Improved Bootstrapping for Approximate Homomorphic Encryption

Hao Chen[1], Ilaria Chillotti[1], and Yongsoo Song[2]

[1] Microsoft Research
[2] University fo California, San Diego

October 28, 2018

Abstract. Since Cheon et al. introduced a homomorphic encryption scheme for approximate arithmetic (Asiacrypt '17), it has been recognized as suitable for important real-life usecases of homomorphic encryption, including training of machine learning models over encrypted data. A follow up work by Cheon et al. (Eurocrypt '18) described an approximate bootstrapping procedure for the scheme. In this work, we improve upon the previous bootstrapping result. We improve the amortized bootstrapping time per plaintext slot by two orders of magnitude, from $\sim$ 1 second to $\sim$ 0.01 second. To achieve this result, we adopt a smart level-collapsing technique for evaluating DFT-like linear transforms on a ciphertext. Also, we replace the Taylor approximation of the sine function with a more accurate and numerically stable Chebyshev approximation, and design a modified version of the Paterson-Stockmeyer algorithm for fast evaluation of Chebyshev polynomials over encrypted data.

Keywords. Fully homomorphic encryption, bootstrapping

1 Introduction

Homomorphic Encryption (HE) refers to a specific class of encryption schemes which allows computing directly on encrypted data without having to decrypt. Due to this special property, it has numerous potential applications in data-heavy industries, where one challenge is to gain meaningful insights from data while keeping the data itself private. Since the first construction of HE by Gentry [23], the field has witnessed a lot of growth: more efficient schemes (e.g. [8, 9, 7, 22, 20, 17, 18]) have been proposed, and there has been various design and implementations (e.g. [25, 4, 12, 10, 19, 6]) of confidential computing applications using HE.

However, the HE-based solutions have two issues when we apply them to applications which requires arithmetic on real numbers. The first issue is plaintext growth: the native plaintext in the HE schemes belong to a certain finite space. In order to encrypt a real number, one needs to first scale it up to an integer, so that the fractional part becomes the less significant digits. The size of this scaled integer will grow as we perform homomorphic multiplications on the ciphertext.

Since the plaintext space is finite, after a certain number of multiplications, it is impossible to recover the actual number. So, we need to perform "scale down and truncate" operation on encrypted data. However, it is an expensive operation since the available HE schemes only support addition and multiplication.

The second issue is ciphertext growth. Following Gentry's blueprint, a fresh encryption contains a small amount of "noise", and the noise level grows as operations are done on the ciphertext. It is necessary that the noise does not overwhelm the actual data within a ciphertext. To achieve this, one could use the Somewhat Homomorphic Encryption (SHE) approach, where the parameters are scaled up with the level of the circuit to be evaluated so that noise overflow is unlikely. Using SHE, both the ciphertext size and the performance overhead of HE will grow at least linearly with the circuit level, hence this approach has scaling issues. The other option is the Fully Homomorphic Encryption (FHE) approach, which uses Gentry's *bootstrapping* technique to refresh the noise in a ciphertext, so that circuit of arbitrary level can be evaluated on a fixed set of parameters. The FHE approach solves the ciphertext growth problem, with the caveat being that bootstrapping is expensive in practice despite the continuous effort in optimizations.

In 2017, Cheon et al. [16] proposed a HE scheme (denoted by CKKS scheme from now) which performs approximate arithmetic on encrypted data, by introducing a novel encoding technique and a fast "scaling down" operation, which effectively controls the growth of plaintext. Due to the nice properties, the CKKS scheme performs well at tasks such as training a logistic regression over encrypted data on a medium-sized data set with around 1000 samples [30]. Recently, a bootstrapping algorithm for the CKKS scheme was proposed in [14]. Using the bootstrapping procedure, one could train a logistic regression model over data sets with more than 4×10^5 samples in around 17 hours [29].

1.1 Previous Bootstrapping Method for CKKS

In the CKKS scheme, the cipehrtext modulus q decreases after each homomorphic multiplication, and decryption is correct if and only if the norm of the message is smaller than q. Hence, one can only perform a certain number of sequential multiplications before q gets too low for the next multiplication. Hence, bootstrapping amounts to the following function: giving a ciphertext ct with modulus q encrypted under the secret sk such that

$$[\langle \mathsf{ct}, \mathsf{sk} \rangle]_q = m,$$

bootstrapping outputs a ciphertext ct' in a larger modulus $Q > q$ such that

$$[\langle \mathsf{ct}', \mathsf{sk} \rangle]_Q \approx m,$$

Note that we do not hope to have exact equality, due to the approximate nature of CKKS.

Given this goal, the bootstrapping method in previous work [14] starts by the following observation: if ct is a ciphertext with modulus q and message $m(X)$,

then for a larger modulus $Q \gg q$, the same ciphertext decrypts to $t(X) = m(X) + q \cdot I(X)$ for a polynomial $I(X)$ with small coefficients. The next step approximately evaluates the modulo q function on coefficients to recover the coefficients $m_i = [t_i]_q$ of the input plaintext. It is done by first taking the d-th Taylor polynomial of the scaled exponential function $\exp(2\pi it/(2^r \cdot q))$, raising the polynomial to power 2^r through repeated squaring, and finally take the imaginary part and scale by $q/(2\pi)$. In other words, we have an approximation polynomial indexed by d and r:

$$K_{d,r}(t) = \frac{q}{2\pi}\left[\sum_{k=0}^{d}\frac{1}{k!}\left(\frac{2\pi it}{2^r \cdot q}\right)^k\right]^{2^r},$$

whose imaginary part approximates values of $(q/2\pi) \cdot \sin(2\pi t/q) \approx [t]_q$, as desired. One issue remaining is that homomorphic operations are not performed on coefficients but on plaintext slots. Before and after the evaluation of an exponential function, we have to shift the coefficients into the plaintext slots, and vice versa. It can be done by evaluating the encoding and decoding algorithms which are linear transformations on plaintext vectors.

Why the previous method does not scale well. There have remained some efficiency issues in the previous work. First, the parameters of $K_{d,r}(t)$ were chosen by $d = O(1)$ and $r = O(\log q)$ to guarantee the accuracy of approximation. It requires only $O(\log q)$ homomorphic operations to evaluate the exponential function, but the depth $O(\log q)$ is somewhat large. Meanwhile, the linear transformations require only one level, but their complexity grows linearly with the number of plaintext slots. As a result, the previous solution was not scalable when a ciphertext is densely-packed, and it was not optimal with respect to the level consumption.

1.2 Our Contributions

In this paper, we suggest two improvements upon the bootstrapping algorithm in [14].

Linear Transforms To improve the linear transform step, we first observed that the linear transforms involved in the bootstrapping process admit FFT-like algorithms, which requires more levels but less operations. Then, in order to fully explore the trade-off between level consumption and number of operations, we adopted an idea from Halevi and Shoup in [27], which uses a dynamic programming approach to decide the optimal level collapsing strategy for a generic multi-leveled linear transforms. As a result, our linear transforms are faster while being able to operate on 2-128x more slots, resulting in a large increase on the bootstrapping throughput.

Sine approximation Then, we used a Chebyshev interpolant to approximate the scaled sine function, which not only consumes less levels but also is more accurate than the original method. Our results indicate that in order to achieve the same level of approximation error, our method only requires $\max\{\log K + 2, \log\log q\}$ levels, whereas the previous solution requires $O(\log(Kq))$ levels. Here q is closely related to the plaintext size before bootstrapping, and $K = O(\lambda)$ is a small constant related to the security parameter.

In order to evaluate a Chebyshev interpolant of form $\sum_{k=0}^{n} c_k T_k(x)$ efficiently on encrypted inputs, we proposed a modified Paterson-Stockmeyer algorithm which works for polynomials represented in Chebyshev base. As a result, our approach requires $O(\sqrt{\max\{4K, \log q\}})$ ciphertext multiplications to evaluate the sine approximation, which is asymptotically better compared to $O(\log(Kq))$ in the previous work.

1.3 Related Works

There has been a few works which focus on improving the performance of bootstrapping. In terms of throughput, the works [24, 28, 11] designed optimized bootstrapping algorithms for BGV/BFV schemes. In terms of latency, the line of work [20, 17, 18] designed a specific RLWE-based HE scheme suitable for bootstrapping, and through extensive optimizations brought the bootstrapping time down to 13 ms. However, the scheme encrypts every bit separately, and bootstrapping needs to be performed after every single binary gate. Hence the overhead is still quite large for it to be practical in large scale applications.

Our major point of comparison is [14], bootstrapping for the CKKS approximate homomorphic encryption scheme. It is based on a novel idea of using a scaled sine function $\frac{1}{2\pi}\sin(2\pi t/q)$ to approximate the modulus reduction function $[t]_q$.

1.4 Road map

In Section 2, we recall the constructions and properties of the CKKS scheme and its bootstrapping algorithm. In Section 3, we describe our optimization of the linear transforms. In Section 4, we discuss our optimization of the sine evaluation step in CKKS bootstrapping using Chebyshev interpolants. We analyze our improved bootstrapping algorithm and present performance results in Section 5. Finally, we conclude in Section 6 with future research directions.

2 Background

2.1 The CKKS Scheme

We restate the CKKS scheme [16] below. For a power-of-two integer N, we denote $R = \mathbb{Z}[X]/(X^N + 1)$ be the ring of integers of the $(2N)$-th cyclotomic field. A single CKKS ciphertext can encrypt a complex vector with $\ell \leq (N/2)$

entries. To be precise, let $\zeta = \exp(\pi i/2\ell)$ be a (4ℓ)-th primitive root of unity for a power-of-two integer $1 \le \ell \le N/2$. The decoding algorithm takes as the input an element $m(Y)$ of the cyclotomic ring $\mathbb{R}[Y]/(Y^{2\ell}+1)$ and returns a vector $\mathsf{Decode}(m) = (m(\zeta), m(\zeta^5), \ldots, m(\zeta^{4\ell-3}))$. Note that Decode is a ring isomorphism between $\mathbb{R}[Y]/(Y^{2\ell}+1)$ and $\mathbb{C}^{\ell/2}$. If we identify $m(Y)$ with the vector $\boldsymbol{m} = (m_0, \ldots, m_{2\ell-1})$ of its coefficients, then the decoding algorithm can be viewed a linear transformation whose matrix representation is given by

$$M_\ell = \begin{bmatrix} 1 & \zeta & \zeta^2 & \cdots & \zeta^{2\ell-1} \\ 1 & \zeta^5 & \zeta^{5\cdot 2} & \cdots & \zeta^{5(2\ell-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta^{4\ell-3} & \zeta^{(4\ell-3)\cdot 5} & \cdots & \zeta^{(4\ell-3)(2\ell-1)} \end{bmatrix},$$

i.e., $\mathsf{Decode}(m) = M_\ell \cdot \boldsymbol{m}$. The encoding algorithm is defined by its inverse. When we implement the decoding function, we first define the *special Fourier transformation* matrix

$$\mathsf{SF}_\ell = \begin{bmatrix} 1 & \zeta & \cdots & \zeta^{\ell-1} \\ 1 & \zeta^5 & \cdots & \zeta^{5(\ell-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta^{4\ell-3} & \cdots & \zeta^{(4\ell-3)(\ell-1)} \end{bmatrix},$$

which is an $\ell \times \ell$ square matrix satisfying $M_\ell = [\mathsf{SF}_\ell | i \cdot \mathsf{SF}_\ell]$. Then, the decoding and encoding algorithms can be represented using the multiplication with SF_ℓ, its inverse and some conjugations.

We embed plaintext polynomials in $\mathbb{R}[Y]/(Y^{2\ell}+1)$ into $\mathbb{R}[X]/(X^N+1)$ by $Y \mapsto X^{N/2\ell}$. We say that a plaintext is fully packed (full-slot) when $\ell = N/2$. An encoded polynomial should be rounded to the closest integral polynomial in R to be encrypted.

• $\mathsf{Setup}(1^\lambda)$. Given the security parameter λ, choose a power-of-two integer N. Set the distributions $\chi_{\mathsf{key}}, \chi_{\mathsf{err}}, \chi_{\mathsf{enc}}$ on R for the secret, error, and encryption, respectively. For a base integer p and the number of levels L, set the chain of ciphertext moduli $q_\ell = p^\ell$ for $1 \le \ell \le L$. Choose an integer P.

• $\mathsf{Keygen}()$. Sample $s \leftarrow \chi_{\mathsf{key}}$ and set the secret key as $\mathsf{sk} \leftarrow (1, s)$. Sample $a \leftarrow U(R_{q_L})$ and $e \leftarrow \chi_{\mathsf{err}}$, and set the public key as $\mathsf{pk} \leftarrow (b, a) \in R_{q_L}^2$ for $b = -as + e \pmod{q_L}$. Sample $a' \leftarrow R_{P \cdot q_L}, e' \leftarrow \chi_{\mathsf{err}}$ and set evaluation key as $\mathsf{evk} \leftarrow (b', a') \in R_{P \cdot q_L}^2$ for $b' = -a's + e' + Ps^2 \pmod{P \cdot q_L}$.

• $\mathsf{Enc}_{\mathsf{pk}}(m)$. Sample $r \leftarrow \chi_{\mathsf{enc}}$ and $e_0, e_1 \leftarrow \chi_{\mathsf{err}}$. Output the ciphertext $\mathsf{ct} = r \cdot \mathsf{pk} + (m + e_0, e_1) \pmod{q_L}$. Note that $\langle \mathsf{ct}, \mathsf{sk} \rangle \pmod{q_L}$ is approximately equal to m.

• $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$. For an input ciphertext of level ℓ, compute and output $m = \langle \mathsf{ct}, \mathsf{sk} \rangle \pmod{q_\ell}$.

We remark that the encryption procedure of CKKS introduces an error so its decrypted value is not exactly same as the input value. We describe homomorphic operations (addition, multiplication, scalar multiplication, and rescaling) as follows.

• $\mathsf{Add}(\mathsf{ct}, \mathsf{ct}')$. For ciphertexts $\mathsf{ct}, \mathsf{ct}'$ in the same level ℓ, output $\mathsf{ct}_{\mathsf{add}} = \mathsf{ct} + \mathsf{ct}' \pmod{q_\ell}$.

• $\mathsf{CMult}_{\mathsf{evk}}(a, \mathsf{ct})$. For a constant $a \in R$ and a ciphertext ct of level ℓ, output $\mathsf{ct}_{\mathsf{cmult}} = (d_0, d_1) + \lfloor P^{-1} \cdot d_2 \cdot \mathsf{evk} \rceil \pmod{q_\ell}$.

• $\mathsf{Mult}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{ct}')$. For $\mathsf{ct} = (c_0, c_1), \mathsf{ct}' = (c_0', c_1') \in R_{q_\ell}^2$, let $(d_0, d_1, d_2) = (c_0c_0', c_0c_1' + c_0'c_1, c_1c_1') \pmod{q_\ell}$. Output $\mathsf{ct}_{\mathsf{mult}} = (d_0, d_1) + \lfloor P^{-1} \cdot d_2 \cdot \mathsf{evk} \rceil \pmod{q_\ell}$.

• $\mathsf{Rescale}_{\ell \to \ell'}(\mathsf{ct})$. For an input ciphertext of level ℓ, output $\mathsf{ct}' = \lfloor p^{\ell' - \ell} \cdot \mathsf{ct} \rceil \in \pmod{q_{\ell'}}$.

We note that $\{1, 5, \ldots, 2\ell - 3\}$ is a cyclic subgroup of the multiplicative group $\mathbb{Z}_{2\ell}^\times$ generated by the integer 5. One can rotate or take the conjugate of an encrypted plaintext by evaluating the maps $Y \mapsto Y^5$ or $Y \mapsto Y^{-1}$ based on the key-switching technique. The rotation key rk and conjugation key ck should be published to perform these algorithms (see [14] for details).

• $\mathsf{Rotate}_{\mathsf{rk}}(\mathsf{ct}; k)$. For an input encryption of $m(Y)$, return an encryption of $m(Y^{5^k})$ in the same level. The encrypted plaintext vector is shifted by k slots.

• $\mathsf{Conjugate}_{\mathsf{ck}}(\mathsf{ct})$. For an input encryption of $m(Y)$, return an encryption of $m(Y^{-1})$ in the same level. It takes the conjugation of the encrypted plaintext.

In applications of CKKS, we usually multiply a *scaling factor* to plaintexts to maintain the precision of computations. The rescaling algorithm can divide an encrypted plaintext by a power of p and preserve the size of scaling factors during homomorphic arithmetic.

2.2 Previous Bootstrapping for CKKS

Cheon et al. [14] showed how to refresh a ciphertext of the CKKS scheme. In this section, we briefly explain the previous solution.

Suppose that we have a *low-level* ciphertext $\mathsf{ct} \in R_q^2$ encrypting $m(Y) \in \mathbb{Z}[Y]/(Y^{2\ell}+1) \subseteq R$, i.e., $\langle \mathsf{ct}, \mathsf{sk} \rangle \pmod{q} \approx m(Y)$. Recall that $m(Y)$ can be identified with an ℓ-dimensional complex vector $\mathbf{z} = \mathsf{Decode}(m)$. The goal of bootstrapping is to generate a *high-level* ciphertext ct' satisfying $\langle \mathsf{ct}', \mathsf{sk} \rangle \pmod{Q} \approx m(Y)$ by evaluating the decryption circuit homomorphically.

The first step raises up the modulus of an input ciphertext. We have that $[\langle \mathsf{ct}, \mathsf{sk} \rangle]_{Q_0} \approx q \cdot I(X) + m(Y)$ for some $Q_0 > q$ and $I(X) \in R$. The coefficients of $I(X)$ is bounded by a constant K which depends on the secret distribution χ_{key}. Then, we perform the *subsum* procedure which generates a ciphertext ct' such that $\langle \mathsf{ct}', \mathsf{sk} \rangle \approx (N/2\ell) \cdot t(Y) \pmod{Q_0}$ for $J(Y) = I_0 + I_{N/2\ell} \cdot Y + \cdots +$

$I_{(2\ell-1)N/2\ell} \cdot Y^{N-1}$ and $t(Y) = q \cdot J(Y) + m(Y)$.[3] The constant $(N/2\ell)$ can be canceled by the rescaling process.

The *coefficients to slots* step, denoted by coeffToSlot, is to generate an encryption of the coefficients of $t(Y) = q \cdot J(Y) + m(Y)$, i.e., a ciphertext ct'' which satisfies that

$$[\langle \mathsf{ct}'', \mathsf{sk} \rangle]_{Q_1} \approx \mathsf{Encode}(t)$$

for some Q_1. This step can be done by homomorphically evaluating the encoding algorithm which is a variance of complex Fourier transformation. We point out that the resulting ciphertext should encrypt an (2ℓ)-dimensional vector $(t_0, \ldots, t_{2\ell-1})$ compared to the input ciphertext with ℓ plaintext slots, so we need to generate two ciphertexts encrypting halves of coefficients when the full-slot case $\ell = N/2$.

Now we have one or two ciphertexts which encrypt $t_i = q \cdot J_i + m_i$ for $0 \le i < 2\ell$ in their plaintext slots. The goal of next step (evalExp) is to homomorphically evaluate the reduction modulo q function and return ciphertexts encrypting $m_i = [t_i]_q$ in plaintext slots. Since the modulo reduction is not a polynomial function, the previous work used the following approximation by a trigonometric function which has a good accuracy under the condition that $|m| \ll q$:

$$[t]_q = m \approx \frac{q}{2\pi} \sin\left(\frac{2\pi t}{q}\right).$$

For the evaluation of this sine function, we first evaluate the polynomial

$$P_{-r}(t) = \sum_{k=0}^{d} \frac{1}{k!} \left(\frac{2\pi t}{2^r \cdot q}\right)^k \approx \exp\left(\frac{2\pi i t}{2^r \cdot q}\right)$$

for some integers r and d, which is the d-th Taylor polynomial of complex exponential function. Then, we can recursively perform the squaring r times $P_{i+1}(x) = P_i(x)^2$ to get an encryption of

$$P_0(t) = [P_{-r}(t)]^{2^r} \approx \exp(2\pi i t/q)$$

whose imaginary part is $\sin(2\pi i t/q)$ as desired. The output of evalExp is one or two ciphertexts which contains approximate values of $[t_i]_q = m_i$ in their plaintext slots.

During the evalExp step, one needs to multiply a scaling factor $\delta \cdot q$ to encrypted values for an appropriate constant δ to keep the precision of computation. A larger scaling factor will consume more ciphertext modulus while a small scaling factor makes the result less accurate. Table 1 summarizes the consumption of modulus bits and relative error from approximation based on the parameter Set-I which uses the initial polynomial of degree $d = 7$ and $r = 6$ iterations.

[3] The subsum algorithm can be understood as the evaluation of trace with respect to the field extension $\mathbb{Q}[X]/(X^N+1) \ge \mathbb{Q}[Y]/(Y^{2\ell}+1)$. It does nothing when $\ell = N/2$.

Params	$\log \delta$	Mod bit consumption	Relative error
Set-I	4	337	0.00083
	3	327	0.002
	2	317	0.003

Table 1. Comparison of different log T and log I values

Finally, the *slots to coefficients* (slotToCoeff) stage is exactly the inverse of coeffToSlot. It homomorphically evaluates the decoding algorithm to get a ciphertext such that $[\langle \mathsf{ct}''', \mathsf{sk} \rangle]_{Q_2} \approx m(Y)$ for some Q_2. We stress again that ct''' has ℓ plaintext slots, the same as the input ciphertext ct. The slotToCoeff step merges two output ciphertexts of evalExp and returns a fully packed ciphertext in the full-slot case. Otherwise, the number of plaintext slots is reduced from 2ℓ to ℓ during the evaluation.

3 Improved Linear Transforms from Level-Collapsing

In this section, we present a method to improve the performance of linear transformations coeffToSlot and slotToCoeff.

3.1 FFT-like Algorithms for coeffToSlot and slotToCoeff

The coeffToSlot and slotToCoeff steps in the original bootstrapping algorithm amounts to two linear transforms that are mutually inverses to each other. More precisely, slotToCoeff includes the computation $\mathbf{z} \mapsto \mathsf{SF}_\ell \cdot \mathbf{z}$ where SF_ℓ is the special Fourier transformation matrix defined in the previous section. Meanwhile, coeffToSlot is equivalent to computing the map SF_ℓ^{-1} on the plaintext vector. In order to evaluate these transforms on a ciphertext encrypting the vector $\mathbf{z}$, the previous work [14] adopted the diagonal method combined with a babystep-giantstep trick.

We begin by noting that similar to the Cooley-Tukey butterfly algorithm for DFT, the linear transform SF_ℓ can be expressed as a sequence of "butterfly" operations. The following algorithm is taken from the `HEAANBOOT` library [13].

In the beginning of Algorithm 1, a bit-reversal is performed, which effectively permutes the input vector. Then, the algorithm performs $\log \ell$ layers of transforms. Similarly, we can invert the above algorithm to obtain an FFT-like algorithm to compute SF_ℓ^{-1}, which starts with ℓ levels of transforms, followed by a bit-reversal.

3.2 Our Solution

First, we observe that for the purpose of bootstrapping, the bit-reversal operations are not necessary in the linear transforms. This is because bit-reversal is

Algorithm 1: FFT-like algorithm for evaluating SF_ℓ

Input: $\ell > 1$ a power of 2 integer; $\mathbf{z} \in \mathbb{C}^\ell$, and a precomputed table Ψ of complex 4ℓ-roots of unities $\Psi[j] = \exp(\pi i j/2\ell)$, $0 \le j < 4\ell$.

Output: $\mathbf{w} = \mathsf{SF}_\ell \cdot \mathbf{z}$

1 $\mathbf{w} = \mathbf{z}$
2 $\mathsf{bitReverse}(\mathbf{w}, \ell)$
3 **for** ($m = 2;\ m \le \ell;\ m = 2m$) {
4 **for** ($i = 0;\ i < \ell;\ i = i + m$) {
5 **for** ($j = 0;\ j < m/2;\ j = j + 1$) {
6 $k = (5^j \mod 4m) \cdot \ell/m$
7 $U = \mathbf{w}[i + j]$
8 $V = \mathbf{w}[i + j + m/2]$
9 $V = V \cdot \Psi[k]$
10 $\mathbf{w}[i + j] = U + V$
11 $\mathbf{w}[i + j + m/2] = U - V$
12 }
13 }
14 }
15 **return w**

a permutation of order 2, and the sine evaluation is a SIMD (single instruction multiple data) operation, i.e., the same operation is performed independently on each slot. Hence, bit-reversals right before and after the sine evaluation will cancel themselves out. Therefore, we only need to perform the butterfly transforms homomorphically. For ease of notations, we still use SF_ℓ to denote the linear transform in lines 3-15 of Algorithm 1.

Next, we note that each layer of Algorithm 1 can be implemented using two slot rotations and three SIMD plaintext multiplications. More precisely, the i-th iteration in Algorithm 1 can be represented as

$$\mathbf{w} := \mathbf{a}[i] \odot \mathbf{w} + \mathbf{b}[i] \odot (\mathbf{w} \ll 2^{i-1}) + \mathbf{c}[i] \odot (\mathbf{w} \gg 2^{i-1}),$$

where $\mathbf{w} \ll j$ (resp. $\mathbf{w} \gg j$) denotes rotating the vector $\mathbf{w}$ to the left (resp. right) by j slots, and $\odot$ denotes the component-wise multiplication between vectors. The vectors $\mathbf{a}[i], \mathbf{b}[i], \mathbf{c}[i] \in \mathbb{C}^\ell$ can be precomputed. This gives us a direct algorithm to evaluate linear transform SF_ℓ on an encrypted vector in CKKS scheme using $\log \ell$ levels and $O(\log \ell)$ operations. In contrast, the approach in [14] requires one level and $O(\ell)$ operations to evaluate SF_ℓ.

In practice, a hybrid approach might work better than the above two extremes. For example, we can trade operations for levels by "collapsing" some levels in the above algorithm. We will elaborate on this method below.

3.3 Optimal Level-Collapsing from Dynamical Programming

First we recall the idea of Halevi and Shoup [27]. The task is to apply a sequence of linear transforms $L_1 \circ \cdots \circ L_\ell$ on some input, and each evaluation consumes one "level". One is allowed to collapse some levels by merging some adjacent transforms into one. For example, for $n = 4$ we could merge into two levels by letting $M_1 = L_1 \circ L_2$ and $M_2 = L_3 \circ L_4$. Assuming there is a cost function associated to every linear transform, it is an optimization problem to find the best level collapsing strategy that minimizes the cost. More precisely, let $\mathsf{Cost}(a, b)$ denote the cost of evaluating $L_a \circ \cdots \circ L_{b-1}$ and let $\ell' \leq \ell$ be an upper bound on the level. Then we wish to solve the following optimization problem:

$$\min_{\substack{a_0=1<a_1<\ldots<a_k<a_{k+1}=\ell+1,\\ k+1\leq\ell'}} \sum_{i=0}^{k} \mathsf{Cost}(a_{i-1}, a_i).$$

To solve for an optimal solution, we recall the idea outlined in [27] as follows. Let $\mathsf{Opt}(d, \ell')$ be the optimal cost to evaluate the first d linear transforms using ℓ' levels. Then

$$\mathsf{Opt}(d, \ell') = \min_{1\leq d'\leq d} \mathsf{Cost}(d - d', d + 1) + \mathsf{Opt}(d - d', \ell' - 1).$$

We can then use a dynamic programming algorithm to compute the optimal strategy as a list of splitting points $(a_1, \ldots, a_k)$. Given this optimal level collapsing strategy, we can generate the collapsed levels by merging the individual layers.

Applying level-collapsing to our case First, we give an example of how levels can be merged. Recall that the i-th level of Algorithm 1

$$\mathbf{w} := \mathbf{a}[i] \odot \mathbf{w} + \mathbf{b}[i] \odot (\mathbf{w} \ll 2^{i-1}) + \mathbf{c}[i] \odot (\mathbf{w} \gg 2^{i-1}).$$

Suppose we merge the layers i and $i + 1$. Then the new linear transform is

$$\begin{aligned}
\mathbf{w} := \; & \mathbf{a}[i+1] \odot \big(\mathbf{a}[i] \odot \mathbf{w} + \mathbf{b}[i] \odot (\mathbf{w} \ll 2^{i-1}) + \mathbf{c}[i] \odot (\mathbf{w} \gg 2^{i-1})\big) \\
& + \mathbf{b}[i+1] \odot \big(\mathbf{a}[i] \odot \mathbf{w} + \mathbf{b}[i] \odot (\mathbf{w} \ll 2^{i-1}) + \mathbf{c}[i] \odot (\mathbf{w} \gg 2^{i-1}))\big) \ll 2^i \\
& + \mathbf{c}[i+1] \odot \big(\mathbf{a}[i] \odot \mathbf{w} + \mathbf{b}[i] \odot (\mathbf{w} \ll 2^{i-1}) + \mathbf{c}[i] \odot (\mathbf{w} \gg 2^{i-1})\big) \gg 2^i \\
= \; & A \odot \mathbf{w} + B \odot (\mathbf{w} \ll 2^{i-1}) + C \odot (\mathbf{w} \gg 2^{i-1}) + D \odot (\mathbf{w} \ll 2^i) \\
& + E \odot (\mathbf{w} \gg 2^i) + F \odot (\mathbf{w} \ll 3 \cdot 2^{i-1}) + G \odot (\mathbf{w} \gg 3 \cdot 2^{i-1})
\end{aligned}$$

for some vectors $A, B, \ldots, G$. Overall, this merged layer requires 6 rotations and 7 plaintext multiplications. In general, if we merge some layers together, then we end up with a merged layer which looks like

$$\mathbf{w} := \sum_{i=1}^{k} \mathbf{p}[i] \odot (\mathbf{w} \ll t_i)$$

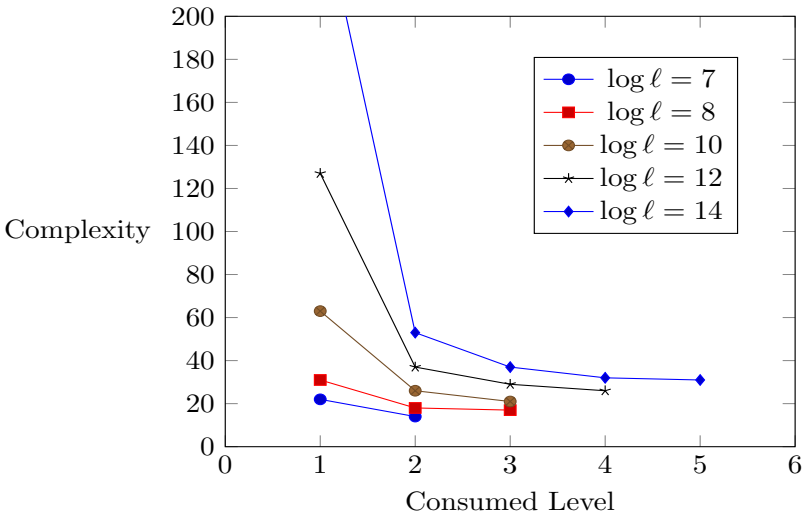

Fig. 1. Optimal complexity (number of rotations) of FFT-like algorithm with respect to the depth and number of slots

for some precomputable vectors $\mathbf{p}[i]$ and integers t_i, and requires $(k-1)$ rotations and k plaintext multiplications to evaluate. To further reduce the complexity, we can utilize a babystep-giantstep method to reduce the number of rotations to about $2\sqrt{k}$. Note that in a new version of the implementation of the CKKS scheme [1], plaintext multiplication takes much more time than rotation. Therefore, we define the cost of the merged layer as $2\sqrt{k}$. In the following Figure 1, we present the optimal costs for different ℓ and level upper bounds.

4 Improved Sine Evaluation from Chebyshev Approximations

4.1 Background: Chebyshev Polynomials and Chebyshev Interpolants

Recall that the *Chebyshev polynomials* is a family of polynomials $\{T_n(x)\}_{n\geq 0}$ defined by the recurrence relation:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{2n}(x) &= 2T_n(x)^2 - 1 \\ T_{2n+1}(x) &= 2T_n(x) \cdot T_{n+1}(x) - x. \end{aligned} \tag{1}$$

Given a Lipschitz continuous function f defined on the interval $[-1, 1]$, the n-th *Chebyshev interpolant* of f is defined as

$$p_n^{cheb}(x) = \sum_{k=0}^{n} c_k T_k(x)$$

where the coefficients c_k are uniquely determined such that $p_n^{cheb}(x_j) = f(x_j)$ for

$$x_j = \cos(j\pi/n), \text{ for } 0 \leq j \leq n.$$

Let p_n^* denote the *minimax* polynomial of degree $\leq n$ which minimizes the infinity norm $\|f - p_n^*\|_\infty$. It would be optimal to use p_n^* as a polynomial approximation to f. However, computing such polynomials is not trivial in practice. On the other hand, Chebyshev interpolants are not only easy to compute, but also almost as good as the minimax approximation. More precisely, we have the following formula from [21]:

$$\|f - p_n^{cheb}\|_\infty \leq \left(\frac{2}{\pi}\log n + 2\right) \cdot \|f - p_n^*\|_\infty. \tag{2}$$

4.2 Chebyshev Interpolants of Sine Function

Recall that in the bootstrapping procedure, we need to homomorphically evaluate

$$\frac{q}{2\pi}\sin\left(\frac{2\pi t}{q}\right).$$

with $t \in [-Kq, Kq]$. After a change of variables, we see that it suffices to evaluate

$$g(x) := \frac{1}{2\pi}\sin(2\pi K x)$$

with $x \in [-1, 1]$. Our goal is to find a polynomial $p(x)$ with small degree such that $\|g - p\|_\infty$ is small. How good can the approximation be? For the scaled sine function g, it has been shown (see e.g. [26]) that the minimax error $\epsilon_n = \|g - p_n^*\|_\infty$ satisfies

$$\limsup_{n\to\infty} n\epsilon_n^{1/n} = \frac{eK}{2}. \tag{3}$$

Therefore, ϵ_n decreases like $\left(\frac{eK}{2n}\right)^n$ as $n \to \infty$, i.e., the approximation error decreases super-exponentially as a function of the degree n. So, the $\log n$ loss factor from replacing the minimax approximation with Chebyshev interpolant is almost negligible compared to the decreasing speed of ϵ_n. Hence, Chebyshev interpolants provide a decent approximation the sine function in our bootstrapping algorithm.

We compare the Chebyshev interpolant approach with the approach in [14]. Recall that [14] first uses a Taylor polynomial of $\exp(2\pi i K x/2^r)$ of degree d

to approximate it. Then, it performs r repeated squaring operations to obtain an approximation of $\exp(2\pi iKx)$. Finally, $g(x)$ is equal to $1/(2\pi)$ times the imaginary part of $\exp(2\pi iKx)$. In Figure 2 below, we present the log-log plot of approximation error versus polynomial degree for different values of d.

From the plot, we see that the Chebyshev interpolant achieves small error quickly for degree less than 128. On the other hand, the [14] apporoach requires a much larger degree to reach the same error when $d = 7$. For a larger $d = 55$, the difference between the approaches becomes smaller. However, since the Taylor coefficients of $\exp(2\pi Kix/2^r)$ decrease super-exponentially, evaluating such a large degree Taylor approximation is likely to result in large numerical errors. Therefore, we decided to use Chebyshev interpolants for approximating the sine function.

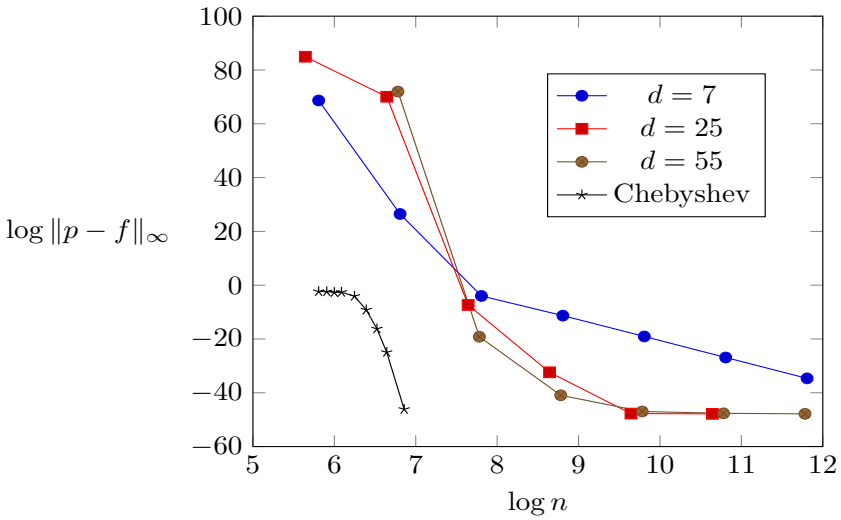

Fig. 2. Polynomial approximation errors to $\frac{1}{2\pi}\sin(2\pi Kx)$ $(K = 12)$.

4.3 Computing Chebyshev Polynomials in FHE

The Chebyshev coefficients c_k of the scaled sine function g can be precomputed and stored. Next, our task is to evaluate $\sum_{k=0}^{n} c_k T_k(x)$ homomorphically. There are several choices:

Since each $T_k(x)$ is a polynomial in x, we could rewrite $p_n^{cheb}(x)$ as $\sum_{k=0}^{n} c'_k x^k$, and use any existing method for homomorphic evaluation of polynomials in one variable in the literature. However, the transition matrix between the c_k and c'_k coefficients is ill-conditioned (actually, its conditional number grows exponentially as a function of n (see e.g. [3]), and the coefficients c'_k differ by many

orders of magnitude. Therefore, the evaluation is likely to generate large numerical errors, even over unencrypted input.

A better method is to use the recurrence relation (1) to evaluate $T_k(x)$ for $0 \leq k \leq n$, and then compute $\sum c_k T_k(x)$ using scalar multiplications and additions. This method yields smaller numerical errors in practice. However, the efficiency is sub-optimal: we still need $O(n)$ homomorphic multiplications in order to evaluate a degree-n Chebyshev interpolant.

Paterson-Stockmeyer for Chebyshev Our next idea is to use the Paterson-Stockmeyer algorithm [31], which requires only $O(\sqrt{n})$ non-scalar multiplications to evaluate a polynomial of degree n in x. However, we could not directly apply this algorithm since it requires the polynomial to be presented in the *power base* $1, x, \ldots, x^n$. Of course, one could rewrite the Chebyshev interpolant in power base first, and then execute the Paterson-Stockmeyer algorithm. But as we discussed above, such method is subject to large numerical errors, hence it is not a desirable solution.

Instead, we propose a new approach by modifying the Paterson-Stockmeyer algorithm to directly evaluate Chebyshev interpolants. As a result, we can evaluate a Chebyshev interpolant of degree n with $\sqrt{2n} + O(\log n)$ non-scalar multiplications. In order to describe our algorithm, we first recall the Paterson-Stockmeyer algorithm in [31]:

Algorithm 2: The original Paterson-Stockmeyer algorithm

Input: $(a_0, a_1, \ldots, a_n), u$
Output: $f(u) = \sum_i a_i u^i$

1 Find positive integers k, m such that $k \approx \sqrt{n/2}$ and $k(2^m - 1) > n$
2 $\tilde{f}(x) = f(x) + x^{k(2^m-1)}$
3 Compute powers $\mathbf{bs} = (u, u^2, u^k)$, $\mathbf{gs} = (u^k, u^{2k}, u^{4k}, \ldots, u^{2^{m-1}k})$
4 Using long division, write $\tilde{f}(x) = x^{k2^{m-1}} q(x) + r(x)$
5 Set $\tilde{r}(x) = r(x) - x^{k(2^{m-1}-1)}$
6 Using long division, write $\tilde{r}(x) = c(x)q(x) + s(x)$
7 Evaluate $c(x)$ at u using the precomputed powers
8 Set $\tilde{s}(x) = s(x) + x^{k(2^{m-1}-1)}$
9 Recursively evaluate $q(x)$ and $\tilde{s}(x)$ at u (with lines 1-3 skipped)
10 Compute $\tilde{f}(u) = (u^{k(2^m-1)} + c(u))q(u) + \tilde{s}(u)$.
11 Compute $u^{k(2^m-1)}$ by multiplying all values in **gs**
12 return $f(u) = \tilde{f}(u) - u^{k(2^m-1)}$

Now suppose we wish to use the Chebyshev basis $\{T_k(x)\}_k$ instead of the power base in Algorithm 2. We can start by replacing every occurrence of x^i in the algorithm with $T_i(x)$. Line 3 requires computing certain $T_i(x)$ values,

which can be done in $k+m$ operations using the recurrence formula (1). Thus we only need an algorithm for long division of polynomials in Chebyshev base. That is, given Chebyshev coefficients of polynomials f and g, output Chebyshev coefficients of the quotient and remainder polynomials q and r such that $\deg q = \deg f - \deg g$, $\deg r < \deg g$ and $f = qg + r$. A first attempt is to convert f and g to the power base, perform long division as usual, and convert the resulting q and r back to Chebyshev base. Again, this approach is likely to generate a lot of numerical errors since the transform matrices are ill-conditioned. To resolve this issue, we present a direct algorithm.

Long Division for polynomials in Chebyshev base

Lemma 1 (Long Division). *Given two polynomials f and g with positive degrees n and k given by their Chebyshev coefficients, there exists an algorithm with $O(k(n-k))$ operations to compute the Chebyshev coefficients of polynomials $q(x)$ and $r(x)$, such that $\deg q = \deg f - \deg g$, $\deg r < \deg g$, and*

$$f(x) = g(x)q(x) + r(x).$$

Proof. For simplicity, we assume both f and g are monic, meaning their highest Chebyshev coefficient is 1. We do it with induction on $n = \deg f$. If $n \leq \deg g$ then we are done. Now suppose $n > k = \deg g$ and $k \geq 1$. Let

$$r_0(x) = T_n(x) - 2g(x)T_{n-k}(x).$$

Using the formula

$$T_m(x) = 2T_i(x)T_{m-i}(x) - T_{|m-2i|}(x),$$

we see that $\deg(r_0) < n$, and we may compute the Chebyshev coefficients of $r_0(x)$. Now we could recursively perform the division r_0 by g to finish the algorithm. The correctness is easy to verify, and since computing r_0 requires $O(k)$ operations, the algorithm requires $O(k(n-k))$ operations. This finishes the proof.

Given the above lemma, we can modify Algorithm 2 to directly perform long division of polynomials in Chebyshev base. We omit the detailed description of the modified algorithm since it is straightforward. As a result, we have

Theorem 1. *There exists an algorithm to evaluate a polynomial of degree n given in Chebyshev base with $\sqrt{2n} + O(\log n)$ non-scalar multiplications and $O(n)$ scalar multiplications.*

5 Putting it together

5.1 Asymptotic analysis

Combining the optimizations in Section 3 and 4, we come up with a new bootstrapping algorithm for the CKKS scheme, whose complexity improves upon the algorithm in [14]. We make a detailed comparison below:

Linear Transforms The subSum step remains unchanged from [14], which requires $O(N/2\ell)$ rotations. For the two transforms coeffToSlot and slotToCoeff, recall that [14] takes $O(\sqrt{\ell})$ rotations and ℓ plaintext multiplications, whereas our algorithm provides a spectrum of trade-offs between level consumption and operation counts. For example, if we fix the level budget to be $\ell' = 2$, then both the coeffToSlot and slotToCoeff requires $O(\ell^{1/4})$ rotations and $O(\sqrt{\ell})$ plaintext multiplications.

Sine evaluation The approach of [14] to evaluate the sine approximation requires a polynomial of degree $d \cdot 2^r$ and $O(d+r)$ ciphertext multiplications. They took $d = O(1)$ and $r = O(\log(Kq))$ in order to achieve an approximation error of $O(1)$ for the function $(q/2\pi)\sin(2\pi t/q)$. Thus, both the required level and the number of operations are $O(\log(Kq))$.

In our case, we used a Chebyshev interpolant to approximate the sine function. From the results in Section 4, we see that it suffices to take degree $n \leq \max\{4K, \log q\}$ to achieve $1/q$ approximation error from (2) and (3). Therefore, our approach consumes only $\log n \leq \max\{\log K + 2, \log\log q\}$ levels. In terms of the number of operations, by using the modified Paterson-Stockmeyer algorithm, we can evaluate the Chebyshev interpolant in $O(\sqrt{n})$ ciphertext multiplications.

5.2 Implementation and performance

Recently, the authors of [16] published a improved version [1] of the implementation of the CKKS scheme with faster operations. We implemented our bootstrapping algorithm on top of the new version. In order to separate the causes of speedups, we also experimented with the original bootstrapping algorithm with the new library. We summarize our findings in Table 4.

Parameter Choices To benchmark the original bootstrapping algorithm, we used the same parameter sets (Table 2) from [14]. We modified these parameters slightly for our new bootstrapping algorithm. The modified parameters are presented in Table 3. We note that these modifications do not involve $\log N, \log Q$ or the initial noise in the ciphertexts, hence the security level remains the same as previous work.

Parameter	$\log N$	$\log Q_0$	$\log p$	$\log q$	r
Set-I	15	620	23	29	6
Set-II			27	37	7
Set-III	16	1240	31	41	7
Set-IV			39	54	9

Table 2. Parameter sets

Parameter	$\log p$	$\log q$	l_{ctos}	l_{stoc}
Set-I*	25	29	2	2
Set-II*	25	34	2	2
Set-II**	27	37	2	1
Set-III*	33	41	2	2
Set-III**	35	41	3	3
Set-IV*	43	54	3	3
Set-IV**	43	54	4	4

Table 3. New Parameter sets

In Table 3, the columns labeled l_{ctos} and l_{stoc} denote the level consumption for coeffToSlot and slotToCoeff, respectively. Note that larger levels result in less operations. For the sine evaluation, we fixed $K = 12$ and a Chebyshev interpolant of degree $n = 119$ based on experimental results. All experiments are performed on a laptop with 2.8GHz Intel Core i7 Processor and 16GB memory, running on a single thread.

5.3 Comparison

In order to make a meaningful comparison of the efficiency of the different bootstrapping methods/implementations, we need to provide a common measure, and one such measure is the number of slots times the number of levels allowed after bootstrapping, divided by the bootstrapping time. We argue that this definition makes sense, since in the process of evaluating a typical circuit homomorphically, the frequency of bootstrapping should be inverse proportional to the after level. Also, since the complexity of bootstrapping depends on the bit precision of the output, we plot the utility versus precision in the following Figure 3.

From Figure 3, we see that our new algorithms can improve the utility of bootstrapping by two orders of magnitude. For example, [14] could bootstrap numbers with around 20 bits of precision with a utility of 2.94 (Level $\times$ Slot / Second). With a slightly larger precision, we achieved a utility of 150, yielding a 50x improvement.

6 Conclusion and Future Work

In this work, we showed that algorithmic improvements to the linear transforms and sine evaluation steps could boost the efficiency of bootstrapping for the CKKS approximate homomorphic encryption scheme by two orders of magnitude.

Our results suggest that using Chebyshev interpolant together with the Paterson-Stockmeyer algorithm is a promising solution for approximately evaluating non-polynomial functions in FHE. For example, we could apply this idea

Params	logSlots	Method	LT	Sine Eval	Total Time (s)	Amortized Time (s)	Average Precision	After Level
Set-I	7	[13]	139.2	12.3	151.5	1.2	7.64	8
	7	[13] + [1]	36.1	5.26	41.36	0.32	7.64	8
Set-I*	10	This work	28.78	9.55	38.33	0.04	6.92	5
Set-II	7	[13]	127.3	12.5	139.8	1.1	9.9	1
	7	[13] + [1]	43.9	8.73	52.63	0.41	9.9	1
Set-II*	8	This work	16.87	9.18	26.05	0.04	10.03	2
Set-II**	10	This work	37.11	9.18	85.83	0.08	9.1	1
Set-III	7	[13]	528	63	591	4.6	13.2	19
	7	[13] + [1]	158.2	29.3	187.5	1.46	13.2	19
Set-III*	10	This work	154.28	47.7	201.98	0.2	13.7	17
Set-III**	12	This work	134.35	43.7	178.05	0.04	11.75	13
Set-IV	7	[13]	456	68	524	4.1	20.1	7
	7	[13] + [1]	224.2	80.7	304.9	2.38	20.1	7
Set-IV*	12	This work	127.49	40.38	167.87	0.04	20.86	6
Set-IV**	14	This work	119.76	38.56	158.32	0.01	18.63	3

Table 4. Performance comparisons for bootstrapping: LT (linear transformations) timing is the sum of the timings for subSum, coeffToSlot and slotToCoeff. Precision is averaged among all slots.

to evaluate the sigmoid function or the RELU function, which is interesting from the point of view of doing machine learning over encrypted data. Also, this idea can be applied to the absolute value function, which may expedite evaluation of a sorting network over encrypted data.

The improved linear transform technique for the CKKS scheme can be used to provide a fast evaluation of discrete Fourier transform (DFT) over encrypted data, which might be of independent interest. Also, we could utilize our algorithm to provide an efficient implementation of the conversion between CKKS ciphertexts and ciphertexts from TFHE or BFV/BGV schemes, outlined in a recent work [5].

Recently, there is another variant of the CKKS scheme [15] based on the *Residue Number System* (RNS), following an idea of Bajard et al. [2]. The reported performance numbers of this new variant are up to 10x better than the original implementation. Thus, it would be interesting to implement our bootstrapping algorithm on this RNS variant to obtain even better performance.

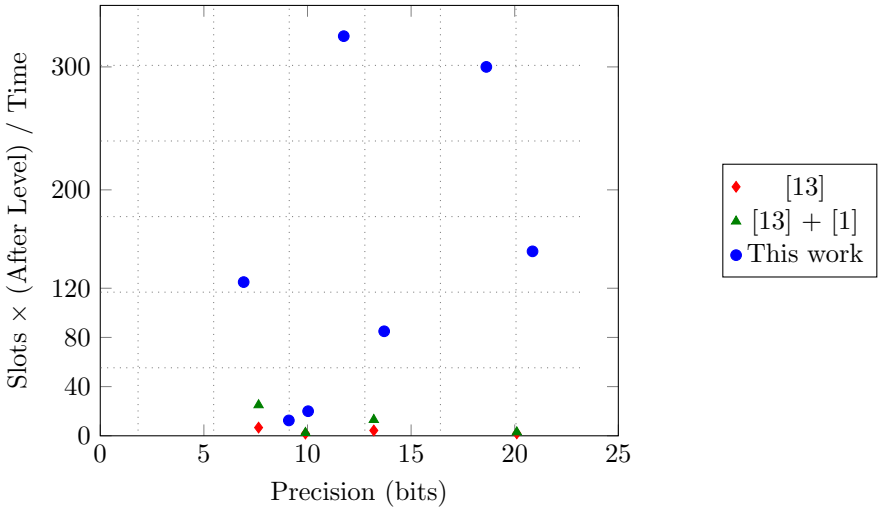

Fig. 3. Bootstrapping utility comparisons

References

1. HEAAN with Faster Multiplication. `https://github.com/snucrypto/HEAAN/releases/tag/2.1`, 2018.
2. J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *International Conference on Selected Areas in Cryptography*, pages 423–442. Springer, 2016.
3. B. Beckermann. *On the numerical condition of polynomial bases: estimates for the condition number of Vandermonde, Krylov and Hankel matrices*. PhD thesis, Verlag nicht ermittelbar, 1997.
4. C. Bonte, C. Bootland, J. W. Bos, W. Castryck, I. Iliashenko, and F. Vercauteren. Faster homomorphic function evaluation using non-integral base encoding. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 579–600. Springer, 2017.
5. C. Boura, N. Gama, and M. Georgieva. Chimera: a unified framework for b/fv, tfhe and heaan fully homomorphic encryption and predictions for deep learning. Cryptology ePrint Archive, Report 2018/758, 2018. `https://eprint.iacr.org/2018/758`.
6. F. Bourse, M. Minelli, M. Minihold, and P. Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, pages 483–512. Springer, 2018.
7. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proc. of ITCS*, pages 309–325. ACM, 2012.
8. Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS'11, pages 97–106. IEEE Computer Society, 2011.

9. Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In *Advances in Cryptology–CRYPTO 2011*, pages 505–524. Springer, 2011.
10. H. Chen, R. Gilad-Bachrach, K. Han, Z. Huang, A. Jalali, K. Laine, and K. Lauter. Logistic regression over encrypted data from fully homomorphic encryption. Cryptology ePrint Archive, Report 2018/462, 2018. `https://eprint.iacr.org/2018/462`.
11. H. Chen and K. Han. Homomorphic lower digits removal and improved fhe bootstrapping. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 315–337. Springer, 2018.
12. H. Chen, K. Laine, and P. Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1243–1255. ACM, 2017.
13. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Implementation of boostrapping for HEAAN, 2017. `https://github.com/kimandrik/HEAANBOOT`.
14. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In *Advanced in Cryptology–EUROCRYPT 2018*, pages 360–384. Springer, 2018.
15. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. A full rns variant of approximate homomorphic encryption. Cryptology ePrint Archive, Report 2018/931, 2018. `https://eprint.iacr.org/2018/931`.
16. J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017*, pages 409–437. Springer, 2017.
17. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–33. Springer, 2016.
18. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Application of Cryptology and Information Security*, pages 377–408. Springer, 2017.
19. J. L. Crawford, C. Gentry, S. Halevi, D. Platt, and V. Shoup. Doing real work with fhe: The case of logistic regression. Cryptology ePrint Archive, Report 2018/202, 2018. `https://eprint.iacr.org/2018/202`.
20. L. Ducas and D. Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology–EUROCRYPT 2015*, pages 617–640. Springer, 2015.
21. H. Ehlich and K. Zeller. Auswertung der normen von interpolationsoperatoren. *Mathematische Annalen*, 164(2):105–112, 1966.
22. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
23. C. Gentry. Fully homomorphic encryption using ideal lattices. In *In Proc. STOC*, pages 169–178, 2009.
24. C. Gentry, S. Halevi, and N. P. Smart. Better bootstrapping in fully homomorphic encryption. In *Public Key Cryptography–PKC 2012*, pages 1–16. Springer, 2012.
25. R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

26. A. Giroux. Approximation of entire functions over bounded domains. *Journal of Approximation Theory*, 28(1):45–53, 1980.
27. S. Halevi and V. Shoup. Algorithms in HElib. In *Advances in Cryptology–CRYPTO 2014*, pages 554–571. Springer, 2014.
28. S. Halevi and V. Shoup. Bootstrapping for HElib. In *Advances in Cryptology–EUROCRYPT 2015*, pages 641–670. Springer, 2015.
29. K. Han, S. Hong, J. H. Cheon, and D. Park. Efficient logistic regression on large encrypted data. Cryptology ePrint Archive, Report 2018/662, 2018. `https://eprint.iacr.org/2018/662`.
30. A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon. Logistic regression model training based on the approximate homomorphic encryption. Cryptology ePrint Archive, Report 2018/254, 2018. `https://eprint.iacr.org/2018/254`.
31. M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2(1):60–66, 1973.