CKKS Homomorphic Encryption Part 3 - Improved Bootstrapping

Notes by Linus Tang.

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

Thanks to Brian Lawrence, Noah Walsh, and Colin Tang for working with me to better understand Chebyshev interpolants!

Sources:

- [CHKKS'18a] The first bootstrapping for the CKKS scheme
  - Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song
  - https://eprint.iacr.org/2018/153.pdf
- [CHKKS'18b] A Full RNS Variant of Approximate Homomorphic Encryption
  - Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, Yongsoo Song
  - https://eprint.iacr.org/2018/931.pdf
- [CCS'18] Improvements on the bootstrapping
  - Hao Chen, Ilaria Chillotti, and Yongsoo Song
  - https://eprint.iacr.org/2018/1043.pdf
- [CHH'18] More improvements on the bootstrapping, and homomorphic DFT
  - Jung Hee Cheon, Kyoohyung Han, and Minki Hhan
  - https://eprint.iacr.org/2018/1073.pdf
- [HK'19] Better Bootstrapping for Approximate Homomorphic Encryption
  - Kyoohyung Han, Dohyeong Ki
  - https://eprint.iacr.org/2019/688.pdf
- [BMTH'19] Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-Sparse Keys
  - Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Pierre Hubaux
  - https://eprint.iacr.org/2020/1203.pdf

Assumed background knowledge:

- Read and understand CKKS, from Parts 1 & 2 of my notes or from the papers they were based on.
  - www.mit.edu/ linust/files/CKKS_Homomorphic_Encryption_Part_1.pdf
  - www.mit.edu/ linust/files/CKKS_Homomorphic_Encryption_Part_2.pdf
  - https://eprint.iacr.org/2016/421.pdf
  - https://eprint.iacr.org/2018/153.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.

If there is a Part 4, it will likely cover techniques that focus on improving the precision of the scheme.

Contents

In case you want to read the papers as well:

- Paterson-Stockmeyer and Chebyshev interpolants are introduced in [CCS'18] section 4 and refined by [HK'19].
- Bootstrapping of sparse ciphertexts is covered by [CHKKS'18a] section 4.4.
- Fourier transforms are covered by [CCS'18] section 3 and [CHH'18] section 3.
- The residue number system is introduced in [CHKKS'18b].
- Improved key switching is in [BMTH'19] section 4.1.
- Improved matrix multiplication is in [BMTH'19] sections 4.2 and 4.3.

Part 3: Improvements on Bootstrapping

9. Paterson-Stockmeyer and Chebyshev interpolants

Recall that one of the steps in bootstrapping was to find a polynomial which serves as a good approximation to a certain sine wave (which we transformed to $\sin(x)$ for simplicity). The approximation needs to be accurate for all x in some interval $[-2\pi K, 2\pi K]$. We reduced the problem to estimating the complex exponential e^{ix}, which we did by taking a degree-d Taylor series for $e^{ix/2^k}$ and squaring it k times. Specifically,

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

Our goal is to be able to compute this approximation with very low complexity (few multiplications) and low depth. Furthermore, the intermediate steps in the computation should not be too large, because the noise growth of a homomorphic multiplication depends on the size of the plaintext. We call this last property numerical stability.

As a subtle note, multiplications by public values (e.g. coefficients of our approximation polynomial) do not contribute to the complexity analysis, because the corresponding multiplications on the ciphertexts can be done very efficiently.

In this section we present four methods of evaluating a polynomial P of degree d on an input x (or equivalently, writing P as a circuit with input x). We analyze the complexity, depth, and numerical stability of these circuits. In the analysis, we omit $(1 + o(1))$ factors.

Keep in the back of your mind that for the purposes of bootstrapping we want to approximate $\sin(x)$ accurately on $[-2\pi K, 2\pi K]$. In particular,

- Choose $P(x)$ to be some degree-d polynomial approximation of e^{ix}, say for now the Taylor series.
- Divide the input by 2^k and evaluate P homomorphically (to approximate $e^{ix/2^k}$).
- Square the result k times (to approximate e^{ix}).
- Take the imaginary component (to approximate $\sin(x)$).

The optimal choice of k and d depend on K and on the required precision of the bootstrapping. Details of choosing k and d are outside the scope of these notes.

There is a table at the end of this section which summarizes the analysis of the various circuits, which may be helpful to refer to while reading this section.

Horner evaluation

First, consider a the following way of evaluating a polynomial $P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d$.

We can compute this polynomial as

$$P(x) = (\cdots((a_d x + a_{d-1})x + a_{d-2})x + \cdots + a_1)x + a_0.$$

This is called Horner evaluation. It achieves good numerical stability but takes about d depth and d multiplications, which can be improved.

Folding evaluation

We can decrease the depth by recursively "folding" the polynomial in half:
1. Let d_1 be the greatest power of 2 less than d.
2. By repeated squaring, compute $x, x^2, x^4, ..., x^{d_1}$.
3. Write $P(x) = x^{d_1} P_{\text{low}}(x) + P_{\text{high}}(x)$, where P_{low} and P_{high} have degree at most d_1.
4. Recursively compute $P_{\text{low}}(x), P_{\text{high}}(x)$ (skipping step 2 since these powers have already been computed).
5. Compute $P(x) = x^{d_1} P_{\text{low}}(x) + P_{\text{high}}(x)$.

This has a complexity of about d but the depth has decreased to $\log d$. However, this is not numerically stable because we have large powers of x as intermediate calculations.

So, folding evaluation offers a way to reduce the depth, but it loses numerical stability.

Note. The name "folding evaluation" is made-up and you probably won't find relevant results online by searching for it.

Paterson-Stockmeyer evaluation

The Paterson-Stockmeyer algorithm gives us a way to reduce the complexity to $2\sqrt{d}$ while keeping roughly the same depth.
1. Assume (by padding the tuple of coefficients with zeros) that d can be written as $d = uv - 1$ with $u \approx v$.
2. Compute $1, x, x^2, ..., x^u$, by using the formula $x^j = x^{\lceil j/2 \rceil} \cdot x^{\lfloor j/2 \rfloor}$.
3. Write
$$P(x) = Q_0(x) + x^u Q_1(x) + x^{2u} Q_2(x) + \cdots + x^{(v-1)u} Q_{u-1}(x),$$
where each P_i has degree at most $u - 1$.
4. Use the powers $1, x, x^2, ..., x^{u-1}$ computed earlier to calculate all of the Q_i. Note that all multiplications involved are by constants and are therefore cheap to run homomorphically.
5. The above expression for $P(x)$ can be thought of as a degree v polynomial applied to x^u whose coefficients are Q_i. We can apply the folding evaluation to this polynomial!

Step 2 incurs u complexity and $\log u$ depth, and step 5 incurs v complexity and $\log v$ depth. So we have that the overall complexity and depth are $2\sqrt{d}$ and $\log d$, respectively.

However, we still suffer from numerical instability because step 5 requires us to compute a large power of x as an intermediate step.

Stockmeyer-Paterson, modified using Chebyshev polynomials

There is a sequence $T_0, T_1, ...$ of polynomials, called Chebyshev polynomials, satisfying the following:
1. $T_n(\cos(\theta)) = T(\cos(n\theta))$ for all nonnegative integers n and all θ.
2. Consequently, $T_{n_1 n_2} = T_{n_1} \circ T_{n_2}$ for all nonnegative integers n_1, n_2.
3. T_n has degree n for all nonnegative integers n.
4. Recursive formula: $T_0(x) = 0$, $T_1(x) = x$, and $T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x)$ for $n \geq 2$.

For a proof of the above, see https://www.mit.edu/~linust/files/Chebyshev_Interpolant.pdf.

We can modify Paterson-Stockmeyer evaluation to evaluate a polynomial P of degree d with similar multiplicative depth and complexity, while achieving numerical stability for x on the interval $[-1, 1]$ (or in fact any small interval, by applying the appropriate scaling before and after).

1. Assume (by padding the tuple of coefficients with zeros) that d can be written as $d = uv - 1$ with $u \approx v$ and u a power of 2.
2. Compute $T_0(x), ..., T_u(x)$, using $T_{2j}(x) = T_2\big(T_j(x)\big)$ and $T_{2j+1}(x) = 2xT_{2j}(x) - T_{2j-1}(x)$.
3. At the corresponding step in the original Paterson-Stockmeyer algorithm, we wrote
$$P(x) = Q_0(x) + x^u Q_1(x) + x^{2u} Q_2(x) + \cdots + x^{(v-1)u} Q_{u-1}(x),$$
using $1, x^u, x^{2u}, ..., x^{(v-1)u}$ as a "sparse basis." It maybe tempting to replace the sparse basis with $T_0(x), T_u(x), T_{2u}(x), ..., T_{(v-1)u}(x)$. However, this basis does not work will with the folding in step 5. Instead, we want to work with the basis $\mathcal{T}_{(j,u)_{0 \le j < v}}$ where
$$(\mathcal{T}_{j,u}) = \prod_{k \in \text{ Bin}(j)} T_k(x).$$
Here, $\text{Bin}(j)$ is unique set of powers of 2 whose sum is j. Specifically, we write
$$P(x) = \mathcal{T}_{0,u} Q_0(x) + \mathcal{T}_{1,u} Q_1(x) + \mathcal{T}_{2,u} Q_2(x) + \cdots + \mathcal{T}_{v-1,u} Q_{u-1}(x),$$
where the Q_j have degree at most $u - 1$.
4. Write each Q_j in the Chebyshev basis, and compute $Q_j(x)$ using scalar multiplications with $T_0(x), ..., T_{u-1}(x)$.
5. Finally, use folding evaluation to evaluate $P(x)$. This involves computing $T_u(x), T_{2u}(x), T_{4u}(x), ...$, by repeatedly applying the identity $T_{2j}(x) = T_2\big(T_j(x)\big)$.

We claimed earlier that this modified version of Paterson-Stockmeyer achieves numerical stability while the original version does not. The reason for this is that we no longer have large powers x^j as intermediate computations, but rather Chebyshev polynomials T_j evaluated at x. Recalling the identity $T_n(\cos(\theta)) = T(\cos(n\theta))$, it follows that T_j is small (in particular, $x \in [-1, 1]$ implies $T_j(x) \in [-1, 1]$), so our computations are numerically stable, provided that the coefficients of the Q_i in the Chebyshev basis are reasonably sized. It turns out that they are, for the exponential function $e^{ix/2^k}$ that we're trying to approximate, but we don't prove this.

Here's a table comparing the complexity, depth, and numerical stability of the evaluations above. We add the term k to the complexity and depth to account for the fact that we approximate $e^{ix/2^k}$ and square the result k times.

Evaluation method	Complexity	Depth	Numerical stability
Horner evaluation	$d + k$	$d + k$	Good
Folding evaluation	$d + k$	$\log d + k$	Bad
Paterson-Stockmeyer	$2\sqrt{d} + k$	$\log d + k$	Bad
Paterson-Stockmeyer on Chebyshev	$2\sqrt{d} + k$	$\log d + k$	Good

10. Interpolation for estimation

Chebyshev interpolation

There is a polynomial, called the Chebyshev interpolant, which often approximates functions better than the Taylor series.

For a more detailed overview of the Chebyshev interpolant, with proofs, see the notes below.

- https://www.mit.edu/~linust/files/Chebyshev_Interpolant.pdf

Otherwise, we summarize the relevant results here:

- We are trying to approximate $f : [-1, 1] \to \mathbb{R}$ on the interval and want a degree $\leq n-1$ polynomial P_n that is accurate in the worst case, i.e. we want

$$\|f - P_n\|_{L_\infty} = \max_{x \in [x_{\min}, x_{\max}]} |f(x) - P_n(x)|$$

  to be small.
- Suppose that the n-th derivate $f^{(n)}$ is continuous and $|f^{(n)}|$ is bounded by M.
- Consider $x_0, ..., x_{n-1} \in [-1, 1]$. There is a unique polynomial P_n of degree at most $n-1$ such that $P_n(x_b) = f(x_b)$ for all $k = 0, ..., n-1$, and it achieves the bound

$$|f(x) - P_n(x)| \leq \frac{M}{n!} \prod_{b=0}^{n-1} (x - x_b).$$

- The Chebyshev interpolant $\mathcal{P}_n$ uses the interpolation points

$$x_b^* = \cos\left(\frac{(b + \frac{1}{2})\pi}{n}\right)$$

  because these points minimize

$$\max_{x \in [-1,1]} \prod_{b=0}^{n-1} (x - x_b)$$

  at 2^{1-n}. Thus, it achieves

$$\|f - \mathcal{P}_n\|_{L_\infty} \leq \frac{M}{n!} \cdot 2^{1-n}.$$

- We say that the interpolation points $x_0, ..., x_{n-1}$ are n Chebyshev nodes on $[-1, 1]$.

You may have noticed that the above results are about approximating functions with codomain $\mathbb{R}$, whereas we're trying to approximate a complex exponential $e^{ix/2^k}$. We can easily deal with this by generalizing the above results to functions with codomain $\mathbb{C}$, or alternatively by taking interpolants of $\cos(2/2^k)$ and $\sin(x/2^k)$.

It is important to note that while we use the interval $[-1, 1]$ as the domain over which we need to achieve accuracy, we can apply a linear function to our input x (as well as the interpolation points $x_0, ..., x_{n-1}$) to achieve accuracy on any small interval. In particular, we want our approximation P_n of e^{ix} to be accurate on $I = \left[\frac{-2\pi K}{2^k}, \frac{2\pi K}{2^k}\right]$ because we evaluate it on $\frac{x}{2^k}$ for $x \in [-2\pi K, 2\pi K]$. So, our interpolation points will actually be

$$x_b^* = \frac{2\pi K}{2^k} \cdot \cos\left(\frac{(b + \frac{1}{2})\pi}{n}\right) \in I.$$

Over any given interval, the degree-d Chebyshev interpolant gives a slightly better estimate of $\sin(x)$ than the degree-d Taylor series does. (The Chebyshev interpolant's error bound is better by a factor of $O(2^d)$, which sounds like a lot, but it's actually not that significant because each error bound has an $d!$ in the denominator.)

Better interpolation points

Recall that the interpolation points $x_0, ..., x_{n-1}$ used by Chebyshev interpolants is given by

$$x_b^* = \frac{2\pi K}{2^k} \cdot \cos\left(\frac{\left(b+\frac{1}{2}\right)\pi}{n}\right) \in I$$

is the choice which minimizes the quantity

$$\max_{x \in I} \prod_{b=0}^{n-1} (x - x_b),$$

motivated by the error bound

$$|f(x) - P_n(x)| \leq \frac{M}{n!} \prod_{b=0}^{n-1} (x - x_b).$$

We can choose better interpolation points by noticing that we don't accuracy on the entire interval I.

Taking a step back, recall that the goal of this step of bootstrapping is to approximately map $m_i' = m_i + q_0 t_i$ to m, where we can assume that m_i and t_i are small, say $|m_i| \leq \nu$ and $t_i \leq K$. We do this by using

$$m_i \approx \frac{q_0}{2\pi} \sin\left(\frac{2\pi m_i'}{q_0}\right),$$

so our approximation of $\sin(\cdot)$ must be accurate on an interval approximately $[-2\pi K, 2\pi K]$. Noting our "square k times" trick, our approximation of the complex exponential must be accurate on interval $I = \left[\frac{-2\pi K}{2^k}, \frac{2\pi K}{2^k}\right]$.

However, since we are assuming that $|m_i| \leq \nu$, we actually only need accuracy on a union of small intervals,

$$J = \bigcup_{a=-K}^{K} I_a = \bigcup_{a=-K}^{K} \left[\frac{2\pi\left(a - \frac{\nu}{q_0}\right)}{2^k}, \frac{2\pi\left(a + \frac{\nu}{q_0}\right)}{2^k}\right].$$

Let $\varepsilon = \frac{\nu}{q_0}$ so we can write $I_a = \left[\frac{2\pi(a-\varepsilon)}{2^k}, \frac{2\pi(a+\varepsilon)}{2^k}\right]$.

We now seek to choose $x_0, ..., x_{n-1}$ to minimize

$$\max_{x \in J} \prod_{b=0}^{n-1} (x - x_b) = \max_{a \in [-K,K]} \max_{x \in I_a} \prod_{b=0}^{n-1} (x - x_b).$$

Naturally, we want to pick points which lie in the intervals I_a of interest. In particular, we choose some $n_{-K}, n_{-K+1} ..., n_{K-1}, n_K$ with $n = n_{-K} + \cdots + n_K$, and put n_a interpolation points $x_{a,0}, ..., x_{a,n_a-1}$ in interval I_a according to

$$x_{a,b} = \frac{2\pi\left(a + \varepsilon \cdot \cos\left(\frac{\left(b+\frac{1}{2}\right)\pi}{n_a}\right)\right)}{2^k} \in I_a.$$

(These are n_a Chebyshev nodes on I_a.)

The choice of $n_{-K}, ..., n_K$ can be done by smoothing or greedy algorithm, but we won't discuss the details of this choice.

11. Bootstrapping sparse ciphertexts

12. Fourier transform for moving between coefficients and slots

13. Residue number systems

14. Improved key switching

15. Improved matrix multiplication