Homomorphic Lower Digits Removal and Improved FHE Bootstrapping

Hao Chen[1] and Kyoohyung Han[2]

[1] Microsoft Research, USA, haoche@microsoft.com
[2] Seoul National University, Korea, satanigh@snu.ac.kr

Abstract. Bootstrapping is a crucial operation in Gentry's breakthrough work on fully homomorphic encryption (FHE), where a homomorphic encryption scheme evaluates its own decryption algorithm. There has been a couple of implementations of bootstrapping, among which HElib arguably marks the state-of-the-art in terms of throughput, ciphertext/message size ratio and support for large plaintext moduli.
In this work, we apply a family of "lowest digit removal" polynomials to improve homomorphic digit extraction algorithm which is crucial part in bootstrapping for both FV and BGV schemes. If the secret key has 1-norm $h = l_1(s)$ and the plaintext modulus is $t = p^r$, we achieved bootstrapping depth $\log h + \log(\log_p(ht))$ in FV scheme. In case of the BGV scheme, we bring down the depth from $\log h + 2\log t$ to $\log h + \log t$. We implemented bootstrapping for FV in the SEAL library. Besides the regular mode, we introduce another "slim mode", which restrict the plaintexts to batched vectors in $\mathbb{Z}_{p^r}$. The slim mode has similar throughput as the regular mode, while each individual run is much faster and uses much smaller memory. For example, bootstrapping takes 6.75 seconds for 7 bit plaintext space with 64 slots and 1381 seconds for $GF(257^{128})$ plaintext space with 128 slots. We also implemented our improved digit extraction procedure for the BGV scheme in HElib.

Keywords: Homomorphic Encryption, Bootstrapping, Implementation

1 Introduction

Fully Homomorphic Encryption (FHE) allows an untrusted party to evaluate arbitrary functions on encrypted data, without knowing the secret key. Gentry introduced the first FHE scheme in the breakthrough work [19]. Since then, there has been a large collection of work (e.g., [6–10,13,16,18,21,30]), introducing more efficient schemes.

These schemes all follow Gentry's original blueprint, where each ciphertext is associated with a certain amount of "noise", and the noise grows as homomorphic evaluations are performed. When the noise is too large, decryption will fail to give the correct result. Therefore, if no bootstrapping is performed, one set of HE parameters can only evaluate circuits of a bounded depth. This approach is called leveled homomomorphic encryption (LHE) and is used in a many works.

If FHE is desired, then we need to refresh the noise in a ciphertext. This can be done via Gentry's brilliant *bootstrapping* technique. Roughly speaking, bootstrapping a ciphertext in some given scheme means running its own decryption algorithm homomorphically, using an encryption of the secret key.

Bootstrapping is a very expensive operation. First, the decryption circuits of a scheme can be complex. Moreover, the decryption circuit of a scheme may not be conveniently supported by the scheme itself. Hence, in order to perform bootstrapping, one either needs to make significant optimizations in order to bring down its cost, or design some scheme who can handle its decryption circuit more comfortably. Among the best works on bootstrapping implementations, the work of Halevi and Shoup [24], which optimized and implemented bootstrapping over the scheme of Brakerski, Gentry and Vaikuntanathan (BGV), is arguably still the state-of-the-art in terms of throughput, ciphertext/message size ratio and flexible plaintext moduli. For example, they were able to bootstrapp a vector of size 1024 over $GF(2^{16})$ within about 5 minutes. However, when the plaintext modulus reaches 2^8, bootstrapping still takes a few hours to perform. The reason is mainly due to a high-cost digit extraction procedure, whose cost goes up significantly with the plaintext modulus. The Fan-Vercauteran (FV) scheme, a scale-invariant variant of BGV, has also been implemented in [1, 26] and used in applications. We are not aware of any previous implementation of bootstrapping in FV.

1.1 Contributions

In this paper, we aim at improving the efficiency of bootstrapping under large prime power plaintext moduli.

- We used a low degree lowest-digit-removal polynomial to design an improved algorithm to remove r lowest base-p digits from integers modulo p^e. Our new algorithm has depth $r \log p + \log(e)$, compared to $(e-1)\log p$ in previous work.
- We then applied our algorithm to improve the digit extraction step in the bootstrapping procedure for FV and BGV schemes. Let $h = l_1(s)$ denote the 1-norm of the secret key, and the plaintext space is a prime power $t = p^r$. Then for FV scheme, we achieved bootstrapping depth $\log h + \log\log_p(ht)$. In case of BGV, we have reduced the bootstrapping degree from $\log h + 2\log(t)$ to $\log h + \log t$.
- We provided a first implementation of the bootstrapping functionality for FV scheme in the SEAL library [26]. We also implemented our revised digit extraction algorithm in HElib which can directly be applied to improve HElib bootstrapping for large plaintext modulus p^r.
- We also introduced a light-weight mode of bootstrapping which we call the "slim mode" by restricting the plaintexts to a subspace. In this mode, messages are batched-vectors where each slot only holds a value in $\mathbb{Z}_{p^r}$, instead of its degree-d extension. We argue that the slim mode may be more applicable in many use-cases of FHE, including machine learning over encrypted data.

We implemented the slim mode of bootstrapping in SEAL and showed that in this mode, bootstrapping is about d times faster, hence we can achieve a similar throughput as in the regular mode.

1.2 Related works

After bootstrapping was introduced by Gentry at 2009, many methods are proposed to improve its efficiency. Existing bootstrapping implementations can be classified into three branches. The first branch [20,24] builds on top of somewhat homomorphic encryption schemes based on RLWE problem. The second branch aims at minimizing the time to bootstrap one single bit of message after each boolean gate evaluation. Works in this direction include [3,14,15,17]. They were able to obtain very fast results: less than 0.1 seconds for a single bootstrapping. The last branch considers bootstrapping over integer-based homomorphic encryption schemes under the sparse subset sum problem assumption [11,16,27,30]. (fixme: cite) use a squashed decryption circuit and evaluate bit-wise (or digit-wise) addition in encrypted state instead of doing a digit extraction. In [11], they show that using digit extraction for bootstrapping results in lower computational complexity while consuming a similar amount of depth as previous approaches.

Our work falls into the first branch. We aim at improving the bootstrapping procedure for the two schemes BGV and FV, with the goal of improving the throughput and after level for bootstrapping in case of large plaintext modulus. Therefore, our main point of comparison in this paper will be the work of Halevi and Shoup which is in the same branch [24]. We note that a digit extraction procedure is used for all branches except the second one. Therefore, improving the digit extraction procedure is one of the main tasks for an efficient bootstrapping algorithm.

1.3 Roadmap

In section 2, we introduce notations and necessary background on the BGV and FV schemes. In section 3, after reviewing the digit extraction procedure of [24], we define the lowest digit removal polynomials, and use them to give an improved digit removal algorithm. In section 4, we describe our method for bootstrapping in the FV scheme, and how our algorithm leads to an improved bootstrapping for BGV scheme when the plaintext modulus is p^r with $r > 1$. In section 5, we present and discuss our performance results. Finally, in section 6 we conclude with future directions. Proofs and more details regarding the SEAL implementation of bootstrapping are included in the Appendix.

1.4 Acknowledgements

We wish to thank Kim Laine, Amir Jalali and Zhicong Huang for implementing new functionalities and significant performance optimizations to SEAL. We thank Shai Halevi for helpful discussions on bootstrapping in HElib.

2 Background

2.1 Basics of BGV and FV schemes

First, we introduce some notations. Both BGV and FV schemes are initialized with integer parameters m, t and q. Here m is the cyclotomic field index, t is the plaintext modulus, and q is the coefficient modulus. Note that in BGV, it is required that $(t, q) = 1$.

Let $\phi_m(x)$ denote the m-th cyclotomic polynomial and let $n = \varphi(m)$ be the degree of $\phi_m(x)$. We use the following common notations $R = \mathbb{Z}[x]/(\phi_m(x))$, $R_t = R/tR$, and $R_q = R/qR$. In both schemes, the secret key s is an element of R_q. It is usually taken to be ternary (i.e., each coefficient is either -1, 0 or 1) and often sparse (i.e., the number of nonzero coefficients of s are bounded by some $h \ll n$). A ciphertext is a pair (c_0, c_1) of elements in R_q.

Decryption formula. The decryption of both schemes starts with a dot-product with the extended secret-key $(1, s)$. In BGV, we have

$$c_0 + c_1 s = m + tv + \alpha q,$$

and decryption returns $m = ((c_0 + c_1 s) \mod q) \mod t$. where as in FV, the equation is

$$c_0 + c_1 s = \Delta m + v + \alpha q$$

and decryption formula is $m = \lfloor \frac{(c_0 + c_1 s) \mod q}{\Delta} \rceil$.

Plaintext space. The native plaintext space in both schemes is R_t, which consists of polynomials with degree less than n and integer coefficients between 0 and $t - 1$. Additions and multiplications of these polynomials are performed modulo both $\phi_m(x)$ and t.

A widely used plaintext-batching technique [29] turns the plaintext space into a vector over certain finite rings. Since batching is used extensively in our bootstrapping algorithm, we recall the details here. Suppose $t = p^r$ is a prime power, and assume p and m are coprime. Then $\phi_m(x) \mod p^r$ factors into a product of k irreducible polynomials of degree d. Moreover, d is equal to the order of p in $\mathbb{Z}_m^*$, and k is equal to the size of the quotient group $\mathbb{Z}_m^*/\langle p \rangle$. For convenience, we fix a set $S = \{s_1, \ldots, s_k\}$ of integer representatives of the quotient group. Let f be one of the irreducible factors of $\phi_m(x) \mod p^r$, and consider the finite extension ring

$$E = \mathbb{Z}_{p^r}[x]/(f(x)).$$

Then all primitive m-th roots of unity exist in E. Fix $\zeta \in E$ to be one such root. Then we have a ring isomorphism

$$\begin{aligned} R_t &\to E^k \\ m(x) &\mapsto (m(\zeta^{s_1}), m(\zeta^{s_2}), \ldots, m(\zeta^{s_k})) \end{aligned}$$

Using this isomorphism, we can regard the plaintexts as vectors over E, and additions/multiplications between the plaintexts are executed coeffcient-wise on the components of the vectors, which are often called *slots*.

In the reset of the paper, we will move between the above two ways of viewing the plaintexts, and we will distinguishing them by writing them as polynomials (no batching) and vectors (batching). For example, $\mathsf{Enc}(m(x))$ means an encryption of $m(x) \in R_t$, whereas $\mathsf{Enc}((m_1, \ldots, m_k))$ means a batch encryption of a vector $(m_1, \ldots, m_k) \in E^k$.

Modulus switching. Modulus switching is a technique which scales a ciphertext (c_0, c_1) with modulus q to another one (c_0', c_1') with modulus q' that decrypts to the same message. In BGV, modulus switching is a necessary technique to reduce the noise growth. Modulus switching is not necessary for FV, at least if used in the LHE mode. However, it will be of crucial use in our bootstrapping procedure. More precisely, modulus switching in BGV requires q and q' to be both coprime to t. For simplicity, suppose $q = q' = 1(\mod t)$. Then $c_i' =$ the closest integer polynomial to $\frac{q'}{q}c$ such that $c_i' \equiv c_i \mod t$. For FV, q and q' do not need to be coprime to t, and modulus switching simply does scaling and rounding to integers, i.e., $c_i' = \lfloor q'/qc_i \rceil$.

We stress that modulus switching slightly increase the noise-to-modulus ratio due to rounding errors in the process. Therefore, one can not switch to arbitrarily small modulus q'. On the other hand, in bootstrapping we often like to switch to a small q'. The following lemma puts a lower bound on the size of q' for FV (the case for BGV is similar).

Lemma 1 *Suppose $c_0 + c_1 s = \Delta m + v + aq$ is a ciphertext in FV with $|v| < \Delta/4$. if $q' > 4t(1 + l_1(s))$, and (c_0', c_1') is the ciphertext after switching the mdoulus to q', then (c_0', c_1') also decrypts to m.*

Proof. See appendix.

We remark that although the requirement in BGV that q and t are coprime seems innocent, it affects the depth of the decryption circuit when t is large. Therefore, it results in an advantage for doing bootstrapping in FV over BGV. We will elaborate on this point later.

Multiply and divide by p in plaintext space. In bootstrapping, we will use following functionalities: dividing by p, which takes an encryption of $pm \mod p^e$ and returns an encryption of $m \mod p^{e-1}$, and multiplying by p which is the inverse of division. In BGV scheme, multiply by p can be realized via a fast scalar multiplication $(c_0, c_1) \to ((pc_0) \mod q, (pc_1) \mod q)$. In the FV scheme, these operations are essentially free, because if $c_0 + c_1 s = \lfloor \frac{q}{p^{e-1}} \rfloor m + v + q\alpha$, then the same ciphertext satisfies $c_0 + c_1 s = \lfloor \frac{q}{p^e} \rfloor pm + v + v' + q\alpha$ for some small v'. In the rest of the paper, we will omit these operations, assuming that they are free to perform.

3 Digit Removal Algorithm

In the previous method for digit extraction, some lifting polynomials with good properties are used. We used a family of "lowest digit removal" polynomials which have a stronger lifting property. We then combined these lowest digit removal polynomials with the lifting polynomials to construct a new digit removal algorithm.

For convenience of exposition, we use some slightly modified notations from [24]. Fix a prime p. Let z be an integer with (balanced) base-p expansion $z = \sum_{i=0}^{e-1} z_i p^i$. For integers $i, j \geq 0$, we use $z_{i,j}$ to denote any integer with first base-p digit equal to z_i and the next j digits zero. In other words, we have $z_{i,j} \equiv z_i \mod p^{j+1}$.

3.1 Reviewing the digit extraction method of Halevi and Shoup

The bootstrapping procedure in [24] consists of five main steps: modulus switching, dot product (with an encrypted secret key), linear transform, digit extraction, and another "inverse" linear transform. Among these, the digit extraction step dominates the cost, in terms of both depth and work. Hence we will focus on optimizing the digit extraction. Essentially, it executes the following functionality.

$\mathsf{DigitRemove}(p, e, r)$: fix prime p, for two integers $r < e$ and an input $u \mod p^e$, let $u = \sum u_i p^i$ with $|u_i| \leq p/2$ when p is odd (and $u_i = 0, 1$ when $p = 2$), returns

$$u\langle r, \ldots, e-1\rangle := \sum_{i=r}^{e-1} u_i p^i.$$

We say this functionality "removes " the r lowest significant digits in base p from an e-digits integer. To realize the above functionality over homomorphically encrypted data, the authors in [24] constructed some special polynomials $F_e(\cdot)$ which has the following lifting property.

Lemma 2 (Corollary 5.5 in [24]) *For every prime p and $e \geq 1$ there exist a degree p-polynomial F_e such that for every integer z_0, z_1 with $z_0 \in [p]$ and every $1 \leq e' \leq e$ we have $F_e(z_0 + p^{e'} z_1) = z_0 \pmod{p^{e'+1}}$.*

For example, if $p = 2$, we can take $F_e(x) = x^2$. One then uses these lifting polynomials F_e to extract each digit u_i from u in a successive fashion. The digit extraction procedure is defined in Figure 1 in [24], and it can be visualized in the following diagram.

In the below diagram, the top-left entry is the input. This algorithm starts with the top row. From left to right, it successively applies the lifting polynomial to obtain all the blue entries. Then the green entry on the next row can be

$$
\begin{array}{lllllll}
u = u_{0,0} & u_{0,1} & \cdots & & u_{0,r-1} & \cdots & u_{0,e-1} \\
u_{1,0} & u_{1,1} & \cdots & u_{1,r-2} & \cdots & u_{1,e-2} & \\
\vdots & \vdots & & & \iddots & & \\
u_{e-2,0} & u_{e-2,1} & & & & & \\
u_{e-1,0} & & & & & &
\end{array}
$$

obtained from subtracting all blue digits on the same diagonal from the input and then dividing by p. When this procedure concludes, the (i,j)-th entry of the diagram will be $u_{i,j}$. In particular, digits on the final diagonal will be $u_{i,e-1-i}$. Then it can be seen that

$$u\langle r, \cdots, e-1\rangle = u - \sum_{i=0}^{r-1} u_{i,e-1-i} \cdot p^i.$$

3.2 Lowest digit removal polynomials

Recall that in the method above, it is not enough to obtain the $u_i \mod p$. Rather, one requires $u_{i,e-1-i}$. The reason is one has to clear the higher digits to create numbers with base -p expansion $(u_i, \underbrace{0, 0, \ldots, 0}_{e-1-i})$, otherwise it will mess up the u'_i for $i' > i$. Previously, to obtain $u_{i,j}$, one needs to apply the lifting polynomial j times. Actually, there is a lower-degree polynomial with the same functionality.

Lemma 3 *Let p be a prime and $n \geq 1$. Then there exists a polynomial f of degree at most $(e-1)(p-1)+1$ such that for every integer $0 \leq x < p^e$, we have*

$$f(x) \equiv x - (x \mod p) \mod p^e,$$

where $|x \mod p| \leq (p-1)/2$ when p is odd.

Proof. We complete the proof sketch in [25] by adding in the necessary details. To begin, we introduce a function

$$F_A(x) := \sum_{j=0}^{\infty} (-1)^j \binom{A+j-1}{j} \binom{x}{A+j}.$$

This function $F_A(x)$ converges on every integer, and for $M \in \mathbb{Z}$,

$$F_A(M) = \begin{cases} 1 & \text{if } M > A \\ 0 & \text{otherwise.} \end{cases}$$

Define $\hat{f}(x)$ as

$$\hat{f}(x) = p \sum_{j=1}^{\infty} F_{j \cdot p}(x) = \sum_{m=p}^{\infty} a(m) \binom{x}{m}. \tag{1}$$

We can verify that the function $\hat{f}(x)$ satisfies the properties in the lemma (for the least residue system), but its degree is infinite. So we let

$$f(x) = \sum_{m=p}^{(e-1)(p-1)+1} a(m)\binom{x}{m}.$$

Now we will prove that the polynomial $f(x)$ has p-integral coefficients and has the same value with $\hat{f}(x)$ for $x \in \mathbb{Z}_{p^e}$.

Claim. $f(x)$ has p-integral coefficients and $a(m)\binom{x}{m}$ is multiple of p^e for all $x \in \mathbb{Z}$ when $m > (e-1)(p-1)+1$.

Proof. If we rewrite the equation 1,

$$\hat{f}(x) = p\sum_{j=1}^{\infty} F_{j\cdot p}(x) = p\sum_{j=1}^{\infty}\left(\sum_{i=0}^{\infty}(-1)^i\binom{jp+i-1}{i}\binom{x}{jp+i}\right).$$

By replacing the $jp+i$ to m, we arrive at the following equation:

$$a(m) = p\sum_{k=1}^{\infty}(-1)^{m-kp}\binom{m-1}{m-kp}.$$

In the equation, we can notice that the term $(-1)^{m-kp}\binom{m-1}{m-kp}$ is the coefficient of X^{m-pk} in the Taylor expansion of $(1+X)^{-kp}$. Therefore, $a(m)$ is actually the coefficient of X^m in the Taylor expansion of $\sum_{k=1}^{\infty} pX^{kp}(1+X)^{-kp}$.

$$\sum_{k=1}^{\infty} pX^{kp}(1+X)^{-kp} = p\sum_{k=1}^{\infty}(\frac{X}{X+1})^{kp} = p\frac{(1+X)^p}{(1+X)^p - X^p}$$

We can get a m-th coefficient of Taylor expansion from following equation:

$$p\frac{(1+X)^p}{(1+X)^p - X^p} = p\frac{(1+X)^p}{1+B(X)} = p(1+X)^p(1 - B(X) + B(X)^2 - \cdots).$$

Because $B(X)$ is multiple of pX, the coefficient of X^m can be obtained from a finite number of powers of $B(X)$. We can also find out the degree of $B(X)$ is $p-1$, so

$$\mathsf{Deg}(p(1+X)^p(1 - B(X) + \cdots + (-1)^{(e-2)}B(X)^{(e-2)})) = (e-1)(p-1)+1.$$

Hence these terms do not contribute to X^m. This means that $a(m)$ is m-th coefficient of

$$p(1+X)^p B(X)^{e-1}\sum_{i=0}^{\infty}(-1)^i B(X)^i$$

which is multiple of p^e (since $B(X)$ is multiple of p). ■

By the claim above, the p-adic valuation of $a(m)$ is larger than $\frac{m}{p-1}$ and it is trivial that the p-adic valuation of $m!$ is less than $\frac{m}{p-1}$. Therefore, we proved that the coefficients of $f(x)$ are p-integral. Indeed, we proved that $a(m)\binom{x}{m}$ is multiple of p^n for any integer when $m > (e-1)(p-1)+1$. This means that $\hat{f}(x) = f(x) \bmod p^e$ for all $x \in \mathbb{Z}_{p^e}$.

As a result, the degree $(e-1)(p-1)+1$ polynomial $f(x)$ satisfies the conditions in lemma for the least residue system. For balanced residue system, we can just replace $f(x)$ by $f(x+p/2)$. □

Note that the above polynomial $f(x)$ removes the lowest base-p digit in an integer. It is also desirable sometimes to "retain" the lowest digit, while setting all the other digits to zero. This can be easily done via $g(x) = x - f(x)$. In the rest of the paper, we will denote such polynomial that retains the lowest digit by $G_{n,p}(x)$ (or $G_n(x)$ if p is clear from context). In other words, if $x \in \mathbb{Z}_{p^e}$ and $x \equiv x_0 \mod p$, with $|x_0| \leq p/2$, then $G_e(x) = x_0 \mod p^e$.

Example 4 *When $n = 2$, we have $f(x) = -x(x-1)\cdots(x-p)$ and $G_2(x) = x - f(x)$.*

We recall that in previous method, it takes degree p^{e-i-1} and $(e-i-1)$ evaluations of polynomials of degree p to obtain $u_{i,e-i}$. With our lowest digit removing polynomial, it only takes degree $(e-i-1)(p-1)+1$. Therefore, we can use the digit removal polynomials to reduce some lifting work in the previous method.

As a result, by combining the lifting polynomials and lowest digit removing we can make the digit extraction algorithm faster with lower depth.

The following diagram illustrates how our new digit removal algorithm works. First, each blue digit is obtained by evaluating lift polynomial to the entry on its left. Then, the red digit on each row is obtained by evaluating the remaining lowest digit polynomial to the left-most digit on its row. Finally, green digits are obtained by subtracting all the blue digits on the same diagonal from the input, and dividing by p.

$$
\begin{array}{llcll}
u_{0,0} & u_{0,1} & \cdots & u_{0,r-2} & u_{0,r-1} \qquad\qquad u_{0,e-1} \\
u_{1,0} & u_{1,1} & \cdots & u_{1,r-2} & \qquad\qquad u_{1,e-2} \\
\vdots & & & & \\
u_{r-2,0} & u_{r-2,1} & & u_{r-2,e-r+1} & \\
u_{r-1,0} & \qquad u_{r-1,e-r} & & &
\end{array}
$$

Finally, in order to remove the r lowest digits, we subtract from the input all the red digits obtained. We note that the major difference of this procedure is that we only need to populate the top left triangle of side length r, plus the right most r-by-1 diagonal, where as the previous method needs to populate the entire triangle of side length e.

Moreover, the red digits in our method has lower depth: in previous method, the i-th red digit is obtained by evaluating lift polynomial $(e-i-1)$ times, hence its degree is p^{e-i-1} on top of the i-th green digit. However, in our method, its degree is only $(p-1)(e-i-1)+1$ on top of the i-th green digit, which has degree at most p^i, the total degree of the algorithm is bounded by the maximum degree over all the red digits

$$\max_{0 \le i < r} p^i((e-1-i)(p-1)+1)$$

Since each individual term is bounded by ep^r, the degree is at most ep^r. This makes our digit extraction method lower degree, which is critical to bootstrapping in HE.

Data: $x \in \mathbb{Z}_{p^e}$
Result: $x - [x]_{p^r} \bmod p^e$
// $F_i(x)$: `lift poly with` $F_i(x + O(p^i)) = x + O(p^{i+1})$
// $G_i(x)$: `lowest digit remain poly with` $G_i(x) = [x]_p \bmod p^i$
Find largest ℓ such that $p^\ell < (p-1)(e-1)+1$;
Initialize res $= x$;
for $i \in [0, r)$ **do**
  // r `time evaluate lowest digit remain polynomial`
  $R_i = G_{e-i}(x')$; // $R_i = x_i \bmod p^{e-i}$
  $R_i = R_i \cdot p^i$; // $R_i = x_i p^i \bmod p^e$
  if $i < r-1$ **then**
    // $(r-1)$ `time evaluate lift polynomial`
    $L_{i,0} = F_1(x')$
  end
  for $j \in [0, \ell-2)$ **do**
    if $i + j < r - 1$ **then**
      $L_{i,j+1} = F_{j+2}(L_{i,j})$
    end
  end
  if $i < r-1$ **then**
    $x' = x$;
    for $j \in [0, i+1)$ **do**
      if $i - j > \ell - 2$ **then**
        $x' = x' - R_j$
      end
      else
        $x' = x' - L_{j,i-j}$
      end
    end
  end
  res $=$ res $- R_i$;
end

Algorithm 1: New method for extract r bottom digits from $x \in \mathbb{Z}_{p^e}$

3.3 Improved algorithm for removing digits

We discuss one further optimization to remove r lowest digits in base p from an e-digit integer. If ℓ is an integer such that $p^\ell > (p-1)(e-1)+1$, then instead of using lifting polynomials to obtain the ℓ-th digit, we can just use the result of evaluating remain polynomial (or, the red digit) to obtain the green digit in the next row. This could save some work and also lowers the depth of the overall procedure.

The depth and computation cost of Algorithm 1 is summarized in Theorem 5. The depth of Algorithm 1 is simply the maximum depth of all the removed digits. To determine the computational cost to evaluate Algorithm 1 homomorphically, we need to specify the unit of measurement. Since constant-ciphertext multiplication is much faster than FHE schemes than ciphertext-ciphertext multiplications, we choose to measure by the number of multiplications. Thus, we will measure the computational cost in terms of ciphertext-ciphertext multiplications. The Paterson-Stockmeyer algorithm [28], already used by HElib evaluates a polynomial of degree d with $\sim \sqrt{2d}$ non-constant multiplications, and we use that as the base of our estimate.

Theorem 5. *Algorithm 1 is correct. Its depth is bounded above by*

$$\log(ep^r) = \log(e) + r\log(p).$$

The number of non-constant multiplications is asymptotically equal to

$$\left\{ \frac{2}{3}(e^{\frac{3}{2}} - (e-r)^{\frac{3}{2}}) + (1+\log_p(e))r \right\} \cdot \sqrt{2p}.$$

Table 1 compares the asymptotic depth and number of non-constant multiplications between our method for digit removal and the method of [24]. From the table, we see that the advantage of our method grows with the difference $e-r$.

Method	Depth	No. multiplications
[24]	$O(e\log(p))$	$O(e^2\sqrt{p})$
This work	$O(\log(e)+r\log(p))$	$O(r(\sqrt{e}+\log_p(e))\sqrt{p})$

Table 1: Complexity of $\mathsf{DigitRemove}(p,e,r)$

$$\begin{aligned}
&\mathsf{Enc}(\mathbf{m}(x)) = \mathsf{Enc}((\mathbf{m}_0(x), \cdots, \mathbf{m}_{k-1}(x))) \\
&\qquad\qquad \downarrow \quad \text{Modulus Switching and Dot Product} \\
&\mathsf{Enc}(\mathbf{m}(x) \cdot p^{e-r} + e(x)) \\
&\qquad\qquad \downarrow \quad \mathsf{LinearTransformation} \\
&\mathsf{Enc}((m_0 \cdot p^{e-r} + e_0, \cdots, m_{k-1} \cdot p^{e-r} + e_{k-1})), \cdots \mathsf{Enc}((m_{n-k} \cdot p^{e-r} + e_{n-k}, \cdots, m_{n-1} \cdot p^{e-r} + e_{n-1})) \\
&\qquad\qquad \downarrow \quad d \text{ number of Digit Extraction} \\
&\mathsf{Enc}((m_0, m_1, \cdots, m_{k-1})), \mathsf{Enc}((m_k, \cdots, m_{2k-1})), \cdots \mathsf{Enc}((m_{n-k}, \cdots, m_{n-1})) \\
&\qquad\qquad \downarrow \quad \mathsf{InverseLinearTransformation} \\
&\mathsf{Enc}(\mathbf{m}(x)) = \mathsf{Enc}(\mathbf{m}_0(x), \cdots, \mathbf{m}_{k-1}(x))
\end{aligned}$$

Fig. 1: Regular bootstrapping process

4 Bootstrapping in the FV scheme

4.1 Reviewing the method of [24]

The bootstrapping for FV scheme follows the main steps from [24] for the BGV scheme, while we make two modifications in modulus switching and digit extraction. First, we review the procedure in [24].

Modulus Switching. One fixes some $q' < q$ and compute a new ciphertext c' which encrypts the same plaintext but has much smaller size.

Dot product with bootstrapping key. Here we compute homomorphically the dot product $\langle c', \mathfrak{s} \rangle$, where $\mathfrak{s}$ is an encryption of a new secret key s' under a large coefficient modulus Q and a new plaintext modulus $t' = p^{e+r}$. The result of this step is an encryption of $m + tv$ under the new parameters (s', t', Q).

Linear Transformation. Let d denote the multiplicative order of p in $\mathbb{Z}_m^*$ and $k = n/d$ be the number of slots supported in plaintext batching (see Section on batching). Assuming the plaintext modulus is a prime power p^r. Suppose the input to linear transform is an encryption of $\sum_{i=0}^{n-1} a_i x^i$, then the output of this step is d ciphertexts $C_0, \ldots, C_d$, where C_j is a batch encryption of $(a_{jk}, a_{jk+1}, \ldots, a_{jk+k-1})$.

Digit Extraction. When the above steps are done, we obtain d ciphertexts, where the first ciphertext is a batch encryption of

$$(m_0 \cdot p^{e-r} + e_0, m_1 \cdot p^{e-r} + e_1, \cdots, m_{k-1} \cdot p^{e-r} + e_{k-1}).$$

Assuming that $|e_i| \leq \frac{p^{e-r}}{2}$ for each i, we will apply Algorithm 1 to remove the lower digits e_i, resulting in d new ciphertexts encrypting Δm_i for $0 \leq i < n$ in their slots. Then we perform a free division to get d ciphertexts, encrypting m_i in their slots.

Inverse Linear Transformation. Finally, we apply another linear transformation which combines the d ciphertexts into one single ciphertext encrypting $m(x)$.

4.2 Our modifications

Suppose $t = p^r$ is a prime power, and we have a ciphertext (c_0, c_1) modulo q. Here, instead of switching to a modulus q' coprime to p as done in BGV, we switch to $q' = p^e$, and obtain ciphertext (c'_0, c'_1) such that

$$c'_0 + c'_1 s = p^{e-r} m + v + \alpha p^e.$$

Then, one input ciphertext to the digit extraction step will be a batch encryption

$$\mathsf{Enc}((p^{e-r} m_0 + v_0, \ldots, p^{e-r} m_k + v_k))$$

under plaintext modulus p^e. Hence this step requires $\mathsf{DigitRemove}(p, e, e-r)$, which is almost exactly the same as the BGV digit extraction procedure outlined in Section 2.

However, there is a crucial difference between the two schemes on the parameter e. In case of FV, by Lemma 1, we can choose (roughly) $e = r + \log_p(l_1(s))$. For BGV though, the estimate of e for correct bootstrapping in [24] is

$$e \geq 2r + \log_p(l_1(s)).$$

We can analyze the impact of this difference on the depth of digit removal, and therefore on the depth of bootstrapping. Using the formula in Theorem 5, the depth for the BGV case is

$$(r + \log_p(l_1(s)) \log p + log(2r + \log_p(l_1(s)).$$

We can substitute $r = \log_p(t)$ into the above formula and simplify the depth for BGV bootstrapping as

$$\log t + \log(l_1(s)) + \log(2 \log_p(t) + \log_p(l_1(s)). \tag{2}$$

Note that the depth grows linearly with the logarithm of the plaintext modulus t. On the other hand, the bootstrapping depth for FV case turns out to be

$$\log(l_1(s)) + \log(\log_p(t) + \log_p(l_1(s)).$$

The depth for FV grows linearly with only $\log \log t$, which is smaller than BGV in the large plaintext modulus regime. It is thus interesting future work to investigate whether this difference is inherent for the two schemes, or it is possible to modify the bootstrapping algorithm for BGV to achieve the same asymptotic depth.

We can also compare the number of ciphertext multiplications needed for the digit extraction procedures. If the plaintext modulus is $t = p^r$, then in the

digit extraction in bootstrapping, we need to remove the lowest $(e-r)$ digits from e digits. By replacing r with $e-r$ in the second formula in Theorem 5 and throwing away insignificant terms, the work is determined by the quantity

$$\sqrt{p} \cdot (e^{3/2} - r^{3/2}).$$

Letting $e = 2r + \log_p(l_1(s))$ (resp. $e = r + \log_p(l_1(s))$ we see that the work is dominated by $\sqrt{p}r^{3/2}$ for BGV (resp. $\sqrt{p}r^{1/2}\log_p(l_1(s))$ for FV). Hence when t is large, FV requires less work work for the digit extraction procedure in bootstrapping than BGV.

4.3 Improved bootstrapping for BGV

To apply our ideas to improve the digit extraction step in BGV bootstrapping, we simply replace the algorithm in [24] with our digit removal Algorithm 1. Recall that their algorithm has depth $(e-1)\log p$, and takes $e(e-1)/2$ homomorphic evaluations of polynomials of degree p. If we use the Paterson-Stockmeyer method for polynomial evaluation, then the total amount of ciphertext-ciphertext multiplications is

$$\frac{1}{2}(e-1)(e-2)\sqrt{2p}$$

Plugging in the lower bound $e \geq 2r + \log_p(l_1(s))$, we obtain an estimate of depth and work for the original BGV bootstrapping method in [24]. The depth is

$$2\log(t) + l_1(s).$$

Comparing with Equation 2, we see that our revised algorithm for BGV bootstrapping, we see that our algorithm reduced the dependence of bootstrapping depth on the plaintext modulus t from $2\log t$ to $\log t$.

As another advantage of our revised BGV bootstrapping, we make a note on security. They uses sparse secrets in [24], hence its parameters are less secure under the new attack in [2]. Using our method, since the overall depth is reduced for the large plaintext modulus case, we could use smaller modulus q, increasing the security level. At the same time, smaller q immediately lead to faster homomorphic operations.

4.4 Slim bootstrapping algorithm

The bootstrapping algorithm for FV is expensive mainly because of d repetitions of digit extraction. In many parameters, the extension degree d is large for small plaintext modulus. However, many interesting applications requires arithmetic over $\mathbb{Z}_{p^r}$ rather than the degree-d extension ring, making it hard to utilize the full plaintext space.

Therefore we will introduce one more bootstrapping algorithm which is called "slim" bootstrapping. This bootstrapping algorithm works with the plaintext space $\mathcal{M} = \mathbb{Z}_t^n$, embedded as a subspace of R_t through the batching isomorphism.

This method can be adapted using almost the same algorithm as the original bootstrapping algorithm, except that we only need to perform one digit extraction operation, hence it is roughly d times faster than the regular bootstrapping algorithm. Also, we need to revise the linear transformation and inverse linear transformation to specify to this case. We give an outline of our slim bootstrapping algorithm below.

$$
\begin{aligned}
&\mathsf{Enc}(m_0, m_1, m_2, \cdots, m_{k-1}) \\
&\qquad \downarrow \quad \mathsf{InverseLinearTransformation} \\
&\mathsf{Enc}(m_0 + m_1 x^d + \cdots + m_{k-1} x^{d(k-1)}) \\
&\qquad \downarrow \quad \mathsf{Modulus\ Switching\ and\ Dot\ Product} \\
&\mathsf{Enc}(m(x) \cdot p^{e-r} + e(x)) \\
&\qquad \downarrow \quad \mathsf{LinearTransformation} \\
&\mathsf{Enc}(m_0 \cdot p^{e-r} + e_0, \cdots, m_{k-1} \cdot p^{e-r} + e_{k-1}) \\
&\qquad \downarrow \quad \mathsf{Digit\ Extraction} \\
&\mathsf{Enc}(m_0, m_1, m_2, \cdots, m_{k-1})
\end{aligned}
$$

Fig. 2: slim bootstrapping process

Inverse Linear Transformation. We take as input a batch encryption of $(m_1 \ldots, m_k) \in \mathbb{Z}_{p^r}$. In the first step, we apply an "inverse" linear transformation to obtain an encryption of $m_1 + m_2 x^d + \ldots + m_k x^{d(k-1)}$. This can be done using k slot rotations and k plaintext multiplications.

Modulus Switching and Dot product with bootstrapping key. These two steps are exactly the same as the full bootstrapping procedure. After these steps, we obtain a (low-noise) encryption of

$$(\Delta m_1 + v_1 + (\Delta m_2 + v_2) x^d + \ldots + (\Delta m_k + v_k) x^{d(k-1)}).$$

Linear Transformation. In this step, we apply another linear transformation, consisting of k slot permutations and k plain multiplications to obtain a batch encryption of $(\Delta m_1 + v_1, \ldots, \Delta m_k + v_k)$. Revised linear transformation for slim bootstrapping is in appendix.

Digit extraction. Then, we apply digit-removal algorithm to remove the coefficients v_i, resulting in a batch encryption of $(\Delta m_1, \ldots, \Delta m_k)$. We then execute the free division and obtain a batch encryption of$(m_1, \ldots, m_k)$. This completes the bootstrapping process.

	[24]		Our Method	
(p, e, r)	Timing (sec)	Before/After level	Timing (sec)	Before/After level
$(2, 11, 5)$	15	23/3	16	23/**10**
$(2, 21, 13)$	264	56/16	**239**	56/**22**
$(5, 6, 3)$	49.5	39/5	**30**	39/**13**
$(17, 4, 2)$	61.2	38/5	**35.5**	38/**14**
$(31, 3, 1)$	26.3	32/8	**12.13**	32/**18**
$(127, 3, 1)$	73.2	42/3	**38**	42/**20**

Table 2: Comparison of digit removal algorithms in HElib (Toshiba Portege Z30t-C laptop with 2.6GHz CPU and 8GB memory)

5 Implementation and Performance

We implemented both the regular mode and the slim mode of bootstrapping in FV on the SEAL library. We also implemented our revised digit extraction procedure in HElib. Since SEAL only supports power-of-two cyclotomic rings, and p needs to be coprime to m in order to use batching, we can not use $p = 2$ for SEAL bootstrapping, and we chose $p = 127$ and $p = 257$ because they give more slots among primes of reasonable size.

The following tables in this section illustrate some results. We used sparse secrets with hamming weight 64 and 128, and we estimated security level using Martin Albrecht's LWE estimator [2].

We implemented Algorithm 1 on HElib and compared with the results of the original HElib implementation for removing r digits from e digits. From Table 2, we see that for $e \geq r + 2$ and p is large, our digit removal procedure can outperform the current HElib implementation in both depth and work. Therefore, for these settings, we can replace the digit extraction procedure in the recryption function in HElib, and obtain a direct improvement on after level and time for recryption. When $p = 2$ and r, e are small, the current HElib implementation can be faster, since the lifting polynomial is $F_e(x) = x^2$ and squaring operation is faster than generic multiplication. Also, when $e = r + 1$, i.e., the task is to remove all digits except the highest one, our digit removal method has similar performance to the HElib counterpart.

Table 3 and 4 hold timing results for the regular and slim modes of bootstrapping for FV implemented in SEAL. In this table, the column labeled "recrypt init. time" shows the time to compute the necessary data needed in bootstrapping. The "recrypt time" column shows the time it takes to perform one bootstrapping. The before (resp. after) level shows the maximal depth of circuit that can be evaluated on a freshly encrypted ciphertext (resp. freshly bootstrapped ciphertext). We use $\mathsf{R}(p^r, d)$ to denote a finite ring with degree d over base ring $\mathbb{Z}_{p^r}$.

Comparing the corresponding entries from Table 3 and 4, we see that the slim mode of bootstrapping is either close to or more than d times faster than the regular mode.

Parameters					Result			
n	$\log q$	Plaintext Space	Slots	Security	Before /After Level	Recrypt Time (sec)	Recrypt init. time (sec)	Memory usage (GB)
16384	558	$\mathsf{GF}(127^{256})$	64	92.9	24/7	2027	193	8.9
16384	558	$\mathsf{GF}(257^{128})$	128	92.9	22/4	1381	242	7.5
32768	806	$\mathsf{R}(127^2, 256)$	64	126.2	32/12	21295	658	27.6
32768	806	$\mathsf{R}(257^2, 128)$	128	126.2	23/6	11753	732	26.6

Table 3: Time table for bootstrapping for FV scheme, hw=128 (Intel(R) Core(TM) i7-4770 CPU with 3.4GHZ CPU and 32GB memory)

Parameters					Result			
n	$\log q$	Plaintext Space	Number of Slots	Security Parame-ter	Before /After level	Recrypt init time (sec)	Memory usage (GB)	Recrypt Time (sec)
16384	558	$\mathbb{Z}_{127}$	64	92.9	23/10	57	2.0	6.75
32768	806	$\mathbb{Z}_{127^2}$	64	126.2	25/11	59	2.0	30.2
32768	806	$\mathbb{Z}_{127^3}$	64	126.2	20/6	257	8.9	34.5
16384	558	$\mathbb{Z}_{257}$	128	92.9	22/7	59	2.0	10.8
32768	806	$\mathbb{Z}_{257}$	128	126.2	31/15	207	7.4	36.8
32768	806	$\mathbb{Z}_{257^2}$	128	126.2	23/7	196	7.4	42.1

Table 4: Time table for slim bootstrapping for FV scheme, hw=128) (Intel(R) Core(TM) i7-4770 CPU with 3.4GHZ CPU and 32GB memory)

6 Future directions

6.1 Bootstrapping Performance

In this paper, we obtained bootstrapping circuits for the FV scheme whose depths depend linearly on $\log\log t$. For the BGV scheme, we were able to improve the dependence on t from $2\log t$ to $\log t$. One interesting direction is to explore whether we can further improve bootstrapping depth for BGV.

We also presented a slim mode of bootstrapping, which operates on a subspace of plaintext space, equivalent to a vector of elements in $\mathbb{Z}_{p^r}$. The slim mode has a similar throughput as the regular mode while being much faster. For example, it takes less than 7 seconds to bootstrap a vector in $\mathbb{Z}_{127}^{64}$ with after level 10. However, the ciphertext sizes of the slim mode are the same as those of the regular mode, hence we have a larger ciphertext/message expansion ratio. It would be interesting to investigate whether we could reduce the ciphertext sizes while keeping the performance results.

6.2 Applications to Machine Learning

Machine learning over encrypted data is one of the signature use-cases of FHE, and is an active research area. Research works in this area can be divided into two categories: evaluating a pre-trained machine learning model over private testing data, or training a new model on private training data. Often times, the model evaluation requires a lower-depth circuit, and thus can be achieved using LHE. On the other hand, training a machine learning model requires a much deeper circuit, and bootstrapping becomes necessary. This may explain that there are few works in the model training direction.

In the model evaluation case (e.g. [4,5,22,23]), one encodes the data as either polynomials in R_t, or as elements of $\mathbb{Z}_t$ when batching is used. One distinguishing feature of these methods is that the scheme maintains the full precision of plaintexts as evaluations are performed, in contrast to computations over plaintext data, where floating point numbers are used and only a limited precision is maintained. This implies that the plaintext modulus t needs to be taken large enough to "hold the result".

In the training case, because of the large depth and size of the circuit, the above approach is simply infeasible: t needs to be so large that the homomomorphic evaluations become too inefficient, as pointed out in [**?**]. Therefore, some analog of plaintext truncation needs to be performed alongside the evaluation. However, in order to perform the truncation function homomorphically, one has to express the function as a polynomial. Fortunately, our digit removal algorithm can also be used as a truncation method over $\mathbb{Z}_{p^r}$. Therefore, we think that improving bootstrapping for prime power plaintext modulus has practical importance.

There is one other work [12] which does not fall into either categories. It performs homomorphic evaluation over point numbers and outputs an approximate

result. It modifies the BGV and FV schemes: instead of encoding noise and message in different parts of a ciphertexts, one puts noise in lower bits of messages, and uses modulus switching creatively as a plaintext management technique. As a result, they could evaluate deeper circuits with smaller HE parameters. It is then an interesting question whether there exists an efficient bootstrapping algorithm for this modified scheme

References

1. Carlos Aguilar-Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrede Lepoint. NFLlib: NTT-based fast lattice library. In *Cryptographers Track at the RSA Conference*, pages 341–356. Springer, 2016.
2. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.
3. Jean-François Biasse and Luis Ruiz. FHEW with efficient multibit bootstrapping. In *International Conference on Cryptology and Information Security in Latin America*, pages 119–135. Springer, 2015.
4. Charlotte Bonte, Carl Bootland, Joppe W. Bos, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Faster homomorphic function evaluation using non-integral base encoding. Cryptology ePrint Archive, Report 2017/333, 2017. http://eprint.iacr.org/2017/333.
5. Joppe W. Bos, Wouter Castryck, , Ilia Iliashenko, and Frederik Vercauteren. Privacy-friendly forecasting for the smart grid using homomorphic encryption and the group method of data handling. Cryptology ePrint Archive, Report 2016/1117, 2016. http://eprint.iacr.org/2016/1117.
6. Joppe W Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *Cryptography and Coding*, pages 45–64. Springer, 2013.
7. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, pages 868–886, 2012.
8. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, 2012.
9. Zvika Brakerski and Vinod 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. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on Computing*, 43(2):831–871, 2014.
11. Jung Hee Cheon, Kyoohyung Han, and Duhyeong Kim. Faster bootstrapping of FHE over the integers. *IACR Cryptology ePrint Archive*, 2017:79, 2017.
12. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Report 2016/421, 2016. http://eprint.iacr.org/2016/421.
13. Jung Hee Cheon and Damien Stehlé. Fully homomophic encryption over the integers revisited. In *Advances in Cryptology–EUROCRYPT 2015*, pages 513–536. Springer, 2015.
14. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika 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, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22, pages 3–33. Springer, 2016.

15. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Improving TFHE: faster packed homomorphic operations and efficient circuit bootstrapping. 2017. https://eprint.iacr.org/2017/430.
16. Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *Public-Key Cryptography–PKC 2014*, pages 311–328. Springer, 2014.
17. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology–EUROCRYPT 2015*, pages 617–640. Springer, 2015.
18. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. http://eprint.iacr.org/.
19. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.
20. Craig Gentry, Shai Halevi, and Nigel P Smart. Better bootstrapping in fully homomorphic encryption. In *Public Key Cryptography–PKC 2012*, pages 1–16. Springer, 2012.
21. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013*, pages 75–92. Springer, 2013.
22. Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.
23. Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pages 1–21. Springer, 2012.
24. Shai Halevi and Victor Shoup. Bootstrapping for helib. In *Advances in Cryptology–EUROCRYPT 2015*, pages 641–670. Springer, 2015.
25. Michael Griffin (https://mathoverflow.net/users/61910/michael griffin). Lowest degree of polynomial that removes the first digit of an integer in base p. MathOverflow. URL:https://mathoverflow.net/q/269282 (version: 2017-05-08).
26. Kim Laine and Rachel Player. Simple encrypted arithmetic library-SEAL (v2. 0). Technical report, Technical report, September, 2016.
27. Koji Nuida and Kaoru Kurosawa. (Batch) fully homomorphic encryption over integers for non-binary message spaces. In *Advances in Cryptology–EUROCRYPT 2015*, pages 537–555. Springer, 2015.
28. Michael S Paterson and Larry J Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2(1):60–66, 1973.
29. Nigel P Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Designs, codes and cryptography*, pages 1–25, 2014.
30. Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

A Optimizing the Linear Transform for Slim Bootstrapping

In our slim mode of bootstrapping, we used a linear transform which has the following property: suppose the input is an encryption of $\sum m_i x^i$. Then the output is a batch encryption of $(m_0, m_d, \ldots, m_{d(k-1)})$. A straightforward implementation of this functionality requires n slot permutations and n plaintext multiplications. However, in the case when n is a power of 2, we can break down the linear transform into two parts, which we call coefficient selection and sparse linear transform. This reduces the number of slot permutations to $\log(d)+k$ and the number of plaintext multiplications to k.

A.1 Coefficient selection

The first part of the optimized linear transform functionality can be viewed as a coefficient selection. This process get input $\mathsf{Enc}(m(x))$ and return $\mathsf{Enc}(m'(x))$ with $m'(x) = \sum_{i=0}^{n/d} m_{id} \cdot x^{id}$. In other words, it selects the coefficients of $m(x)$ with the exponent of x divisible by d. The following algorithm is specified to the case when n is power of two form. Using the property that $x^n = -1$ in the ring R, we can construct an automorphism ϕ_i of R such that

$$\phi_i : X^{2^i} \to X^{n+2^i} = -X^{2^i}.$$

For example $\phi_0(\cdot)$ negates all odd coefficients, because ϕ_0 maps X to $-X$. This means that $\frac{1}{2}(\phi_0(m(x)) + m(x))$ will remove all odd terms and double the even terms. Using this property we can make a recursive algorithm which return $m'(x) = \sum_{i=0}^{n/d} m_{id} \cdot x^{id}$ for power of two d.

- For given $m(x)$, First compute $m(x) + \phi_0(m(x)) = m_0(x)$.
- Recursively compute, $m_i(x) = \phi_i(m_{i-1}(x)) + m_{i-1}(x)$ for $1 \le i \le \log_2 d$.
- Compute $m'(x) = d^{-1} \cdot m_{\log_2 d} \bmod t$ for plain modulus t.
- Return $m'(x) = m_{\log_2 d}(x)$.

The function $\phi_i : X \to X^{\frac{n+2^i}{2^i}}$ can be evaluated homomorphically by using key switching technique. Another operation is just multiply $d^{-1} \bmod t$, so we can homomorphically obtain $\mathsf{Enc}(m'(x))$. This process needs $\log d$ slot rotations and additions.

A.2 Sparse Linear Transform

The desired functionality of the sparse linear transform is: take as input an encryption c of $\sum m_i x^{id}$ and output a batch encryption of $(m_0, m_1 \ldots, m_{k-1})$. We claim that this functionality can be expressed as $\sum_{i=0}^{k-1} \lambda_i \sigma_{s_i}(c)$, where $\lambda_i \in R_t$ and the s_i form a set of representatives of $\mathbb{Z}_m^*/\langle p \rangle$. This is because the input plaintext only has k nonzero coefficients $m_0, \ldots, m_{k-1}$. Hence for each i it is possible to write m_i as a linear combination of the evaluations of the input at k

different roots of unities. Therefore, this step only requires k slot rotations and k plaintext multiplications. We also can adapt baby and giant step method to reduce the number of rotation to $O(\sqrt{k})$. We omit the details.

B Memory usage

In our implementation of the bootstrapping procedure in SEAL, we pre-compute some data which are used in the linear transforms. The major part of the memory consumption consists of slot-permutation keys and plaintext polynomials. More precisely, each plaintext polynomial has size $n \log t$ bits. The size of one slot-permutation key in SEAL is $(2n \log q) \cdot \lfloor \frac{\log q}{62} \rceil$.

Here we report the number of such keys and plaintext polynomials used in our bootstrapping. In the regular mode, we need $2\sqrt{n}$ slot-permutation keys, and $2\sqrt{n} + d + k$ plaintext polynomials.

On the other hand, the slim mode of bootstrapping in SEAL requires considerably less memory. Both inverse linear transform and the linear transform can be implemented via the babystep-giantstep technique, each using only $2\sqrt{k}$ slot-permutation keys and k plaintext polynomials.

C Proofs

C.1 Proof of Lemma 1

Lemma 1 *Suppose $c_0 + c_1 s = \Delta m + v + aq$ is a ciphertext in FV with $|v| < \Delta/4$. if $q' > 4t(1 + l_1(s))$, and (c_0', c_1') is the ciphertext after switching the mdoulus to q', then (c_0', c_1') also decrypts to m.*

Proof. We define the invariant noise to be the term v_{inv} such that

$$\frac{t}{q}(c_0 + c_1 s) = m + v_{inv} + rt.$$

Decryption is correct as long as $||v_{inv}|| < \frac{1}{2}$. Now introducing the new modulus q', we have

$$\frac{t}{q'}\left(\frac{q'}{q}c_0 + \frac{q'}{q}c_1 s\right) = m + v_{inv} + rt.$$

Taking nearest integers of the coefficients on the left hand side, we arrive at

$$\frac{t}{q'}\left(\lfloor\frac{q'}{q}c_0\rceil + \lfloor\frac{q'}{q}c_1\rceil s\right) = m + v_{inv} + rt + \delta,$$

with the rounding error $||\delta|| \leq t/q'(1 + l_1(s))$. Thus the new invariant noise is

$$v_{inv'} = v_{inv} + \delta$$

We need $||\delta|| < 1/4$ for correct decryption. Hence the lower bound on q' is

$$q' > 4t(1 + l_1(s)).$$

C.2 Proof of Theorem 5

Theorem 5 *Algorithm 1 is correct. Its depth is bounded above by*

$$\log(ep^r) = \log(e) + r\log(p).$$

The number of non-constant multiplications is asymptotically equal to

$$\left\{ \frac{2}{3}(e^{\frac{3}{2}} - (e-r)^{\frac{3}{2}}) + (1+\log_p(e))r \right\} \cdot \sqrt{2p}.$$

Proof. Correctness of Algorithm 1 is easy to show. In fact, the only place we deviate from the algorithm in [24] for digit extraction is that we used the digits R_i to replace $x_{i,j}$ in certain places. Since R_i has lowest digit x_i followed by $(e-i-1)$ zeros, we can actually use it to replace $x_{i,j}$ for any $j \leq e-i-1$ and still maintain the correctness.

To analyze the depth, note that use polynomials of degree p^i to compute $z_{i,i}$ for $0 \leq i \leq r-1$. Then, to compute $z_{i,e-1-i}$, we can use a polynomial of degree $(n-1-i)(p-1)+1$. Since the final result is a sum of the terms $z_{i,e-1-i}$ for $0 \leq i < r$, the degree of the entire evaluation is given by

$$\max_{0 \leq i < r} p^i((e-1-i)(p-1)+1)$$

Since individual term above is bounded by ep^r, the degree is at most ep^r. Hence the depth of the algorithm is bounded by $\log(e) + r\log(p)$.

We now estimate the amount of work of our algorithm in terms of non-constant multiplications. It consists of two parts: evaluating lift polynomials and lowest digit removal polynomials. Let $W(n)$ denote the number of non-constant multiplications to evaluate a polynomial of degree n. Then the total work is

$$\sum_{i=1}^{r} W((e-i)(p-1)+1) + lrW(p)$$

where $l = \lfloor \log_p((e-1)(p-1)+1) \rfloor$ is the optimization parameter used in Algorithm 1. Since we used the Paterson-Stockmeyer algorithm for polynomial evaluation, we have $W(n) \sim \sqrt{2n}$. Substituting into the above formula, we obtain

$$\begin{aligned}
&\sum_{i=1}^{r} \sqrt{2((e-i)(p-1)+1)} + br\sqrt{2p} \\
&\sim \sqrt{2p} \sum_{i=1}^{r} \sqrt{e-i} + \sqrt{2p}(1+\log_p(e))r \\
&\sim \sqrt{2p} \int_{e-r}^{e} \sqrt{x}dx + \sqrt{2p}(1+\log_p(e))r \\
&= \sqrt{2p} \left\{ \frac{2}{3}(e^{3/2} - (e-r)^{3/2}) + (1+\log_p(e))r \right\}.
\end{aligned}$$

This completes the proof.