit, we may have an intermediate computation of x^n, which is too large (as evidenced the fact that we have to multiply it by $\frac{1}{n!}$ just to bring its size "back down to Earth" in the Taylor series).

While there may be hacky ways to compute S_n while mitigating the issue of numerical stability, there is a much nicer and more practical solution we can employ instead.

We use the formula $\sin(x) = \frac{e^{ix} - e^{-ix}}{2i}$. In particular, we choose paradmeters d and k, and first estimate $e^{ix/2^k}$ with a degree-d Taylor series, then square it k times, resulting in an approximation for e^{ix}. (Then we use a key switch to find the conjugate e^{-ix} and finally get $\frac{e^{ix} - e^{-ix}}{2i}$.)

To write things out, we have

$$\begin{aligned} e^{ix} &= \left(e^{ix/2^k}\right)^{2^k} \\ &= \left(\dots\left(\left(e^{ix/2^k}\right)^2\right)^2\dots\right)^2 \\ &\approx \left(\dots\left(\left(1 + \frac{ix/2^k}{1!} + \frac{\left(ix/2^k\right)^2}{2!} + \dots + \frac{\left(ix/2^k\right)^d}{d!}\right)^2\right)^2\dots\right)^2. \end{aligned}$$

Investigating the optimal choice of d and k is outside the scope of these notes, but we will note that we can achieve lower approximation error with better numerical stability while having $d, k \ll K$ (allowing us to use fewer multiplications), compared to the attempt of approximating $\sin(x)$ by $S_n(x)$.

As mentioned earlier, once we approximate e^{ix}, we can use a key switch to find its conjugate e^{-ix} and arrive at an approximation for $\sin(x) = \frac{e^{ix} - e^{-ix}}{2}$.

Summary

In order to bootstrap a ciphertext C at level 0 encrypting message m:

- Write a level L ciphertext $\mathsf{C}_{\text{raise}}$ which is otherwise identical to C. Then $\mathsf{C}_{\text{raise}}$ encrypts $m + q_0 t$ for some small polynomial t. We now want to get rid of the $q_0 t$ term.
- We take two linear transformations which move the coefficients of $m' = m + q_0 t$ to the message slots of two polynomials p_1' and p_2', so that we can perform parallel computations on these coefficients.
- We find a polynomial which approximates a certain sine wave, transforming the coefficients of m' (now in message slots) to coefficients of m.
- We move the coefficients of m back from the message slots to the coefficients of a plaintext.
- Our resulting ciphertext encrypts m and has high level, achieving the goal of bootstrapping.