이학 박사 학위논문

Bootstrapping Methods for Homomorphic Encryption

(동형암호 재부팅 기법에 관한 연구)

2019년 2월

서울대학교 대학원

수리과학부

한규형

Bootstrapping Methods for Homomorphic Encryption

(동형암호 재부팅 기법에 관한 연구)

지도교수 천정희

이 논문을 이학 박사 학위논문으로 제출함

2018년 10월

서울대학교 대학원

수리과학부

한규형

한규형의 이학 박사 학위논문을 인준함

2018년 12월

Bootstrapping Methods for Homomorphic Encryption

A dissertation
submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
to the faculty of the Graduate School of
Seoul National University

by

Kyoohyung Han

Dissertation Director : Professor Jung Hee Cheon

Department of Mathematical Sciences
Seoul National University

February 2019

Abstract

Bootstrapping Methods for Homomorphic Encryption

Kyoohyung Han
Department of Mathematical Sciences
The Graduate School
Seoul National University

After Gentry's blueprint on homomorphic encryption (HE) scheme, various efficient schemes have been suggested. For unlimited number of operations between encrypted data, the bootstrapping process is necessary. There are only few works on bootstrapping procedure because of the complexity and inefficiency of bootstrapping. In this paper, we propose various method and techniques for improved bootstrapping algorithm, and we apply it to logistic regression on large scale encrypted data.

The bootstrapping process depends on based homomorphic encryption scheme. For various schemes such as BGV, BFV, HEAAN, and integer-based scheme, we improve bootstrapping algorithm. First, we improved bootstrapping for BGV (HElib) and FV (SEAL) schemes which is implemented by Microsoft Research and IMB respectively. The key process for bootstrapping in those two scheme is extracting lower digits of plaintext in encrypted state. We suggest new polynomial that removes lowest digit of input, and we apply it to bootstrapping with previous method. As a result, both the complexity and the consumed depth are reduced. Second, bootstrapping for multiple data needs homomorphic linear transformation. The complexity of this part is $O(n)$ for number of slot n, and this part becomes a bottleneck when we use large n. We use the structure of linear transformation which is used in bootstrapping, and we decompose the matrix which is corresponding to the transformation. By applying recursive strategy, we reduce the complexity to $O(\log n)$. Furthermore, we suggest new bootstrapping method for integer-based HE schemes which are based on approximate greatest common divisor problem. By using digit extraction instead of previous bit-wise approach, the complexity of bootstrapping algorithm reduced from $O(\mathsf{poly}(\lambda))$ to $O(\log^2 \lambda)$. Our implementation for this process shows 6 seconds which was about 3 minutes.

To show that bootstrapping can be used for practical application, we implement logistic regression on encrypted data with large scale. Our target data has $400,000$ samples, and each sample has 200 features. Because of the size of the data, direct application of homomorphic encryption scheme is almost impossible. Therefore, we decide the method for encryption to maximize the effect of multi-threading and SIMD operations in HE scheme. As a result, our homomorphic logistic regression takes about 16 hours for the target data. The output model has 0.8 AUROC with about 80% accuracy. Another experiment on MNIST dataset shows correctness of our implementation and method.

Key words: homomorphic encryption, privacy protection, bootstrapping
Student Number: 2013-20250

Contents

Chapter 1

Introduction

1.1 Homomorphic Encryption

Homomorphic Encryption (HE) allows computations on encrypted data, and this can be used to evaluate in an untrusted party. The concept of this scheme is proposed by Rivest et al. [RAD78] with a concrete scheme. Since 2009, however, all proposed schemes are broken by various attacks. At 2009, Gentry introduced the first HE scheme based on ideal lattice problem [Gen09]. After that, there has been a large collection of works (e.g., [BGV12, Bra12, FV12, BLLN13, vDGHV10, CLT14, CS15, CKKS17]), with huge performance improvements.

These schemes follow Gentry's original blueprint, where each ciphertext is associated with a certain amount of "noise". The ratio between ciphertext moduli and the noise grows as homomorphic evaluations are performed. When the ratio is over some threshold, decryption will fail to give the correct result. For this reason, if no additional measure is taken, one set of parameters can only evaluate circuits of a bounded depth. This approach is called leveled homomorphic encryption (LHE) and is used in a many works.

Bootstrapping

However, if we wish to homomorphically evaluate functions of arbitrary complexity using one single set of parameters, then we need a procedure to lower the noise without decryption process. 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. The result is a new ciphertext which encrypts the same message while having lower noise when the depth of decryption function is small enough for evaluation.

The decryption function of existing HE schemes consists of modulo or division operation which is not conveniently supported by the scheme itself. Hence, in order to perform bootstrapping, one either needs to find another representation of modulo or division, or design some scheme which can handle its decryption circuit more comfortably. Among the best works on bootstrapping implementations, the work of Halevi and Shoup [HS15], 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 bootstrap a vector of size 1024 over $GF(2^{16})$ within 5 minutes.

Another approach for bootstrapping is using approximate homomorphic encryption scheme [CKKS17]. This scheme supports operations between encrypted data approximately, and this scheme can be used for evaluation of encrypted complex numbers. By using complex number as plaintext, we use various analytic tools to simplify modulo operations. In [CHK$^+$18], they convert modulo operation to sine function with the condition that input is close to some integer. This condition can be satisfied in most of the HE scheme, we just need to start bootstrapping little bit early. In this work, they reports that their bootstrapping takes only 2 minutes to bootstrapping a vector of size 128 over $\mathbb{C}$.

Despite of various works, bootstrapping procedure is considered as a very expensive operation. In case of bootstrapping for BGV scheme, when the plaintext modulus reaches 2^8, bootstrapping still takes a few hours to perform. The reason is mainly due to a digit extraction procedure, whose cost grows significantly with the plaintext modulus. The Fan-Vercauteran (FV) scheme, a scale-invariant variant of BGV, has also been implement in [LP16, AMBG$^+$16] and used in applications. We are not aware of any previous implementation of bootstrapping for FV. Furthermore, homomorphic linear transformation which is used to bootstrap multiple number of plaintext is also bottleneck for efficient bootstrapping. In [CHK$^+$18], the linear transformation part takes 456 seconds with overall timing 524 seconds.

In this paper, we solve those problems in bootstrapping. First, faster digit extraction algorithm in encrypted state is proposed and this is used for improve bootstrapping in both BGV and FV schemes with implementation of bootstrapping in FV scheme. Second, we improve linear transformation part of bootstrapping by understanding it as variant of discrete Fourier transformation (DFT) and convert it in sense of fast Fourier transformation.

1.2 Machine Learning on Encrypted Data

Suppose multiple financial institutions want to predict credit scores of their customers. Although each institution could independently learn a prediction model using various machine learning techniques, they may be able to collectively learn a better model by considering all of their data together for training. However, it is risky in terms of data security to share financial data between institutions, being even illegal in many countries.

Homomorphic encryption (HE), an encryption scheme that allows arbitrary computations on encrypted data, can be used to solve this dilemma. Using HE, multiple institutions can share their data in an encrypted form and run machine learning algorithms on the encrypted data without ever decrypting. This HE-based approach is flexible in that the training computation can be delegated to any party (or even an *untrusted* third party) without revealing the training data (other than their own). This flexibility is desirable, as other approaches require additional assumptions and conditions that may not be realizable in practice.

Despite many advantages, however, HE has not been used for computation-intensive tasks such as machine learning (especially on the training phase), having been thought to be impractical due to its large computation overhead. Indeed, basic operations (e.g., addition or multiplication) on ciphertexts are several (i.e., three to seven) orders of magnitude slower than the corresponding operations on plaintexts even in the state-of-the-art [GHS12b, HS15, CKKS17, vDGHV10, BV14, BV11, Bra12, BGV12, LATV12, CLT14, DM15]. Moreover, some complex operations may cause additional overhead when they are reduced to a combination of basic operations.* For example, fractional number (e.g., fixed-point or floating-point)

*Most of HE schemes support only basic operations like addition and multiplication arithmetic operations on ciphertexts are quite expensive, as they involve bit-manipulation operations that are expressed as complex arithmetic circuits of a large depth.

In addition to the sheer amount of computation, the use of various complex operations, such as floating-point arithmetic and non-polynomial functions (e.g., sigmoid), makes it challenging to apply HE to machine learning algorithms. Indeed, HEs have been applied to machine learning algorithms only in non-realistic settings [GLN12, KSK$^+$18] where only small-size training datasets over a small number of features are considered; or, they have been applied only on the prediction phase [BLN14, BPTG15, GBDL$^+$16, LLH$^+$17, CdWM$^+$17, JVC18, BMMP17] where the amount of computation is much smaller than that of the training phase.

1.3 List of Papers

This thesis contains the results of the following papers.

- [CH18] Hao Chen, Kyoohyung Han: Homomorphic Lower Digits Removal and Improved FHE Bootstrapping. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Cham, 2018.

- [CHH18] Jung Hee Cheon, Kyoohyung Han, Minki Hhan. Faster Homomorphic DFT and Improved Bootstrapping for FHE. IACR Cryptology ePrint Archive, 2018.

as built-in, and require other operations to be represented in the form of a combination of the built-in operations.

- [CHK17] Jung Hee Cheon, Kyoohyung Han, and Duhyeong Kim. Faster Bootstrapping of FHE over the integers, IACR Cryptology ePrint Archive, 2017:79, 2017.
- [HHCP18] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Deajun Park. Efficient Logistic Regression on Large Encrypted Data. will be appeared at Innovative Applications of Artificial Intelligence Conference (IAAI), 2019.

Chapter 2

Background

2.1 Notation

$[n] = \{0, 1, \cdots, n-1\}$, and $\mathbb{Z}_n$ is treated as $[n]$ in this paper. $a \bmod p$ for $a \in \mathbb{R}$ is a unique number $\in [0, p)$ such that $a - (a \bmod p)$ is an integer multiple of p. $\lfloor a \rceil$ is the nearest integer of a, and $[a]_p$ is a unique integer in $(-p/2, p/2]$ such that $a - [a]_p$ is a multiple of p. $a^{(t)}\langle r \rangle := a_k$ for a non-negative integer $a = \sum a_i t^i$ and $a_i \in [t]$. When $t = 2$, we omit the subscript t.

Column vectors are written by bold and lower case letters and matrices are written by bold and upper case letters. The entries of bold face is denoted as $\boldsymbol{v} = (v_0, v_1, \cdots, v_{n-1})^T$ and $\boldsymbol{M} = (M_{i,j})_{1 \leqslant i,j \leqslant n}$. We sometimes take modular n for indices of vector or matrices, and omit the transpose operator T. The entry-wise multiplication of two vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ is denoted by $\boldsymbol{v}_1 \odot \boldsymbol{v}_2$ which is called *Hadamard* multiplication. For the given vector $\boldsymbol{v}$ with length n, $\mathsf{diag}_i(\boldsymbol{v})$ is n by n matrix $\boldsymbol{M}$ such that $M_{j,j+i} = v_j$ for $0 \leqslant j < n$ and all other entries are zero. We will omit the index i of diag

when $i = 0$. On the other hand, for n by n matrix $\boldsymbol{M}$, $\mathsf{diag}_i(\boldsymbol{M})$ denotes a length n vector $(M_{0,i}, M_{1,1+i}, \cdots, M_{n-1,n-1+i})$. $\mathsf{rot}_i(\boldsymbol{v})$ is left shifted vector with index i, this means that the result vector is $(v_i, v_{i+1}, \ldots, v_{i-1})$. When the index i is negative, it means right shifting with index $-i$.

We sometimes use the special order of indices called *bit-reversal* order. It is defined by ordering the indices in increasing order of the reverse of binary representations that are padded so that each of these binary representation has the same length. For example, bit-reversal order of the given array (a_0, a_1, a_2, a_3) is (a_0, a_2, a_1, a_3) (because bit-reversed index is follows: $(00_{(2)}, 10_{(2)}, 01_{(2)}, 11_{(2)}) = (0, 2, 1, 3)$).

2.2 Homomorphic Encryption

Fully homomorphic encryption (FHE) is an encryption scheme that allows arbitrary computation on ciphertexts without decryption. The first secure FHE (based on the hardness assumption of a plausible number-theoretic problem) was proposed by Gentry [Gen09]. He first constructed a scheme, so-called somewhat homomorphic encryption (SHE), that allows a limited number of addition and multiplication operations.* A notable aspect of the scheme is the addition of a random noise for each encryption. To address the limitation of computation, he proposed the so-called *bootstrapping* procedure that converts a ciphertext with large noise into another ciphertext with the same plaintext but small noise. Using the bootstrapping, he constructed an FHE scheme on top of the SHE scheme.

Various FHE schemes have been proposed since Gentry's construction. Their message spaces are either $\mathbb{Z}_p$ or a vector space over $\mathbb{Z}_p$. In a bit-wise

*Note that an arbitrary computation can be composed of addition and multiplication on $\mathbb{Z}_2$.

FHE ($p = 2$), bit-manipulation and bootstrapping are efficient, but integer arithmetic is not. In a word-wise FHE ($p \gg 2$), however, the integer arithmetic is efficient as long as the result is smaller than p. Recently, an approximate FHE scheme has been proposed by [CKKS17]. The scheme, called HeaAn, supports efficient approximate computation. In addition to addition and multiplication, it supports a rounding operation, called *rescaling*, that is essential for approximate real arithmetic (e.g., floating-point arithmetic).

2.3 Ring Learning with Errors

The ring learning with errors (RLWE) problem was firstly introduced by Lyubaskevsky, Peikert and Regev [LPR10]. The definition of this problem is as follows:

Definition 2.3.1 (RLWE, definition 4 in [BGV12])**.** FOr security parameter λ, let $f(x) = x^d + 1$ where $d = d(\lambda)$ is a power of 2. Let $q = q(\lambda) \geqslant 2$ be an integer. Let $\mathcal{R} = \mathbb{Z}[x]/(f(x))$ and let $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$. Let $\chi = \chi(\lambda)$ be a distribution over $\mathcal{R}$. The $\mathsf{RLWE}_{d,q,\chi}$ problem is to distinguish the following two distributions: In the first distribution, one samples (a_i, b_i) uniformly from $\mathcal{R}_q^2$. In the second distribution, one first draws $s \leftarrow \mathcal{R}_q$ uniformly and then samples $(a_i, b_i) \in \mathcal{R}_q^2$ by sampling $a_i \leftarrow \mathcal{R}_q$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = a_i \cdot s + e_i$. The $\mathsf{RLWE}_{d,q,\chi}$ assumption is that the $\mathsf{RLWE}_{d,q,\chi}$ problem is infeasible.

Usually, the noise distribution χ is used as discrete Gaussian distribution. This Gaussian distribution might need to be "ellipsoidal" for certain reductions [LPR10]. And, secret key is sampled in χ instead of $\mathcal{R}_q$ uniformly.

For efficiency of bootstrapping and HE scheme, secret key is samples as sparse binary which has small number of non-zero coefficients in $\{-1, 1\}$.

2.4 Approximate GCD

The approximate greatest common divisor (AGCD) problem was firstly introduced in [vDGHV10]. Computing GCD of two value can be done easily using Euclidean algorithm. when the given values has noise, the hardness of the problem grows. The definition of the problem is follows:

Definition 2.4.1 (Approximate GCD)**.** For a odd positive integer p, we define the following distribution over γ-bit integers:

$$\mathcal{D}_{\gamma,\rho}(p) := \{q \leftarrow \mathbb{Z} \cap [0, 2^{\gamma}/p), r \leftarrow \mathbb{Z} \cap (-2^{\rho}, 2^{\rho}) \ : \ \textsf{output}\ x = pq + r\}$$

The $(\rho, \eta, \gamma)-$approximate GCD problem is: given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$ for a randomly chose η-bit odd integer p, output p.

Based on the hardness of this problem, various homomorhic encryption schemes were proposed [vDGHV10, CCK$^+$13, NK15]. For more faster homomorphic operations, a variant of this problem is also used in [CLT14].

Definition 2.4.2 (Variant of AGCD)**.** For a odd positive integer p, we define the following distribution over γ-bit integers:

$$\mathcal{D}_{\gamma,\rho}(p) := \left\{q \leftarrow \mathbb{Z} \cap [0, 2^{\gamma}/p^2), r \leftarrow \mathbb{Z} \cap (-2^{\rho}, 2^{\rho}) \ : \ \textsf{output}\ x = p^2 q + r\right\}$$

The $(\rho, \eta, \gamma)-$approximate GCD problem is: given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$ for a randomly chose η-bit odd integer p, output p.

Remark 2.4.1. The difference is just using p^2 instead of p. Note that original AGCD problem has reduction between learning with error problem while the variant of AGCD does not.

Chapter 3

Lower Digit Removal and Improved Bootstrapping

In this chapter, we aim at improving the efficiency of bootstrapping under large prime power plaintext moduli. We used a family of low degree lowest-digit-removal polynomials to design an improved algorithm to remove v lowest base-p digits from integers modulo p^e. Our new algorithm has depth $v \log p + \log e$, 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 $h = ||s||_1$ denote the 1-norm of the secret key, and assume 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 [LP16]. 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 vectors where each slot only holds a value in $\mathbb{Z}_{p^r}$ instead of a degree-d extension ring. The slim mode might be more applicable in some 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 full mode.

3.1 Basis of BGV and BFV scheme

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 denote its degree. 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 message is a polynomial $m(x)$ in R_t, and the secret key s is an element of R_q. In practice, s 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. Following is relation between c_0 and c_1 for each scheme:

BGV: $c_0 = -c_1 s + te + m$ for $c_1 \leftarrow R_q$ and noise $e \leftarrow \mathcal{DG}(\sigma)$,

FV: $c_0 = -c_1 s + \frac{q}{t} m + e$ for $c_1 \leftarrow R_q$ and noise $e \leftarrow \mathcal{DG}(\sigma)$.

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

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

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

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

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

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 [SV14] 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 co-prime. 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 $f(x)$ 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 coefficient-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 distinguish 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 strictly 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 co-prime to t. For simplicity, suppose $q \equiv q' \equiv 1(\mod t)$. Then c'_i equals 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 co-prime 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 3.1.1. *Suppose $c_0 + c_1 s = \Delta m + v + aq$ is a ciphertext in FV with $|v| < \Delta/4$. if $q' > 4t(1 + ||s||_1)$, and (c_0', c_1') is the ciphertext after switching the modulus 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

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

for some integer r. Taking nearest integers of the coefficients on the left hand side, we arrive at

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

with the rounding error $||\delta|| \leqslant t/q'(1 + ||s||_1)$. 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

$$q' > 4t(1 + ||s||_1).$$

□

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.

3.2 Improved Digit Extraction Algorithm

The previous method for digit extraction used certain lifting polynomials with good properties. 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 [HS15]. 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 $i, j \geqslant 0$, 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 $z_{i,j} \equiv z_i \mod p^{j+1}$.

Previous Method

The bootstrapping procedure in [HS15] 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, we need the following functionality.

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

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

We say this functionality "removes " the v lowest significant digits in base p from an e-digits integer. To realize the above functionality over homomorphically encrypted data, the authors in [HS15] constructed some special polynomials $F_e(\cdot)$ with the following lifting property.

Lemma 3.2.1 (Corollary 5.5 in [HS15]). *For every prime p and $e \geqslant 1$ there exist a degree p-polynomial F_e such that for every integer z_0, z_1 with $z_0 \in [p]$ and every $1 \leqslant e' \leqslant e$ 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 [HS15] and can be visualized in the following diagram.

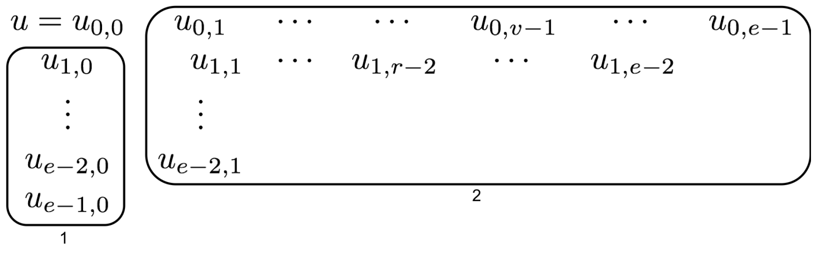
$u = u_{0,0}$
$u_{0,1}$ $\cdots$ $\cdots$ $u_{0,v-1}$ $\cdots$ $u_{0,e-1}$
$u_{1,0}$
$u_{1,1}$ $\cdots$ $u_{1,r-2}$ $\cdots$ $u_{1,e-2}$
$\vdots$
$u_{e-2,0}$
$u_{e-2,1}$
$u_{e-1,0}$
2
1

Figure 3.1: Previous Method for Digit Extraction

In Figure 3.1, the top-left digit is the input. This algorithm starts with the top row. From left to right, it recursively applies the lifting polynomial to obtain all the digits in area 2. Then the digits in area 1 on the next row can be obtained from subtracting all digits in area 2 on the same diagonal from the input and then dividing by p. When this procedure concludes, the (i, j)-th digit of the diagram will be $u_{i,j}$. In particular, digits on the final diagonal will be $u_{i,e-1-i}$. Then we can compute

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

New Method

We first stress that in the above method, 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 $(u_i, \underbrace{0, 0, \ldots, 0}_{e-i-1})$, 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. Fortunately, there is a polynomial of lower degree with the same functionality, as shown in the following lemma.

Lemma 3.2.2. *Let p be a prime and $e \geqslant 1$. Then there exists a polynomial f of degree at most $(e-1)(p-1)+1$ such that for every integer $0 \leqslant x < p^e$, we have*

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

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

Proof. We complete the proof sketch in [Gri17] 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

$$\hat{f}(x) = p \sum_{j=1}^{\infty} F_{j \cdot p}(x) = \sum_{m=p}^{\infty} a(m) \binom{x}{m}. \tag{3.2.1}$$

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 3.2.1,

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

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 $f(x+(p-1)/2)$. □

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 in the balanced base-p representation by $G_{e,p}(x)$ (or $G_e(x)$ if p is clear from context). In other words, if $x \in \mathbb{Z}_{p^e}$ and $x \equiv x_0 \mod p$ with $|x_0| \leqslant p/2$, then $G_e(x) = x_0 \mod p^e$.

Example 3.2.1. *When $e = 2$, we have $f(x) = -x(x-1)\cdots(x-p+1)$ and $G_2(x) = x - f(x+(p-1)/2)$.*

We recall that in the 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 $(e-i-1)(p-1)+1$. As a result, by combining the lifting polynomials and lowest digit removing polynomials, we can make the digit extraction algorithm faster with lower depth.

Figure 3.2 illustrates how our new digit removal algorithm works. First, each digit in area 2 is obtained by evaluating a lifting polynomial to the entry on its left. Then, the digits in area 3 on each row are obtained by evaluating the remaining lowest digit polynomial to the left-most digit on its row. Digits in area 1 are obtained by subtracting all the digits in area 2 on the same diagonal from the input, and dividing by p. Finally, in order to remove the v lowest digits, we subtract all the digits in area 3 from the input.

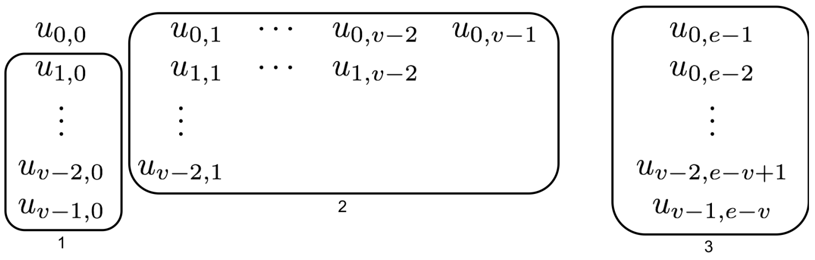

Figure 3.2: New Digit Extraction

We remark that the major difference of this procedure is that we only need to populate the top left triangle of side length v, plus the right most v-by-1 diagonal, where as the previous method needs to populate the entire triangle of side length e.

Moreover, the digits in area 3 (in Figure 3.2) has lower depth: in the 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 element of digit in area 1. However, in our method, its degree is only $(p-1)(e-i-1)+1$ on top of the i-th element of digit in area 1, which has degree at most p^i. The total degree of the algorithm is bounded by the maximum degree over

Algorithm 1: Removing v lowest digits from $x \in \mathbb{Z}_{p^e}$

```
Data: $x \in \mathbb{Z}_{p^e}$
Result: $x - [x]_{p^v} \bmod p^e$
// $F_i(x)$ : lifting polynomial with $F_i(x + O(p^i)) = x + O(p^{i+1})$
// $G_i(x)$ : lowest digit retain polynomial with $G_i(x) = [x]_p \bmod p^i$
1  Find largest $\ell$ such that $p^\ell < (p-1)(e-1)+1$;
2  Initialize res $= x$;
3  for $i \in [0, v)$ do
       // evaluate lowest digit retain polynomial
4      $R_i = G_{e-i}(x')$ ;                         // $R_i = x_i \bmod p^{e-i}$
5      $R_i = R_i \cdot p^i$ ;                        // $R_i = x_i p^i \bmod p^e$
6      if $i < v-1$ then
7          $L_{i,0} = F_1(x')$ ;                      // evaluate lifting polynomial
8      end
9      for $j \in [0, \ell-2)$ do
10         if $i + j < v-1$ then
11             $L_{i,j+1} = F_{j+2}(L_{i,j})$
12         end
13     end
14     if $i < v-1$ then
15         $x' = x$;
16         for $j \in [0, i+1)$ do
17             if $i - j > \ell - 2$ then
18                 $x' = x' - R_j$
19             end
20             else
21                 $x' = x' - L_{j,i-j}$
22             end
23         end
24     end
25     res $=$ res $- R_i$;
26 end
27 return res;
```

all the digits in area 3, that is

$$\max_{0 \leqslant i < r} \left(p^i \cdot ((e-1-i) \cdot (p-1) + 1) \right).$$

Since each individual term is bounded by $e \cdot p^v$, the total degree of the procedure is at most $e \cdot p^v$. This is lower than p^{e-1} in the previous method when $v \leqslant e-2$ and $p > e$.

Comparison.

We discuss one further optimization to remove v 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 the G_i polynomial (or, the red digit) to obtain the green digit in the next row. This saves some work and also lowers the depth of the overall procedure. This optimization is incorporated into Algorithm 1.

The depth and computation cost of Algorithm 1 is summarized in Theorem 3.2.1. The depth 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 scalar multiplication is much faster than FHE schemes than ciphertext multiplication, we choose to measure the computational cost by the number of ciphertext multiplications. The Paterson-Stockmeyer algorithm [PS73] evaluates a polynomial of degree d with $\sim \sqrt{2d}$ non-constant multiplications, and we use that as the base of our estimate.

Theorem 3.2.1. *Algorithm 1 is correct. Its depth is bounded above by*

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

The number of non-constant multiplications is asymptotically equal to $\sqrt{2pev}$.

Table 3.1 compares the asymptotic depth and number of non-constant multiplications between our method for digit removal and the method of

Method	Depth	No. ciphertext multiplications
[HS15]	$e\log(p)$	$\frac{1}{2}e^2\sqrt{2p}$
This work	$v\log(p)+\log(e)$	$\sqrt{2pev}$

Table 3.1: Complexity of $\mathsf{DigitRemove}(p, e, v)$

[HS15]. From the table, we see that the advantage of our method grows with the difference $e - v$. In the bootstrapping scenario, we have $e - v = r$, the exponent of the plaintext modulus. Hence, our algorithm compares favorably for larger values of r.

3.3 Bootstrapping for BGV and BFV Scheme

The bootstrapping for FV scheme follows the main steps from [HS15] for the BGV scheme, while we make two modifications in modulus switching and digit extraction. First, we review the procedure in [HS15].

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 $t' = p^e$. 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. 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 $C_0, \ldots, C_{d-1}$, 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 $|e_i| \leqslant \frac{p^{e-r}}{2}$ for each i, we will apply Algorithm 1 to remove the lower digits e_i, resulting in d new ciphertexts encrypting Δm_i for $0 \leqslant i < n$ 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 $m(x)$.

3.3.1 Our modifications

BFV scheme

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' co-prime 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 $\mathsf{DigitRemove}(p, e, e - r)$.

BGV scheme

To apply our ideas to the digit extraction step in BGV bootstrapping, we simply replace the algorithm in [HS15] with our digit removal Algorithm 1.

Comparison

The major difference in the complexities of bootstrapping between the two schemes comes from the parameter e. In case of BFV, we can choose (roughly) $e = r + \log_p(||s||_1))$. On the other hand, the estimate of e for correct bootstrapping in [HS15] for the BGV scheme is

$$e \geqslant 2r + \log_p(||s||_1).$$

We can analyze the impact of this difference on the depth of digit removal, and therefore on the depth of bootstrapping. Setting $v = e - r$ in Theorem 3.2.1, the depth for the BGV case is

$$(r + \log_p(||s||_1) \log p + \log(2r + \log_p(||s||_1)).$$

Substituting $r = \log_p(t)$ into the above formula and throwing away lower order terms, we obtain the improved depth for the digit extraction in step

BGV bootstrapping as

$$\log t + \log(||s||_1) + \log(\log_p(t^2 \cdot ||s||_1)) \approx \log t + \log(||s||_1).$$

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

$$\log(||s||_1) + \log(\log_p(t \cdot ||s||_1)).$$

which only scales with $\log \log t$. This is smaller than BGV in the large plaintext modulus regime.

We can also compare the number of ciphertext multiplications needed for the digit extraction procedures. Replacing v with $e - r$ in the second formula in Theorem 3.2.1 and letting $e = 2r + \log_p(||s||_1)$ for BGV (resp. $e = r + \log_p(||s||_1)$ for BFV), we see that the number of ciphertext multiplications for BGV is asymptotically equal to

$$\frac{\sqrt{2p}}{(\log p)^{3/2}} \cdot (2\log(t) + \log(||s||_1))^{1/2} \cdot (\log(t) + \log(||s||_1)).$$

In the BFV case, the number of ciphertext multiplications is asymptotically equal to

$$\frac{\sqrt{2p}}{(\log p)^{3/2}} (\log(t) + \log(||s||_1))^{1/2} \log(||s||_1)).$$

Hence when t is large, the digit extraction procedure in bootstrapping requires less work for BFV than BGV.

For completeness, we also analyze the original digit extraction method in BGV bootstrapping. Recall that the previous algorithm has depth $(e - 1)\log p$, and takes about $\frac{1}{2}e^2$ homomorphic evaluations of polynomials of

Method	Depth	No. ciphertext multiplications
[HS15] (BGV)	$2\log(t)+\log(h)$	$\frac{\sqrt{2p}}{2(\log p)^2}(2\log(t)+\log(h))^2$
This work (BGV)	$\log(t)+\log(h)$	$\frac{\sqrt{2p}}{(\log p)^{3/2}}(2\log(t)+\log(h))^{1/2}(\log(t)+\log(h))$
This work (FV)	$\log\log(t)+\log(h)$	$\frac{\sqrt{2p}}{(\log p)^{3/2}}(\log(t)+\log(h))^{1/2}\log(h)$

Table 3.2: Asymptotic complexity of digit extraction step in bootstrapping. Here $h = ||s||_1$ is the 1-norm of the secret key, and $t = p^r$ is the plaintext modulus.

degree p. If we use the Paterson-Stockmeyer method for polynomial evaluation, then the total amount of ciphertext multiplications is roughly $\frac{1}{2}e^2\sqrt{2p}$. Plugging in the lower bound $e \geqslant 2r + \log_p(||s||_1)$, we obtain an estimate of depth and work needed for the digit extraction step in the original BGV bootstrapping method in [HS15]. Table 3.2 summarizes the cost for three different methods.

Fixing p and h in the last column of Table 3.2, we can see how the number of multiplications grows with $\log t$. The method in [HS15] scales by $(\log t)^2$, while our new method for BGV improves it to $(\log t)^{3/2}$. In the FV case, the number of multiplications scales by only $(\log t)^{1/2}$.

Remark 3.3.1. As another advantage of our revised BGV bootstrapping, we make a remark on security. From Table 3.2, we see that in order for bootstrapping to be more efficient, it is advantageous to use a secret key with smaller 1-norm. For this reason, both [HS15] and this work choose to use a sparse secret key, and a recent work [APS15] shows that sparseness can be exploited in the attacks. To resolve this, note that it is easy to keep the security level in our situation: since our method reduces the overall depth for the large plaintext modulus case, we could use a smaller modulus

q, which increases the security back to a desired level.

3.4 Slim Bootstrapping Algorithm

The bootstrapping algorithm for BFV and BGV is expensive also due to the d repetitions of digit extraction. For some parameters, the extension degree d can be large. However, many interesting applications requires arithmetic over $\mathbb{Z}_{p^r}$ rather than its 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 $\mathbb{Z}_t^k$, 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 full bootstrapping algorithm. Also, we need to revise the linear transformation and inverse linear transformation slightly. We give an outline of our slim bootstrapping algorithm below.

Inverse Linear Transformation. We take as input a batch encryption of $(m_1 \ldots, m_k) \in \mathbb{Z}_{p^r}^k$. In the first step, we apply an "inverse" linear transformation to obtain an encryption of $m_1 + m_2x^d + \ldots + m_kx^{d(k-1)}$. This can be done using k slot permutations and k plaintext multiplications.

Modulus Switching and Dot product. These two steps are exactly the same as the full bootstrapping procedure. After these steps, we obtain

$$\begin{tikzcd}
\mathsf{Enc}(m_0, m_1, m_2, \cdots, m_{k-1}) \arrow[d, "\mathsf{InverseLinearTransformation}"] \\
\mathsf{Enc}(m_0 + m_1 x^d + \cdots + m_{k-1} x^{d(k-1)}) \arrow[d, "\mathsf{Modulus\ Switching\ and\ Dot\ Product}"] \\
\mathsf{Enc}(m(x) \cdot p^{e-r} + e(x)) \arrow[d, "\mathsf{LinearTransformation}"] \\
\mathsf{Enc}(m_0 \cdot p^{e-r} + e_0, \cdots, m_{k-1} \cdot p^{e-r} + e_{k-1}) \arrow[d, "\mathsf{Digit\ Extraction}"] \\
\mathsf{Enc}(m_0, m_1, m_2, \cdots, m_{k-1})
\end{tikzcd}$$

Figure 3.3: slim bootstrapping

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 scalar multiplications to obtain a batch encryption of $(\Delta m_1 + v_1, \ldots, \Delta m_k + v_k)$. Details of this step can be found in the below.

Digit extraction. Then, we apply digit-removal algorithm to remove the noise 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 slim bootstrapping process.

Optimizing Linear Transformation for Slim Bootstrapping

In our slim mode of bootstrapping, we used a linear transform which has the following property: the input is an encryption of $\sum m_i x^i$, and 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 scalar 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 scalar multiplications to k.

Coefficient Selection. The first part of the optimized linear transform functionality can be viewed as a coefficient selection. This process gets input $\mathsf{Enc}(m(x))$ and outputs $\mathsf{Enc}(m'(x))$ with $m'(x) = \sum_{i=0}^{n/d-1} m_{id} \cdot x^{id}$. In other words, it selects the coefficients of $m(x)$ where the exponents of x are divisible by d. The following algorithm is specified to the case when n is a power of two . 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 construct a recursive algorithm which return $m'(x) = \sum_{i=0}^{n/d-1} m_{id} \cdot x^{id}$ for power of two d.

- For given $m(x)$, First compute $m_0(x) = m(x) + \phi_0(m(x))$.
- Recursively compute $m_i(x) = \phi_i(m_{i-1}(x)) + m_{i-1}(x)$ for $1 \leqslant i \leqslant \log_2 d$.

- Return $m'(x) = d^{-1} \cdot m_{\log_2 d} \bmod t$ for plain modulus t.

The function $\phi_i : X \to X^{\frac{n+2^i}{2^i}}$ can be evaluated homomorphically by using the same technique used in slot permutation. Another operation is just multiplying by $d^{-1} \bmod t$. Hence we can obtain $\mathsf{Enc}(m'(x))$. This process needs $\log d$ slot permutations and additions.

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 λ_i are pre-computed polynomials 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 unity. Therefore, this step only requires k slot permutations and k plaintext multiplications. We can also adapt the babystep-giantstep method to reduce the number of slot permutations to $O(\sqrt{k})$, and we omit further details.

3.5 Implementation Result

We implemented both the full mode and the slim mode of bootstrapping for BFV in 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 co-prime to m in order to use batching, we can not use $p = 2$ for SEAL bootstrapping. Instead, 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 levels using Martin Albrecht's LWE estimator [APS15].

Digit Extraction

We implemented Algorithm 1 in HElib and compared with the results of the original HElib implementation for removing v digits from e digits. From Table 3.3, we see that for $e \geqslant v+2$ and large p, 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.

	[HS15]		Our Method	
(p, e, v)	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 3.3: Comparison of digit removal algorithms in HElib (Toshiba Portege Z30t-C laptop with 2.6GHz CPU and 8GB memory)

When $p = 2$ and r, e are small, the current HElib implementation can be faster due to the fact that the lifting polynomial is $F_e(x) = x^2$ and squaring operation is faster than generic multiplication. Also, when $e = v + 1$, i.e., the task is to remove all digits except the highest one, our digit removal method has similar performance as the HElib counterpart.

Bootstrapping for BFV scheme

Table 3.4 and 3.5 present timing results for the full and slim modes of bootstrapping for BFV implemented in SEAL. In both tables, 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). Here $\mathsf{R}(p^r, d)$ denotes a finite ring with degree d over base ring $\mathbb{Z}_{p^r}$, and $\mathsf{GF}(p^r)$ denotes the finite field with p^r elements.

Comparing the corresponding entries from Table 3.4 and 3.5, we see that the slim mode of bootstrapping is either close to or more than d times faster than the full mode.

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

Table 3.4: Time table for bootstrapping for BFV 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 Parameter	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 3.5: Time table for slim bootstrapping for BFV scheme, hw=128 (Intel(R) Core(TM) i7-4770 CPU with 3.4GHZ CPU and 32GB memory)

Chapter 4

Faster Homomorphic DFT and Improved Bootstrapping

In this chapter, we study the fast linear transformations for special structured matrices. First, we propose a new way to evaluate discrete Fourier transformation for a given packed ciphertext. Our method only needs $O(\log n)$ number of homomorphic operations while the previous method requires $O(\sqrt{n})$ rotations and $O(n)$ constant vector multiplications for n the length of input vector.

We factorize the DFT matrix into $\log_2 n$ sparse block diagonal matrices using the Cooley-Tukey factorization with radix 2. We observe that each factor has only three diagonal vectors, and each $\log_2 k$ consecutive multiplication of those factors has $(2k-1)$ diagonal vectors. Therefore, homomorphic DFT evaluation is converted to $\log_k n$ number of homomorphic matrix multiplications for matrix with $(2k-1)$ diagonal vectors for an arbitrary integer k dividing n.

From SIMD operation of HE schemes, evaluating matrices with d diagonal vector in encrypted state can be done with $O(\sqrt{d})$ homomorphic rotations and d homomorphic constant vector multiplications using the baby-step algorithm. So, we obtain a homomorphic DFT algorithm which needs $O(\sqrt{k}\log n)$ number of homomorphic rotations and $O(k\log n)$ number of homomorphic constant vector multiplications with $O(\log_k n)$ constant vector multiplication depth. In addition, we can obtain a trade-off between depth and complexity by adjusting k.

Second, we apply the same matrix decomposition strategy into sparse diagonal matrices to improve the linear transformations in bootstrapping for HeaAn. We decompose corresponding matrices recursively, similarly to the Cooley-Tukey algorithm. As a result we obtain the same improvement in the linear transformations in bootstrapping: $O(\sqrt{k}\log n)$ homomorphic rotations and $O(k\log n)$ homomorphic constant vector multiplications with $O(\log_k n)$ constant vector multiplication depth for plaintext vector length n.

We also implement our method using the approximate homomorphic encryption library [snu18] to show the improvements. Our implementation shows that the homomorphic DFT with length 2^{13} only takes about 8 seconds when $k = 2$. This results shows a more than $150\times$ performance improvement compared to previous works on homomorphic DFT (or FFT) [CKKS17, CSV17, CSVW16]. On the other hand, the bootstrapping procedure for HeaAn using our linear transformation algorithm only takes 2 minutes for $\mathbb{C}^{32768}$ plaintext space with 8-bit precision. This result yields an amortized rate per bits of 0.45ms, less than one millisecond. The previous algorithm takes 26 hours in the same setting, which is only realistic for a small number of slots.

4.1 Basis of HEAAN scheme

In our chapter, we will focus on the DFT on complex field. For this reason, we need homomorphic encryption for complex arithmetic. At 2017, homomorphic encryption scheme for approximate number arithmetic is proposed by Cheon et al. [CKKS17] which is called HeaAn. The ciphertext is $(c_0, c_1) \in \mathcal{R}_q$ as in BGV and FV schemes with following relation:

$$c_0 = -c_1 s + \lfloor \Delta \cdot m \rceil + e \text{ for } m \in \mathbb{R}[X], e \leftarrow \mathcal{DG}(\sigma).$$

The plaintext structure of this scheme is $\mathbb{C}^{N/2}$ for polynomial ring dimension N, and it is suitable for our purpose. More precisely, encoded polynomial $m(x) = \sum_{i=0}^{N-1} f_i X^i$ for given plaintext $\boldsymbol{m}$ can be computed as follows: $\boldsymbol{f} = (f_i)_{0 \leqslant i < N} = \frac{1}{N}(\overline{U}^T \cdot \boldsymbol{m} + U^T \cdot \overline{\boldsymbol{m}})$. For given polynomial $m(x)$, decoded vector $\boldsymbol{m}$ can be compute as follows: $U \cdot \boldsymbol{f}$ such that $\boldsymbol{f} = (f_i)_{0 \leqslant i < N}$ for $m(x) = \sum_{i=0}^{N-1} f_i X^i$. Following is description of the matrix $\boldsymbol{U}$:

$$\boldsymbol{U} = \begin{bmatrix} 1 & w_0 & w_0^2 & \cdots & w_0^{N-1} \\ 1 & w_1 & w_1^2 & \cdots & w_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_{N/2-1} & w_{N/2-1}^2 & \cdots & w_{N/2-1}^{N-1} \end{bmatrix}.$$

This scheme supports various kinds of homomorphic operations. Let ct_i is encrypted of $\boldsymbol{m}_i \in \mathbb{C}^{N/2}$ for $i = 0, 1$. Functions in below are homomorphic operations in the scheme.

- $\mathsf{encrypt}(\boldsymbol{m}; \Delta)$: return encryption of $\boldsymbol{m}$ using scaling factor Δ.
- $\mathsf{add}(\mathsf{ct}_0, \mathsf{ct}_1)$: return encryption of $\boldsymbol{m}_0 + \boldsymbol{m}_1$.

- $\mathsf{constAdd}(\boldsymbol{m}, \mathsf{ct}_0)$: return encryption of $\boldsymbol{m} + \boldsymbol{m}_0$.
- $\mathsf{mult}(\mathsf{ct}_0, \mathsf{ct}_1)$: return encryption of $\boldsymbol{m}_0 \odot \boldsymbol{m}_1$.
- $\mathsf{constMult}(\boldsymbol{m}, \mathsf{ct}_0)$: return encryption of $\boldsymbol{m} \odot \boldsymbol{m}_0$.
- $\mathsf{leftRotate}(\mathsf{ct}_0, \mathsf{idx})$: return encryption of $\boldsymbol{m'} = \mathsf{rot}_i(\boldsymbol{m_0})$.
- $\mathsf{rightRotate}(\mathsf{ct}_0, \mathsf{idx})$: return encryption of $\boldsymbol{m'} = \mathsf{rot}_{-i}(\boldsymbol{m_0})$.

4.2 Homomorphic DFT

In this section, we briefly review the previous approach to evaluate DFT with homomorphic encryption (HE) and describe our new homomorphic DFT algorithm. We propose new homomorphic DFT algorithm and also hybrid algorithm that combines our new method with previous approach.

4.2.1 Previous Approach

In [HS15], they proposed faster linear transformation ($\simeq$ NTT) for bootstrapping when the input size of $\phi(m)$ (here m is product of co-prime integers m_i). They understand one variable polynomial ring as multivariate with special basis which is called *powerful basis.* This approach shows that DFT with dimension m can be split to several number of DFT with m_i for co-prime m_is.

On the other hand, in the case of power of prime dimension, there is no specialized algorithm for homomorphic DFT. Previously known approaches apply a general homomorphic linear transform with DFT matrix to the ciphertext [CKKS17, CH18]. We review the key ideas of these approaches.

HE schemes support Hadamard multiplication and rotation for the plaintext vector. The following equation shows a representation of matrix-vector multiplication via Hadamard multiplications and rotations.

$$\begin{aligned}
\boldsymbol{M} \cdot \boldsymbol{v} &= \sum_{i=0}^{n} \mathsf{diag}_i(\boldsymbol{M}) \odot \mathsf{rot}_i(\boldsymbol{v}) \\
&= \sum_{i=0}^{\ell} \sum_{j=0}^{k} \mathsf{diag}_{ki+j}(\boldsymbol{M}) \odot \mathsf{rot}_{ki+j}(\boldsymbol{v}) \\
&= \sum_{i=0}^{\ell} \mathsf{rot}_{ki} \left(\sum_{j=0}^{k} \mathsf{rot}_{-ki}(\mathsf{diag}_{ki+j}(\boldsymbol{M})) \odot \mathsf{rot}_j(\boldsymbol{v}) \right)
\end{aligned}$$

The first line gives a simple way to compute homomorphic matrix-vector multiplication which requires $O(n)$ rotations and Hadamard multiplications. Based on the third line of the equation, we can achieve an algorithm so-called baby-step giant-step (BSGS) to matrix-vector multiplication with $O(\sqrt{n})$ rotations of ciphertexts.

4.2.2 Our method

Now we will introduce our method for fast homomorphic DFT. In this section, we will mainly consider DFT with bit-reversed output $\mathrm{DFT}_n^{\mathsf{NR}}$ and its inverse (the letter NR stands for *normal to reversal*). In addition, we will describe the method to extend our method to input bit-reversed case and its inverse in the last part of this section. We focus on the power-of-two dimension case while our method can be generalized to other power of prime dimensions, because power-of-two cases are appropriate to our applications and, moreover, easy to describe.

The starting point of our method is to observe that the multiplication between matrix and encrypted vector can be much faster when the matrix only has the small number of non-zero $\{\mathsf{diag}_i(\boldsymbol{M})\}_{0\leqslant i<n}$. Bit-reversed order DFT matrix can be decomposed to sparse matrices, and this property is used to fasten the discrete Fourier transform. Our observation is that those sparse matrices have small number of non-zero $\{\mathsf{diag}_i(\cdot)\}_{0\leqslant i<n}$ (exactly two or three non-zero vectors).

The DFT matrix factorization

Let $\mathbf{DFT}_n^{\mathsf{NR}}$ be a matrix corresponding to the DFT algorithm with input length n with bit-reversed output. The following equation shows that the matrix representation of recursive FFT Cooley-Tukey algorithm [CT65].

$$\mathbf{DFT}_n^{\mathsf{NR}} = \begin{bmatrix} \mathbf{DFT}_{n/2}^{\mathsf{NR}} & \mathbf{DFT}_{n/2}^{\mathsf{NR}} \\ \mathbf{DFT}_{n/2}^{\mathsf{NR}} \cdot \boldsymbol{W}_{n/2} & -\mathbf{DFT}_{n/2}^{\mathsf{NR}} \cdot \boldsymbol{W}_{n/2} \end{bmatrix} = \begin{bmatrix} \mathbf{DFT}_{n/2}^{\mathsf{NR}} & \mathbf{0} \\ \mathbf{0} & \mathbf{DFT}_{n/2}^{\mathsf{NR}} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{I}_{n/2} & \boldsymbol{I}_{n/2} \\ \boldsymbol{W}_{n/2} & -\boldsymbol{W}_{n/2} \end{bmatrix}$$

where the matrix $\boldsymbol{W}_{n/2} = \mathsf{diag}(1, \omega_n, \omega_n^2, \cdots, \omega_n^{n/2-1})$ and $\omega_n = e^{2\pi i/n}$. If we adapt this equation repeatedly, we can decompose the DFT matrix $\mathbf{DFT}_n^{\mathsf{NR}}$ to $\log_2 n$ number of matrices. The following matrix illustrates the specific form of matrices in the recursive formula:

$$\boldsymbol{D}_k^{(n)} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{I}_{n/k} & \boldsymbol{I}_{n/k} \\ \boldsymbol{W}_{n/k} & -\boldsymbol{W}_{n/k} \end{bmatrix} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & \begin{bmatrix} \boldsymbol{I}_{n/k} & \boldsymbol{I}_{n/k} \\ \boldsymbol{W}_{n/k} & -\boldsymbol{W}_{n/k} \end{bmatrix} \end{bmatrix} \in \mathbb{C}^{n\times n}. \qquad (4.2.1)$$

which has $k/2$ number diagonal blocks. The recursive equation above implies

$$\mathbf{DFT}_n^{\mathsf{NR}} = \boldsymbol{D}_n^{(n)} \cdot \boldsymbol{D}_{n/2}^{(n)} \cdot \cdots \cdot \boldsymbol{D}_2^{(n)}. \tag{4.2.2}$$

Remark 4.2.1. As noted above, decomposing DFT matrices into sparse diagonal matrices is possible for other power-of-prime cases and this induces a fast homomorphic DFT algorithm for power-of-prime dimension. This fact can be obtained by using general Cooley-Tukey algorithm.

Homomorphic DFT.

We recall the representation of matrix and vector multiplication via Hadamard multiplication and vector shifting:

$$\boldsymbol{M} \cdot \boldsymbol{v} = \sum_{i=0}^{n} \mathsf{diag}_i(\boldsymbol{M}) \odot \mathsf{rot}_i(\boldsymbol{v}).$$

The matrix-vector multiplication algorithm based on this form is especially efficient for the matrix $\boldsymbol{M}$ with only small number of non-zero diagonal vector. Namely, $\mathsf{diag}_i(\boldsymbol{M})$ is a non-zero vector only for small number of i's. We call this matrix by *sparse-diagonal* matrix. For sparse diagonal matrix $\boldsymbol{M}$, we don't need to compute $\mathsf{rot}_i(\boldsymbol{v})$ for those i's satisfying $\mathsf{diag}_i(\boldsymbol{M}) = \boldsymbol{0}$. Therefore, the required number of shifting in naive approach is at most the number of non-zero diagonal vectors that the matrix M has.

Lemma 4.2.1. $\mathsf{diag}_k(\boldsymbol{D}_{2^i}^{(n)})$ *is nonzero only for* $k = 0, \pm n/2^i$.

Proof. See Equantion (4.2.1). □ □

From Lemma 4.2.1, multiplication between matrix $\boldsymbol{D}_{2^i}^{(n)}$ and vector $\boldsymbol{v}$

can be represented as follows:

$$\begin{aligned}\boldsymbol{D}_{2^i}^{(n)} \cdot \boldsymbol{v} = \mathsf{diag}_0(\boldsymbol{D}_{2^i}^{(n)}) \odot \boldsymbol{v} \quad &+ \quad \mathsf{diag}_{n/2^i}(\boldsymbol{D}_{2^i}^{(n)}) \odot \mathsf{rot}_{n/2^i}(\boldsymbol{v}) \\ &+ \quad \mathsf{diag}_{n-n/2^i}(\boldsymbol{D}_{2^i}^{(n)}) \odot \mathsf{rot}_{-n/2^i}(\boldsymbol{v}).\end{aligned}$$

Therefore, $\mathrm{DFT}_n^{\mathsf{NR}} \cdot \boldsymbol{v}$ can be computed recursively as $\prod_{i=1}^{\log_2 n} \boldsymbol{D}_{2^i}^{(n)} \cdot \boldsymbol{v}$, where each multiplication can be done with $O(1)$ number of Hadamard multiplication and shifting. The overall number of operations is $O(\log n)$ homomorphic shiftings and Hadamard multiplications with constant plaintext vectors. The Algorithm 2 shows our homomorphic DFT algorithm in detail with notations in Section 4.1.

Algorithm 2: Homomorphic $\mathrm{DFT}_n^{\mathsf{NR}}$ algorithm

```
  Data: Ciphertext ctxt such that $\mathsf{Dec}(\mathsf{ctxt}, \mathsf{sk}) = \boldsymbol{m} \in \mathbb{C}^n$
1 for $1 \leqslant i \leqslant \log_2 n$ do
2     $\mathsf{ctxt}_0 \leftarrow \mathsf{constMult}(\mathsf{diag}_0(\boldsymbol{D}_{2^i}^{(n)}), \mathsf{ctxt})$;
3     $\mathsf{ctxt}_1 \leftarrow \mathsf{leftRotate}(\mathsf{ctxt}, n/2^i)$;
4     $\mathsf{ctxt}_2 \leftarrow \mathsf{rightRotate}(\mathsf{ctxt}, n/2^i)$;
5     $\mathsf{ctxt}_1 \leftarrow \mathsf{constMult}(\mathsf{diag}_{n/2^i}(\boldsymbol{D}_{2^i}^{(n)}), \mathsf{ctxt}_1)$;
6     $\mathsf{ctxt}_2 \leftarrow \mathsf{constMult}(\mathsf{diag}_{n-n/2^i}(\boldsymbol{D}_{2^i}^{(n)}), \mathsf{ctxt}_2)$;
7     $\mathsf{ctxt} \leftarrow \mathsf{add}(\mathsf{ctxt}_0, \mathsf{ctxt}_1)$;
8     $\mathsf{ctxt} \leftarrow \mathsf{add}(\mathsf{ctxt}, \mathsf{ctxt}_2)$;
9 end
```

In each loop of the Algorithm 2, there are two homomorphic rotations and three homomorphic constant vector multiplications. Furthermore, left rotation by $n/2$ and right rotation by $n/2$ is same. For this reason, we do not need to compute right and left rotations for $i = 1$ case. This will reduce one homomorphic rotation. As a result, our algorithm needs $(2\log_2 n - 1)$ number of homomorphic rotation and $(3\log_2 n)$ number of homomorphic constant vector multiplications.

Trade-off between depth and complexity.

While our method is fairly efficient with respect to the number of operations, the required depth with respect to constant multiplication is also increased by $O(\log_2 n)$. In this respect, we adapt additional parameter r which is called *radix* to generalize our method. Our generalized method gives trade-off between the number of steps and complexity of homomorphic DFT.

Assume that $\log_2 n$ is even and recall the matrix decomposition of DFT matrix:

$$\begin{aligned}\mathbf{DFT}_n^{\mathsf{NR}} &= \boldsymbol{D}_n^{(n)} \cdot \boldsymbol{D}_{n/2}^{(n)} \cdot \boldsymbol{D}_{n/4}^{(n)} \cdot \boldsymbol{D}_{n/8}^{(n)} \cdot \cdots \cdot \boldsymbol{D}_{16}^{(n)} \cdot \boldsymbol{D}_8^{(n)} \cdot \boldsymbol{D}_4^{(n)} \cdot \boldsymbol{D}_2^{(n)} \\ &= (\boldsymbol{D}_n^{(n)} \cdot \boldsymbol{D}_{n/2}^{(n)}) \cdot (\boldsymbol{D}_{n/4}^{(n)} \cdot \boldsymbol{D}_{n/8}^{(n)}) \cdots (\boldsymbol{D}_4^{(n)} \cdot \boldsymbol{D}_2^{(n)}).\end{aligned}$$

This equation give a factorization of DFT matrix into $\log_4 n$ number of matrices of the form

$$\boldsymbol{D}_i^{(n;4)} = (\boldsymbol{D}_{2^{2i}}^{(n)} \cdot \boldsymbol{D}_{2^{2i-1}}^{(n)}) \text{ for } 1 \leqslant i \leqslant \log_4 n.$$

We also can define similar term for $r = 2^k$ by

$$\boldsymbol{D}_j^{(n;r)} = \boldsymbol{D}_{r^j}^{(n)} \cdot \boldsymbol{D}_{r^j/2}^{(n)} \cdot \cdots \cdot \boldsymbol{D}_{r^{j-1}\cdot 2}^{(n)}$$

for $1 \leqslant j \leqslant \log_r n$ and $k | \log_2 n$. This factorization allows us to compute DFT in a new way. To analyze the efficiency, we observe some properties of these matrices.

Lemma 4.2.2. *The multiplication of i-th diagonal matrix and j-th diagonal matrix is $i+j$-th diagonal matrix. More precisely, the following equation holds*

$$\mathsf{diag}_i(\boldsymbol{a}) \cdot \mathsf{diag}_j(\boldsymbol{b}) = \mathsf{diag}_{i+j}(\boldsymbol{a} \odot \mathsf{rot}_i(\boldsymbol{b})).$$

Proof. Trivial. □

Lemma 4.2.3. *Let D_k be a multiplication of k consecutive matrices in Equation 4.2.2:*

$$\boldsymbol{D}_k = \boldsymbol{D}^{(n)}_{2^{s+k}} \cdot \boldsymbol{D}^{(n)}_{2^{s+k-1}} \cdot \cdots \cdot \boldsymbol{D}^{(n)}_{2^{s+1}}.$$

Then at most $2^{k+1}-1$ diagonals of D is nonzero vector. Further, the indices of nonzero diagonals form arithmetic progression.

Proof. Lemma 4.2.2 clearly holds. To show Lemma 4.2.3, we decompose $D^{(n)}_{2^t}$ into $\mathsf{diag}_{-n/2^t}(\boldsymbol{D}^{(n)}_{2^t}) + \mathsf{diag}_0(\boldsymbol{D}^{(n)}_{2^t}) + \mathsf{diag}_{n/2^t}(\boldsymbol{D}^{(n)}_{2^t})$ as in Lemma 4.2.1. By Lemma 4.2.2, the index of $\boldsymbol{D}_k$ that is non-zero is of the form

$$e_{s+1} \cdot \frac{n}{2^{s+1}} + e_{s+2} \cdot \frac{n}{2^{s+2}} + \cdots + e_{s+t} \cdot \frac{n}{2^{s+t}},$$

where $e_i \in \{-1, 0, 1\}$ for $s+1 \leqslant i \leqslant s+t$. These indices are multiple of $n/2^{s+t}$, and the absolute value of it is bounded by $\sum_{j=s+1}^{s+t} n/2^j = (2^t - 1)n/2^{s+t}$. □ □

According to Lemma 4.2.3, the number of nonzero diagonal of $\boldsymbol{D}^{(n;r)}_j$ is $2r-1$ for $j > 1$ and r for $j = 1$. Thus the required number of homomorphic multiplication and slot shifting to compute multiplication of encryption of $\boldsymbol{v}$ and $\boldsymbol{D}^{(n;r)}_j$ is less than $2r-1 = O(r)$ for radix r, respectively. By recursively multiplying $\boldsymbol{D}^{(n;r)}_j$ to $\boldsymbol{v}$, we obtain a new algorithm to compute homomorphic DFT which requires $O(r \log_r n)$ homomorphic rotations and constant vector multiplications while has $O(\log_r n)$ depth. Overall, we obtain depth-efficiency trade-off using larger radix. We note that we assumed that the used radix is a divisor of $\log_2 n$, but this condition can be removed by considering dynamic radices for each recursive step.

4.2.3 Hybrid method

An interesting observation in Lemma 4.2.3 is that the indices of $\boldsymbol{D}_j^{(n;r)}$ forms an arithmetic progression. We call this property *regular.* Here we show that this property yield a hybrid method of our homomorphic DFT algorithm and baby-step giant-step (BSGS) algorithm. To do this, we apply a BSGS matrix-vector multiplication method for sparse diagonal matrix $\boldsymbol{M}$ with arithmetic progression indices as follows:

$$\begin{aligned}\sum_{i=1}^{t} \boldsymbol{m}_i \odot \mathsf{rot}_{\ell i}(v) &= \sum_{i=0}^{k_1-1} \sum_{j=1}^{k_2} \boldsymbol{m}_{ik_2+j} \odot \mathsf{rot}_{\ell \cdot (ik_2+j)}(\boldsymbol{v}) \\ &= \sum_{i=0}^{k_1-1} \mathsf{rot}_{lk_2 i} \left(\sum_{j=1}^{k_2} \mathsf{rot}_{-lb_2 i}(\boldsymbol{m}_{ik_2+j}) \odot \mathsf{rot}_{\ell j}(\boldsymbol{v}) \right)\end{aligned}$$

where $\boldsymbol{m}_i = \mathsf{diag}_{\ell i}(\boldsymbol{M})$ and $k_1 k_2 = t$.

In this BSGS method we can obtain a matrix multiplication $\boldsymbol{M} \cdot \boldsymbol{v}$ by $O(k_1 + k_2)$ rotations and $O(t)$ constant multiplications. We remark that we can vary the choice of k_1 and k_2 by increasing t and add zero diagonals. For this reason, we can say that the hybrid method needs $O(\sqrt{t})$ homomorphic rotations and $O(t)$ number of homomorphic constant vector multiplications. The Table 4.1 shows comparison our methods with other techniques.

	Naive	BSGS	**Ours**	**Hybrid**
# Hadamard Mult	$O(n)$	$O(n)$	$O(r\log_r n)$	$O(r\log_r n)$
# Slot Shifting	$O(n)$	$O(\sqrt{n})$	$O(r\log_r n)$	$O(\sqrt{r}\log_r n)$
Depth	1	1	$O(\log_r n)$	$O(\log_r n)$

Table 4.1: Comparison: homomorphic operation number and depth consume for homomorphic DFT with radix r

Remark 4.2.2. Another advantage of our method is that it highly reduces the size of public key for operations. While the previous BSGS method requires $O(\sqrt{n})$ rotation key, our method only needs $O(r\log_r n)$ number of rotation key.

4.2.4 Implementation Result

We implemented our DFT algorithm using HeaAn library [snu18]. HeaAn library supports batch encodings, or encoding for vectors, for complex plaintext space thus it is suitable for our target; discret Fourier transform. All of experiments in this paper are done at the PC having 32 number of Intel(R) Xeon(R) CPU E5-2620 v4 2.10 GHz CPU (each CPU has 8 cores) and 64GB RAM. We used multi-threding with 8 number of threads.

The following HeaAn parameter setting is what we used in the experiment for our homomorphic DFT algorithm.

- $q_L = 2^{440}$: the largest ciphertext modulus.
- $N = 2^{15}$: the dimension of polynomial ring $\mathcal{R}$.
- $\Delta = p^k = 2^{30}$: scaling factor which is used to make integer polynomial in encryption and constant vector multiplication both.
- $\sigma = 3.2$, $\rho = 0.5$, and $h = 64$: distribution related parameters.

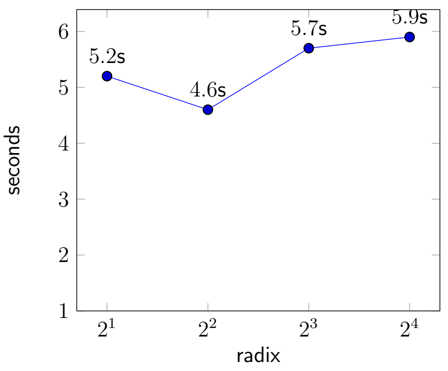

Figure 4.1: Timming results for various radix setting with dimension $n = 2^{12}$

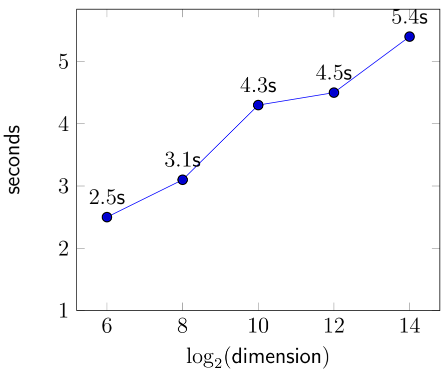

Figure 4.2: Timming results for various dimension setting with radix $r = 4$

Note that the expected security of this parameter setting is about 128 bit following the `LWEestimator` [APS15].

The Figure 4.1 and 4.2 show timing results for various setting. In case of the first one, radix varies from 2 to 16 with the fixed dimension of input vector 2^{12}. In case of the second one, dimension varies from 2^6 to 2^{12} with the fixed radix 4.

By the effect of baby-step giant-step method, the left one of the Figure 4.1 and 4.2 shows that timing does not increase a lot when we increase the radix. And, the right figure shows that timing increase linearly to logarithm of the dimension n. Therefore, we can get a homomorphic DFT algorithm which is significantly faster and similar depth consume. In our experiment, we compare the result with DFT on un-encrypted vector. We use average of $|a_i - b_i|$ for all $0 \leqslant i < n$ as difference between two length n vector $\boldsymbol{a}$ and $\boldsymbol{b}$. The difference between DFT on encrypted and un-encrypted state in our experiment is 2^{-9} to 2^{-10}. We can reduce this difference by using larger $\Delta = p^k$.

There are a few previous implementation results about homomorphic DFT. In [CSV17], there homomorphic DFT takes about 22 minutes for $n = 2^{13}$ with 8-bit precision. In [CKKS17], it takes about 22 minutes with same length. But these works focus on amortized time by put each element of the input vector in different ciphertext. We note that our results shows about 200 times faster than previous one.

4.3 Improved Bootstrapping for HEAAN

In this section, we explain about linear transformations in bootstrapping for approximate homomorphic encryption scheme. And, we give an improved transformation algorithms for such linear transform using our homomorphic DFT which provides an improved bootstrapping procedure for approximate homomorphic encryption.

4.3.1 Linear Transformation in Bootstrapping

The bootstrapping procedure for approximate homomorphic encryption in [CHK+18] can be divided as following steps:

1. Put polynomial coefficients in plaintext slots,

2. Evaluate exponent function,

3. Extract Imaginary part,

4, Switch back to the coefficient representation.

The transformations in the first and the last step are called CoeffToSlot and SlotToCoeff respectively. In [CHK+18], the authors use the index i of slots corresponding to $5^k \pmod{2N}$ for $0 \leqslant k \leqslant N/2$ by considering $w_{2N}^{5^k}$ as in Encode map. To transform coefficients of polynomial representation of plaintext into slots, we should construct two encodings since there are only $N/2$ slots while the number of coefficients is N.

Let $t(x) = t_0 + t_1 x + \cdots t_{N-1}x^{N-1}$ be a polynomial representation of encoding with messages $\boldsymbol{z} = (z_0, \cdots, z_{N/2-1})$ in slots, and let $\boldsymbol{v} = (t_0, \cdots, t_{N-1}) = (\boldsymbol{v}_0, \boldsymbol{v}_1)$ be its vector representation. Suppose that $\boldsymbol{U}$ be the encoding matrix defined in Section 4.1 and parsed into $[\boldsymbol{U}_0|\boldsymbol{U}_1]$ for $N/2$ by $N/2$ matrices $\boldsymbol{U}_k$'s. Then the following equation holds by definition of encoding map, which yields the $\mathsf{SlotToCoeff}$ map,

$$\boldsymbol{z} = \boldsymbol{U} \cdot \begin{bmatrix} \boldsymbol{v}_0 \\ \boldsymbol{v}_1 \end{bmatrix} = \boldsymbol{U}_0 \cdot \boldsymbol{v}_0 + \boldsymbol{U}_1 \cdot \boldsymbol{v}_1.$$

Note that $i \cdot \boldsymbol{U}_0 = \boldsymbol{U}_1$ and $\boldsymbol{U}_0^{-1} = \frac{2}{N} \cdot \overline{\boldsymbol{U}_0}^T$ hold. Using this, we can obtain that

$$\boldsymbol{v}_k = \frac{1}{N}\left(\overline{\boldsymbol{U}_k}^T \cdot \boldsymbol{z} + \boldsymbol{U}_k^T \cdot \overline{\boldsymbol{z}}\right) \text{ for } k = 0, 1.$$

This equation corresponds to $\mathsf{CoeffToSlot}$ map.

4.3.2 Improved CoeffToSlot and SlotToCoeff

We now describe a modified linear transforms for bootstrapping. We mainly focus on how to decompose the matrix $\boldsymbol{U}$ into sparse diagonal matrices. To obtain this, the bit-reversal permutation matrix $\boldsymbol{R}$ works a central role in this method. Note that the order of the slots after $\mathsf{CoeffToSlot}$ does not play any role in the bootstrapping. For this reason, we replace $\boldsymbol{U}_k$ to $\boldsymbol{V}_k$

which is row permuted by $\boldsymbol{R}$:

$$\boldsymbol{V}_k = \boldsymbol{U}_k \cdot \boldsymbol{R} \textit{ for } k = 0, 1.$$

As in $\boldsymbol{U}$, the relation $\boldsymbol{V}_1 = i \cdot \boldsymbol{V}_0$ holds. For this reason, we focus on the matrix decomposition of $\boldsymbol{V}_0$ using recursive relation; this induces the decomposition of $\boldsymbol{V}_1$. Let $\mathsf{rev}_n(i)$ denotes bit-reversal permutation of i with size n.

Lemma 4.3.1. *Let* $\boldsymbol{S}_n = \left(\omega_{4n}^{5^i \cdot \mathsf{rev}_n(j)}\right)_{0 \leqslant i,j < n}$. *Then,* $\boldsymbol{V}_0 = \boldsymbol{S}_{N/2}$ *and following equation holds:*

$$\boldsymbol{S}_n = \begin{bmatrix} I & \boldsymbol{W}_n \\ I & -\boldsymbol{W}_n \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{S}_{n/2} & 0 \\ 0 & \boldsymbol{S}_{n/2} \end{bmatrix}$$

for $\boldsymbol{W}_n = \mathsf{diag}(\omega_{4n}^{5^i})_{0 \leqslant i < n}$.

Proof. $\boldsymbol{V}_0 = \boldsymbol{S}_{N/2}$ is clear by definition. Let's start the proof with the following claim. Here $v_2(a)$ is the maximal integer k such that 2^k is a divisor of integer a.

Claim: $v_2(5^e - 1) = v_2(e) + 2$ holds for a positive integer e.

Proof: This claim can be proven using the mathematical induction on $v_2(e)$. ■

To prove the recursive formula, it suffices to show the following equation:

$$\boldsymbol{S}_n = \begin{bmatrix} \boldsymbol{S}_{n/2} & \boldsymbol{W}_n \cdot \boldsymbol{S}_{n/2} \\ \boldsymbol{S}_{n/2} & -\boldsymbol{W}_n \cdot \boldsymbol{S}_{n/2}. \end{bmatrix}$$

Let $\boldsymbol{S}_n = (s_{i,j})_{0 \leqslant i,j < n}$, i.e. $s_{i,j} = \omega_{4n}^{5^i \cdot \mathsf{rev}_n(j)}$. The following equations show the above equation. Note that $4n$ is a power of two integer.

1. $s_{i,j} = s_{i+n/2,j}$ for all i and for $0 \leqslant j < n/2$: this is equivalent to $4n | (5^{(i+n/2)} \cdot \mathsf{rev}_n(j) - 5^i \cdot \mathsf{rev}_n(j))$. By the claim $v_2(5^{n/2}-1) = v_2(n/2)+2 = v_2(2n)$ holds and $\mathsf{rev}_n(j)$ is even for $j < n/2$. Combining this we obtain the desired result is induced.

2. $s_{i,j} = -s_{i+N/2,j}$ for all i and for $n/2 \leqslant j < n$: as in the above, it is equivalent to $v_2(5^{(i+n/2)} \cdot \mathsf{rev}_n(j) - 5^i \cdot \mathsf{rev}_n(j)) = v_2(2n)$. It is showed by $v_2(5^{n/2}-1) = v_2(n/2)+2 = v_2(2n)$ and $\mathsf{rev}_n(j)$ is odd for $j \geqslant n/2$.

3. $s_{i,j+N/2} = s_{i,j} \cdot \omega_{4n}^{5^i}$ for all i and $0 \leqslant j < n/2$: this is clear by definition of rev_n.

If we combine these cases, we can easily show that the recursive relation of $\boldsymbol{S}$ holds. □

By adapting Lemma 4.3.1 repeatedly, we can decompose $\boldsymbol{V}_0$ to $\log_2 n$ number of matrices as in Equation 4.2.2. The following matrix illustrates the specific form of matrices in the recursive formula:

$$\boldsymbol{E}_k^{(n)} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{I}_{n/k} & \boldsymbol{W}_{n/k} \\ \boldsymbol{I}_{n/k} & -\boldsymbol{W}_{n/k} \end{bmatrix} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & \begin{bmatrix} \boldsymbol{I}_{n/k} & \boldsymbol{W}_{n/k} \\ \boldsymbol{I}_{n/k} & -\boldsymbol{W}_{n/k} \end{bmatrix} \end{bmatrix} \in \mathbb{C}^{n \times n}. \tag{4.3.3}$$

which has $k/2$ number diagonal blocks. Lemma 4.3.1 implies

$$\boldsymbol{V}_0 = \boldsymbol{E}_2^{(N/2)} \cdot \boldsymbol{E}_4^{(N/2)} \cdot \boldsymbol{E}_8^{(N/2)} \cdots \boldsymbol{E}_{N/2}^{(N/2)}.$$

These factor matrices have exactly the same structure with $\boldsymbol{D}_k^{(n)}$, so we can apply our method in previous section (from radix to hybrid method).

Furthermore, we can also multiply the inverse of $\boldsymbol{V}_0$ in encrypted state, as in the same way to the inverse DFT matrix case.

Now we will describe two linear transformations, CoeffToSlot and SlotToCoeff, using $\boldsymbol{V}_0$, $\boldsymbol{V}_0^{-1}$ and its conjugations. As we noted above, $\boldsymbol{V}_1 = i \cdot \boldsymbol{V}_0$ and further $\boldsymbol{V}_k^{-1} = \frac{2}{N}\overline{\boldsymbol{V}_k}^T$ hold as in the case of $\boldsymbol{U}$ for $k = 0, 1$. Therefore, CoeffToSlot with bit-reversed result and SlotToCoeff with bit-reversed input are computed as follows for $\boldsymbol{t}_k = \boldsymbol{R} \cdot v_k$ for $k = 0, 1$:

$$\boldsymbol{t}_0 = \frac{1}{2}\left(\boldsymbol{V}_0^{-1} \cdot \boldsymbol{z} + \overline{\boldsymbol{V}_0^{-1} \cdot \boldsymbol{z}}\right), \ \boldsymbol{t}_1 = -\frac{1}{2}i\left(\boldsymbol{V}_0^{-1} \cdot \boldsymbol{z} - \overline{\boldsymbol{V}_0^{-1} \cdot \boldsymbol{z}}\right),$$

$$\boldsymbol{z} = \boldsymbol{V}_0 \cdot (\boldsymbol{t}_0 + i \cdot \boldsymbol{t}_1).$$

Optimization

We can further improve the efficiency of the bootstrapping in light of *hoisting*, i.e. by computing the common part first or last. More precisely, for CoeffToSlot, compute $\boldsymbol{V}_0^{-1} \cdot \boldsymbol{z}$ first and compute other parts using conjugation. Therefore, $\boldsymbol{t}_0$ and $\boldsymbol{t}_1$ can be computed from $\boldsymbol{z}$ in $2\sqrt{r}\log_r(N/2)$ homomorphic operations for the radix r. For SlotToCoeff, we compute $(\boldsymbol{t}_0 + i \cdot \boldsymbol{t}_1)$ first and multiply $\boldsymbol{V}_0$. This also needs only $2\sqrt{r}\log_r(N/2)$ number of homomorphic operations.

Remark 4.3.1. Our technique can be applied for bootstrapping of $(n/2)$-sparsely packed ciphertext in [CHK+18]. The plaintext space of sparse packed ciphertext is $\mathbb{Z}[Y]/(Y^n + 1)$ for $Y = X^{N/n}$. So, we just need to replace ω_{2N} to ω_{2n}.

4.3.3 Implementation Result

Use one of the parameter sets which is in the previous work [CHK+18] for easier comparison. And, we run the previous method which is implemented in HeaAn library [snu18] in the same machine for fare comparison (with recently release version v2.1). The PC information is same as the previous implementation in Section 4.2.4.

- $q_0 = 2^{41}$: the smallest ciphertext modulus (before bootstrapping).
- $q_L = 2^{1240}$: the largest ciphertext modulus.
- $N = 2^{16}$: the dimension of polynomial ring $\mathcal{R}$.
- $\Delta = p^k = 2^{31}$: scaling factor which is used to make integer polynomial in encryption and constant vector multiplication both.
- $\sigma = 3.2$, $\rho = 0.5$, and $h = 64$: distribution related parameters.
- $r = 7$ which is the number of iteration in sin evaluation.

The Table 4.2 shows implementation result of bootstrapping using our linear transformation and previous method. To maximize the effect of our method, we used number of slots as the largest one ($= N/2$).

	Key Gen	Linear Trans	Eval sin	**Total**
Previous	25 hours	26 hours	30 sec	26 hours
Ours	**44 sec**	**97 sec**	30 sec	127 sec

Table 4.2: Timing of Bootstrapping with comparison for $\mathbb{C}^{32768}$ plaintext space. Here amortized time means that bootstrapping time per one complex element. Both works gives about 2^{-7} additive error while bootstrapping.

The timing results for linear transformation time shows about 700 times faster result than previous one. We use radix 32 which means each linear transformation consumes 3 ($= \log_{32} 2^{15}$) constant vector multiplication depth. As a result, the modulus of the return ciphertext is 468 bits which means 14 depth computation can be done after bootstrapping. In the previous method, the modulus of the return ciphertext is 632 bits which means 19 depth computation can be done after bootstrapping.

Another advantage of our method is key generation time. Key generation includes public key generation for various rotations and pre-encodings for diagonal vectors. In the previous method, they need to encode for $N/2(= 32768)$ number of constant vectors for each linear transformation. The number of rotation key is $2\sqrt{N/2}$ which is quite large compare to $2\sqrt{k}\log_k N/2$ in our case. In the experiment, this problem makes their key generation time slower and the size of pre-encoded vector and public keys to be huge. Previous method need 800GB to save them and 7GB for ours.

Remark 4.3.2. In this paper, we only consider the linear transformation part of bootstrapping. But we can use improved evaluation strategy for sine as in [CCS18]. This paper use Chebyshev approximation method instead of double angle plus Taylor approximation. This method gives better accuracy and the depth is reduced with slightly larger complexity.

Chapter 5

Faster Bootstrapping for FHE over the integers

In this chapter, we proposed new bootstrapping method for fully homomorphic encryption over the integers. The first FHE over the integers was proposed by van Dijk et al. [vDGHV10], and it was extended to the batch version [CCK$^+$13] and the non-binary message space version [NK15]. Furthermore, following the scale-invariant technique of a lattice-based FHE [BGV12], so called the modulus switching technique, Coron et al. [CLT14] succeeded to construct a scale-invariant FHE over the integers.

In FHEs over the integers with the secret integer $p \in \mathbb{Z}^+$, the message $m \in \mathbb{Z}_t$ is encrypted into an integer $c = pq + te + m$, $pq + \lfloor p/t \rceil m + e$ or $p^2 q + \lfloor p/t \rceil (m + tr) + e$ according to the schemes [vDGHV10, CS15, CLT14], where the q, r, e are uniform randomly chosen integers from some prescribed intervals. Hence the bootstrapping procedure is to homomorphically evaluate the decryption function

$$m = \lfloor \frac{t}{p} \cdot c \rceil \bmod t \text{ or } c - p \cdot \lfloor \frac{c}{p} \rceil \bmod t.$$

The complicated division by p can be relaxed by using the hardness assumption of the sparse subset sum problem (SSSP): $1/p \approx \sum_{i=1}^{\Theta} s_i y_i \bmod t$ for secret bit $s_i \in \{0, 1\}$, public rational number $y_i \in [0, t)$ of κ-bit precision with $\kappa > \log |c| + \lambda$ and $\Theta = \tilde{O}(\lambda^4)$. In that case, the decryption function is reduced to

$$m = \left\lfloor \sum_{i=1}^{\Theta} s_i \frac{w_i}{t^{n-1}} \right\rceil \bmod t \text{ or } c - p \cdot \left\lfloor \sum_{i=1}^{\Theta} s_i \frac{w_i}{t^{n}} \right\rceil \bmod t,$$

where $n = O(\log_t \lambda)$ and $w_i = \lfloor c \cdot y_i \cdot t^n \rceil \bmod t^{n+1}$.

In the previous methods [vDGHV10, NK15], each s_i is encrypted under a HE with message space $\mathbb{Z}_t$, so we need to expand each w_i t-adically and each digits of $s_i w_i$ are encrypted separately. As a result, each of digits of $s_i w_i$ are encrypted as different ciphertexts, so the homomorphic evaluation of the additions in the decryption circuit should be done digit-wisely. In that case, a large number of carry computations are required, which results in $\tilde{O}(\lambda^4)$ homomorphic multiplication in the bootstrapping. One possible approach to avoid this massive homomorphic multiplication is to encrypt s_i with a HE with plaintext space as large as w_i. However, in that case, $\log t$-bit should be homomorphically extracted and this is regarded as a problem as hard as bootstrapping.

Overview of our Bootstrapping method. The idea of our new bootstrapping method is to use a homomorphic encryption scheme with *various message spaces*. Let us denote $\mathsf{Enc}_{\mathcal{M}}(m)$ be a ciphertext of message m under an encryption with plaintext space $\mathcal{M}$. For a given bootstrapping key $(\mathsf{bk}_i)_{1 \leqslant i \leqslant \Theta} = (\mathsf{Enc}_{\mathbb{Z}_{t^{n+1}}}(s_i))_{1 \leqslant i \leqslant \Theta}$, the output of the homomorphic additions for bootstrapping is the ciphertext $\hat{c}$ of the form $\hat{c} = \sum_{i=1}^{\Theta} w_i \cdot \mathsf{bk}_i = \mathsf{Enc}_{\mathbb{Z}_{t^{n+1}}}(m \cdot t^n + \sum_{j=0}^{n-1} z_j \cdot t^j)$ for some integers $z_j \in [0, t)$. To complete the bootstrapping process, it is required to compute the ciphertext $c = \mathsf{Enc}_{\mathbb{Z}_t}(m)$ from the ciphertext $\hat{c}$.

For this, we first suggest plaintext space contraction and dilation functions over the ciphertexts of HEs over the integers.

$$\mathsf{PSCon}_i : \mathsf{Enc}_{\mathbb{Z}_{t^k}}(m \cdot t^i) \to \mathsf{Enc}_{\mathbb{Z}_{t^{k-i}}}(m) \text{ for } 1 \leqslant i < k,$$

$$\mathsf{PSDil}_i : \mathsf{Enc}_{\mathbb{Z}_{t^{k-i}}}(m) \to \mathsf{Enc}_{\mathbb{Z}_{t^k}}(m \cdot t^i) \text{ for } 1 \leqslant i < k,$$

which do not affect error growth. In case of lattice-based FHEs, these techniques already exist and bootstrapping can be done efficiently by exploiting them. In this paper, we suggest PSCon and PSDil techniques for HEs over the integers. With these techniques, we can homomorphically extract n-th digit of $(m \cdot t^n + \sum_{i=0}^{n-1} z_i \cdot t^i)$ in HEs over the integers using a gap-increasing polynomial $F_{t,n}(X)$ suggested by Halevi and Shoup [HS15], which satisfies the following equation:

$$F_{t,n}(x \cdot t^k + a) = y \cdot t^{k+1} + a \text{ for } a \in [0, t) \cap \mathbb{Z}, x, y \in \mathbb{Z}.$$

The overview of homomorphic digit extraction with the functions PSCon and PSDil in HE schemes over the integers are follows. For digit extraction, we need to compute $c_i = \mathsf{Enc}_{\mathbb{Z}_{t^{n-i+1}}}(tx_i + z_i)$ for $0 \leqslant i \leqslant n$ and $x_i \in \mathbb{Z}$.

Assume that following ciphertexts are given:

$$\begin{aligned} c_0 &= \mathsf{Enc}_{\mathbb{Z}_{t^{n+1}}}(tx_0 + z_0) \\ c_1 &= \mathsf{Enc}_{\mathbb{Z}_{t^n}}(tx_1 + z_1) \\ &\vdots \\ c_{i-1} &= \mathsf{Enc}_{\mathbb{Z}_{t^{n-i+2}}}(tx_{i-1} + z_{i-1}). \end{aligned}$$

By $(i-j)$ time evaluating $F_{t,n}(X)$ for each c_j and using PSDil technique, we can get $c'_j = \mathsf{Enc}_{\mathbb{Z}_{t^{n+1}}}(y_j t^{i+1} + z_j t^j)$ for each $j \in \{0, 1, \cdots, i-1\}$. By subtraction and PSCon technique, we get $c_i = \mathsf{Enc}_{\mathbb{Z}_{t^{n-i+1}}}(tx_i + z_i)$. Now we can get c_i from c_j for $1 \leqslant j \leqslant i-1$, so we can compute $c_n = \mathsf{Enc}_{\mathbb{Z}_t}(m)$ recursively. The figure below is our bootstrapping process with simple case of $n = 2$ and $t = 2$.

As you can see in Figure 1, our bootstrapping process is simpler than previous works which are very hard to describe. In our implementations for the security parameter $\lambda = 72$, we set parameters $n = 5$ and $t = 2$, which are very small.

	[NK15]	Our Method
Degree	$O(\lambda)$	$O(\lambda^{1+\epsilon})$
#.$\mathsf{Hommult}$	$O(\lambda^4 \log^6 \lambda)$	$O(\log^2 \lambda)$

Table 5.1: Comparison with NK15 method

Faster bootstrapping method. We propose a faster bootstrapping method for FHEs over the integers than previous methods [vDGHV10, NK15]. Since the complexity of long integers addition depends on both the length and the number of the integers, the number of multiplications in

$$\begin{tikzcd}
 & \sum \bar{s}_i w_i \arrow[r, "\mathsf{HomAdd}"] & \mathrm{Enc}_{\mathbb{Z}_8}(4m+2a+b) \arrow[d, "X^2 \bmod 8"] \\
\mathrm{Enc}_{\mathbb{Z}_8}(4m+2a+b) \arrow[d, Rightarrow, "\text{substract}"] & & \mathrm{Enc}_{\mathbb{Z}_8}(4\star+b) \arrow[ddd, "X^2 \bmod 8"] \\
\mathrm{Enc}_{\mathbb{Z}_8}(4\star+2a) \arrow[r, "\mathsf{PSCon}"] & \mathrm{Enc}_{\mathbb{Z}_4}(2\star+a) \arrow[d, "X^2 \bmod 4"] & \\
 & \mathrm{Enc}_{\mathbb{Z}_4}(a) \arrow[d, "\mathsf{PSDil}"] & \\
\mathrm{Enc}_{\mathbb{Z}_8}(4m+2a+b) \arrow[d, Rightarrow, "\text{substract}"] & \mathrm{Enc}_{\mathbb{Z}_8}(2a) & \mathrm{Enc}_{\mathbb{Z}_8}(b) \\
\mathrm{Enc}_{\mathbb{Z}_8}(4m) \arrow[r, "\mathsf{PSCon}"] & \mathrm{Enc}_{\mathbb{Z}_2}(m) &
\end{tikzcd}$$

Figure 5.1: Our bootstrapping process for simple case. Note that $F_{2,n}(X) = X^2$ for all $n \in \mathbb{N}$. Here star shape denotes some integer of which we do not need to consider the exact value.

previous works relies on the large parameter $\Theta = \tilde{O}(\lambda^4)$. Contrary to that, the complexity of homomorphic digit extraction only depends on the small parameter $n = O(\log \lambda)$, and this difference makes our method efficient. Table 5.1 below is a comparison with the previous method [NK15]. Note that the small constant ϵ is consequence of using the large message space $\mathbb{Z}_{t^{n+1}}$.

An implementation on the CLT scheme. The CLT scheme has the fastest homomorphic multiplication algorithm among the HEs over the integers, and the ciphertext form of the scheme is appropriate to apply our method. Therefore, we apply our method on the CLT scheme, and provide a precise noise analysis of the bootstrapping procedure. Our implementation on the CLT scheme takes about 6 seconds for a 500-bit message space with a 80-bit security. This result is far superior comparing to the previous result in [CCK$^+$13], 13 minutes for a 500-bit message space.

Homomorphic evaluation of AES circuit. Due to the inefficiency of bootstrapping, homomorphic evaluations of AES circuit with leveled or scale-invariant FHEs so far have been implemented without bootstrapping. Contrary to the previous works, we implement a homomorphic evaluation of an AES-128 circuit using our bootstrapping method on the CLT scheme with low depth parameters. In our implementation, the evaluation takes about 8 seconds per block, and this result is faster than the result in [CLT14] without bootstrapping (with large depth parameters), 26 seconds per block. Furthermore, this is the first time that homomorphic evaluation of AES circuit *with bootstrapping is more efficient* than homomorphic evaluation of AES circuit without bootstrapping.

5.1 Basis of FHE over the integers

In this section, we will introduce HE scheme based on integer problems. DGHV and CS schemes are based on AGCD problem and CLT scheme is based on variant of the problem. We note that our new method for bootstrapping can be adapted to all these three schemes. But we will focus on CLT scheme which is the most efficient integer based HE scheme. We follow the notation from original paper [CLT14], and describe the scheme with the message space $\mathbb{Z}_t$ for a positive integer t [*]. For an η-bit odd integer

[*] Original schemes are described with plaintext space $\mathbb{Z}_2$, but the extension to $\mathbb{Z}_t$ is trivial.

p and an integer q_0 in $[0, 2^\gamma/p^2)$, we define a distribution as follows:

$$\mathcal{D}^{\rho}_{p,q_0} = \{p^2 \cdot q + r : \textsf{Choose } q \leftarrow [0, q_0), r \leftarrow (-2^\rho, 2^\rho)\}.$$

Note that this distribution is hard to distinguish with uniform distribution in $[0, 2^\gamma)$ based on the variant of AGCD problem in Section 2.4. Followings are brief description of CLT scheme (for more information see [CLT14]).

KeyGen$_t(1^\lambda)$**.** Generate an odd η-bit integer p and a γ-bit integer $x_0 = q_0 \cdot p^2 + r_0$ with $r_0 \leftarrow (-2^\rho, 2^\rho) \cap \mathbb{Z}$ and $q_0 \leftarrow [0, 2^\gamma/p^2) \cap \mathbb{Z}$. Let $x_i \leftarrow \tilde{\mathcal{D}}^{\rho}_{p,q_0}$ for $1 \leqslant i \leqslant \tau$, $y' \leftarrow \tilde{\mathcal{D}}^{\rho}_{p,q_0}$, and $y = y' + \lfloor p/t \rceil$, which is the encryption of 1. Let $\boldsymbol{z}$ be a vector of length Θ, the components of which have $\kappa = 2\gamma + 2$ bits of precision following the binary point. Let $\boldsymbol{s} \in \{0,1\}^\Theta$ such that

$$\frac{t \cdot 2^\eta}{p^2} = \langle \boldsymbol{s}, \boldsymbol{z} \rangle + \epsilon \mod (t \cdot 2^\eta),$$

with $|\epsilon| \leqslant 2^{-\kappa}$. Now define

$$\boldsymbol{\sigma} = \boldsymbol{q} \cdot p^2 + \boldsymbol{r} + \left\lfloor \textsf{PowersofTwo}_\eta(\boldsymbol{s}) \cdot \frac{p}{2^{\eta+1}} \right\rceil,$$

where the components of $\boldsymbol{q}$ are randomly chosen from $[0, q_0) \cap \mathbb{Z}$ and those of $\boldsymbol{r}$ from $(-2^\rho, 2^\rho) \cap \mathbb{Z}$. The secret key is $\textsf{sk} = \{p\}$ and the public key is $\textsf{pk} = \{x_0, x_1, \cdots, x_\tau, y, \boldsymbol{\sigma}, \boldsymbol{z}\}$.

Encrypt$_t$(**pk**, $m \in [t]$)**.** Choose a random subset $S \subset \{1, \cdots, \tau\}$ and output

$$c \leftarrow [m \cdot y + \sum_{i \in S} x_i]_{x_0}.$$

Decrypt$_t(\mathsf{sk}, c)$. Output $m \leftarrow \left\lfloor t \cdot \frac{c}{p} \right\rceil \bmod t$.

Add$_t(\mathsf{pk}, c_1, c_2)$**.** Output $c' \leftarrow c_1 + c_2 \bmod x_0$.

Convert$_t(\mathsf{pk}, c)$**.** Output $c' \leftarrow 2 \cdot \langle \boldsymbol{\sigma}, \mathsf{BitDecomp}_\eta(\boldsymbol{c}) \rangle$ where $\boldsymbol{c} = (\lfloor c \cdot z_i \rceil \bmod 2^\eta)_{1 \leqslant i \leqslant \Theta}$.

Mult$_t(\mathsf{pk}, c_1, c_2)$**.** Output $c' \leftarrow [\mathsf{Convert}(\mathsf{pk}, c_1 \cdot c_2)]_{x_0}$.

Remark 5.1.1. Small difference with original scheme is using plaintext $\mathbb{Z}_t$ and $\langle \boldsymbol{s}, \boldsymbol{z} \rangle + \epsilon$ is $\frac{t \cdot 2^\eta}{p^2}$ instead of $\frac{2^\eta}{p^2}$. This changes the homomorphic multiplication algorithm as $[\mathsf{Convert}(\mathsf{pk}, c_1 \cdot c_2)]_{x_0}$ instead of $[\mathsf{Convert}(\mathsf{pk}, 2 \cdot c_1 \cdot c_2)]_{x_0}$.

Semantic Security

Security for this scheme is from same problem introduced in [CLT14]. The only difference is change of message space from $\mathbb{Z}_2$ to $\mathbb{Z}_t$, so we omit this part. Note that the security of this scheme is from hardness of the variant of AGCD problem in Section 2.4.

Conditions on the Parameters

The parameters must satisfy the following conditions for security parameter λ and message space $\mathbb{Z}_t$:

- $\rho = \Omega(\lambda)$ to avoid brute force attacks on noise [CN12, CNT12],
- $\eta \geqslant \rho + O(L(\log \lambda + \log t))$, where L is the depth of multiplication of the circuits to be evaluated,

- $\gamma \geqslant \omega((2\eta - \rho)^2 \cdot \log \lambda)$ to avoid lattice-based attacks [vDGHV10, CMNT11],
- $\Theta^2 \geqslant \gamma \cdot \omega(\log \lambda)$ to avoid lattice attacks on the subset sum problem [CMNT11],
- $\tau \geqslant \gamma + 2\lambda$ to apply the leftover hash lemma.

5.2 Decryption Function via Digit Extraction

5.2.1 Squashed Decryption Function

In the FHEs over the integers with secret integer p, a ciphertext of message $m \in [t]$ is of the form $c = pq + te + m$ or $pq + \frac{p}{t}m + e$ for $q, e \in \mathbb{Z}$. Each form of ciphertext is decrypted by

$$\left(c - p \cdot \left\lfloor \frac{c}{p} \right\rceil\right) \bmod t \text{ or } \left(\left\lfloor \frac{t}{p} \cdot c \right\rceil\right) \bmod t.$$

The decryption function involves the computation of $\lfloor c/p \rceil$ and this should be evaluated homomorphically for bootstrapping. Since division is very complicated for homomorphic evaluation, decryption functions of FHEs over the integers are *squashed* for efficient bootstrapping. The *Squashing* is a procedure of expressing secret value $1/p$ as a subset sum of public numbers within very small error, which enable to bootstrap efficiently.

The squashing technique was first introduced in [vDGHV10], and generalized in [NK15]. Let κ', Θ', and θ' be additional parameters satisfying $\kappa' > (\gamma + \lambda)/\log t$ (from security of sparse subset sum problem). The difference with the previous methods of squashing is using digit extraction instead of rounding function. Followings are out squashed decryption function for CLT scheme.

- KeyGen. Generate secret key and public key as same as original scheme. Set $x_p = \lfloor t^{\kappa'+1}/p \rceil$, choose a random Θ'-bit vector s with Hamming weight θ', and let $S = \{i : s_i = 1\}$. Choose random integers $u_i \in [0, t^{\kappa'+1})$ such that $\sum_{i \in S} u_i = x_p$.
- Encrypt. c^* is a ciphertext of a given FHE over the integers. For $1 \leqslant i \leqslant \Theta'$, let w_i given by an integer nearest to the value of $c^* \cdot u_i / t^{\kappa'-n}$ where $n = \lceil \log_t \theta' \rceil + 3$. Output both c^* and $\boldsymbol{w}$.
- Decrypt. Output $m' \leftarrow \mathsf{digitExtrac}_{n+1,r}(\sum s_i w_i + \lfloor t^n/2 \rceil)$.

Remark 5.2.1. The squashing technique can be applied not only to the original scheme in [vDGHV10, CLT14, CS15], but also to the batch version of the scheme by squashing for each p_j as in [NK15].

5.2.2 Digit extraction Technique

Let $F^k(X)$ be a k-time evaluation of the function F. In general, $F(X) = X^t$ does not satisfy the following property when $t > 2$:

$$F^k(x) \bmod t^{k+1} = x \bmod t \quad \forall k \in \mathbb{N}, x \in [0, t) \cap \mathbb{Z}.$$

In [HS15], for the prime t and the positive integer e, they constructed the polynomial $F_{t,e}(X)$ satisfying the above equation for any $k \leqslant e$. With this polynomial, we can extract $a\langle e' \rangle_t$ for $1 \leqslant e' \leqslant e$ using similar method in [GHS12a]. Following lemmas are about existence and construction of the polynomial $F_{t,e}(X)$, which are introduced in [HS15].

Digit Extraction Algorithm:

Input: non-negative integers x and r

Compute x_i for $1 \leqslant i \leqslant r$ as following :

$$\begin{aligned}
x_0 &= x \\
x_1 &= \frac{[x - F_{t,r}(x)]_{t^{r+1}}}{t} \\
x_2 &= \frac{[x - F_{t,r}^2(x) - tF_{t,r}(x_1)]_{t^{r+1}}}{t^2} \\
x_3 &= \frac{[x - F_{t,r}^3(x) - tF_{t,r}^2(x_1) - t^2F_{t,r}(x_2)]_{t^{r+1}}}{t^3} \\
&\vdots \\
x_r &= \frac{[x - F_{t,r}^r(x) - \sum_{i=1}^{r-1} t^i F_{t,r}^{r-i}(x_i)]_{t^{r+1}}}{t^r}
\end{aligned}$$

Output: $x_r = x^{(t)}\langle r\rangle$

Figure 5.2: Digit Extraction Algorithm

Lemma 5.2.1. *(Corollary 5.4 in [HS15]) For every prime t, there exists a sequence of integer polynomial $f_1, f_2, \cdots$, all of degree $\leqslant t-1$, such that for every exponent $e \geqslant 1$ and every integer $z = z_0 + t^e z_1$ ($z_0 \in [t]$, $z_1 \in \mathbb{Z}$), we have*

$$z^t \equiv z_0 + \sum_{i=1}^{e} f_i(z_0)t^i \pmod{t^{e+1}}.$$

Lemma 5.2.2. *(Corollary 5.5 in [HS15]) For every prime t and every $e \geqslant 1$, there exists a polynomial $F_{t,e}$ of degree p such that the equality $F_{t,e}(z_0 + t^{e'} z_1) \equiv z_0 \pmod{t^{e'+1}}$ holds for every integer z_0, z_1 with $z_0 \in [t]$ and every $1 \leqslant e' \leqslant e$.*

Using a special polynomial $F_{t,r}$, we can extract $x^{(t)}\langle r\rangle$ from x, through a polynomial for any non-negative integers x and r, by digit extraction algorithm in Figure 2. Note that the equality in Lemma 5.2.2 implies that recursively defined x_is are integers.

5.2.3 Homomorphic Digit Extraction in FHE over the integers

To follow the digit extraction method in [GHS12b] at FHE over the integers, additional method to control plaintext space is needed.

Plaintext Space Contraction and Dilation

Let $\mathcal{E}_k(m)$ is a set of ciphertexts which encrypt m with message space $\mathcal{M} = \mathbb{Z}_{t^k}$. For all HE schemes over the integers, we can construct following two plaintext space switching functions for $1 \leqslant i < k$:

$$\mathsf{PSCon}_i : \mathcal{E}_(t^i m) \to \mathcal{E}_{k-i}(m),$$

$$\mathsf{PSDil}_i : \mathcal{E}_{k-i}(m) \to \mathcal{E}_k(t^i m).$$

The definitions of these functions are somewhat different depending on the form of a ciphertext.

Case 1 ($c = pq+tr+m$). In the schemes of [vDGHV10] and [NK15], with public exact multiplication $x_0 = pq_0$, plaintext space switching functions are described as follows:

- $\mathsf{PSCon}_i(c) = [t^{-i}]_{x_0} \cdot c \bmod x_0$,
- $\mathsf{PSDil}_i(c) = t^i c \bmod x_0$.

Correctness of these functions can be checked easily by following equations:

$$\begin{aligned}\mathsf{PSCon}_i(c) &= [t^{-i}]_{x_0}pq + [t^{-i}]_{x_0}t^k r + [t^{-i}]_{x_0}t^i m \bmod x_0\\ &= pq' + t^{k-i}r + m,\\ \mathsf{PSDil}_i(c) &= t^i pq + t^{i+1}r + t^i m \bmod x_0\\ &= pq' + t^{i+1}r + t^i m.\end{aligned}$$

Case 2 ($c = pq + \lfloor p/t \rceil \cdot m + r$). In the schemes of [CS15] and [CLT14], the functions PSCon_i and PSDil_i are identity functions since

$$\left\lfloor \frac{p}{t^k} \right\rceil (tm + t^k r^*) = \left\lfloor \frac{p}{t^{k-1}} \right\rceil (m + t^{k-1} r^*) + \epsilon,$$

$$\left\lfloor \frac{p}{t^k} \right\rceil \cdot tm = \left\lfloor \frac{p}{t^{k-1}} \right\rceil \cdot m + \epsilon.$$

Homomorphic digit extraction

The rounding function in the squashed decryption function can be expressed as follows:

$$\begin{aligned}\lfloor \textstyle\sum s_i w_i / t^n \rceil \bmod t &= \lfloor \textstyle\sum s_i w_i / t^n + 0.5 \rfloor \bmod t\\ &= (\textstyle\sum s_i w_i + \lfloor t^n/2 \rfloor)^{(t)}\langle n \rangle.\end{aligned}$$

Thus, the squashed decryption could be expressed as additions and a digit-extraction. The problem is how to homomorphically evaluate the function $(\sum s_i w_i + \lfloor t^n/2 \rfloor)^{(t)}\langle n \rangle$ where each w_i is defined in Section 5.2.1.

Let t be a prime integer, n be a positive integer less than $\log \lambda$, and $\mathcal{M}$ be a message space. We follow notations in Section 5.2.1 about squashing.

In this Section, we suggest a new bootstrapping method. It works on any HE over the integers which satisfies following conditions:

1. The form of a decryption function is

$$\lfloor \sum s_i w_i / t^n \rceil \bmod t \text{ or } c - \lfloor \sum s_i w_i / t^n \rceil \bmod t$$

   where w_i can be computed by public values c and u_i.

2. It supports homomorphic operations with $\mathcal{M} = \mathbb{Z}_{t^i}$ for $1 \leqslant i \leqslant n+1$.

3. There exists a polynomial time algorithm HomExt, a function from $\mathcal{E}_{n+1}(m)$ to $\mathcal{E}_1(m\langle n \rangle_t)$, which is a homomorphic evaluation of digit-extraction algorithm in Figure 5.2.

Given a HE over the integers satisfying the conditions above, our bootstrapping method works as diagram below. New parameters $s_0 = 1$ and $w_0 = \lfloor t^n/2 \rfloor$ are included in the summation.

$$\begin{tikzcd}
c \arrow[r, "\mathsf{Enc}_{\mathbb{Z}_{t^{n+1}}}(s_i),\, u_i"] \arrow[dr, "\mathsf{Bootstrap}"'] & \mathsf{Enc}_{\mathbb{Z}_{t^{n+1}}}(\sum s_i w_i) \arrow[d, "\mathsf{HomExt}"] \\
 & \mathsf{Enc}_{\mathbb{Z}_t}(m)
\end{tikzcd}$$

Actually, all HEs over the integers satisfy the above conditions, which means our method can be applied to all integers-based HEs. In the diagram, our bootstrapping method consists of two steps: addition and extraction. Since w_i can be computed by public values and the set $\{\mathsf{Enc}_{\mathbb{Z}_{t^{n+1}}}(s_i)\}$ is given as bootstrapping key, the addition step in this diagram is composed of homomorphic additions on $\mathcal{M} = \mathbb{Z}_{t^{n+1}}$ and modulus operation. Note that modulus operation $\bmod t^{n+1}$ is automatically done since the message space is given by $\mathcal{M} = \mathbb{Z}_{t^{n+1}}$.

5.3 Bootstrapping for FHE over the integers

We apply our method on scale-invariant homomorphic encryption scheme in [CLT14], the CLT scheme, since error growth during homomorphic evaluation is linear so that it is suitable to choose low depth parameter for implementation. Furthermore, as mentioned in remark 1, since PSCon and PSDil are trivial mapping, the description of HomExt is very simple.

As mentioned above, we need three conditions: squashed decryption function with $\mathcal{M} = \mathbb{Z}_t$, homomorphic operations on message spaces $\mathbb{Z}_{t^a}$, and homomorphic digit extraction technique. The scheme below is almost same with scale-invariant homomorphic encryption scheme in [CLT14], and we just extend it to the message space $\mathbb{Z}_t$ for the prime t.

5.3.1 CLT scheme with $\mathcal{M} = \mathbb{Z}_t$

Scheme Description

In this section, we follow the notation in [CLT14], and describe the scheme with the message space $\mathbb{Z}_t$ for the prime t. For an η-bit odd integer p and an integer q_0 in $[0, 2^\gamma/p^2)$, we define the set

$$\mathcal{D}^{\rho}_{p,q_0} = \{p^2 \cdot q + r : \mathsf{Choose}\ q \leftarrow [0, q_0), r \leftarrow (-2^\rho, 2^\rho)\}.$$

- $\mathsf{KeyGen}_t(1^\lambda)$. Generate an odd η-bit integer p and a γ-bit integer $x_0 = q_0 \cdot p^2 + r_0$ with $r_0 \leftarrow (-2^\rho, 2^\rho) \cap \mathbb{Z}$ and $q_0 \leftarrow [0, 2^\gamma/p^2) \cap \mathbb{Z}$. Let $x_i \leftarrow \mathcal{D}^{\rho}_{p,q_0}$ for $1 \leqslant i \leqslant \tau$, $y' \leftarrow \mathcal{D}^{\rho}_{p,q_0}$, and $y = y' + \lfloor p/t \rceil$, which is the encryption of 1. Let $\boldsymbol{z}$ be a vector of length Θ, the components of which have $\kappa = 2\gamma + 2$ bits of precision following the binary point. Let $\boldsymbol{s} \in \{0,1\}^\Theta$ such that

$$\frac{t \cdot 2^\eta}{p^2} = \langle \boldsymbol{s}, \boldsymbol{z} \rangle + \epsilon \mod (t \cdot 2^\eta),$$

  with $|\epsilon| \leqslant 2^{-\kappa}$. Now define

$$\boldsymbol{\sigma} = \boldsymbol{q} \cdot p^2 + \boldsymbol{r} + \left\lfloor \mathsf{PowersofTwo}_\eta(\boldsymbol{s}) \cdot \frac{p}{2^{\eta+1}} \right\rceil,$$

  where the components of $\boldsymbol{q}$ are randomly chosen from $[0, q_0) \cap \mathbb{Z}$ and those of $\boldsymbol{r}$ from $(-2^\rho, 2^\rho) \cap \mathbb{Z}$. The secret key is $\mathsf{sk} = \{p\}$ and the public key is $\mathsf{pk} = \{x_0, x_1, \cdots, x_\tau, y, \boldsymbol{\sigma}, \boldsymbol{z}\}$.

- $\mathsf{Encrypt}_t(\mathsf{pk}, m \in [t])$. Choose a random subset $S \subset \{1, \cdots, \tau\}$ and output

$$c \leftarrow [m \cdot y + \sum_{i \in S} x_i]_{x_0}.$$

- $\mathsf{Decrypt}_t(\mathsf{sk}, c)$. Output $m \leftarrow \left\lfloor t \cdot \frac{c}{p} \right\rceil \bmod t$.
- $\mathsf{Add}_t(\mathsf{pk}, c_1, c_2)$. Output $c' \leftarrow c_1 + c_2 \bmod x_0$.
- $\mathsf{Convert}_t(\mathsf{pk}, c)$. Output $c' \leftarrow 2 \cdot \langle \boldsymbol{\sigma}, \mathsf{BitDecomp}_\eta(\boldsymbol{c}) \rangle$ where $\boldsymbol{c} = (\lfloor c \cdot z_i \rceil \bmod 2^\eta)_{1 \leqslant i \leqslant \Theta}$.
- $\mathsf{Mult}_t(\mathsf{pk}, c_1, c_2)$. Output $c' \leftarrow [\mathsf{Convert}(\mathsf{pk}, c_1 \cdot c_2)]_{x_0}$.

Semantic Security

Security for this scheme is from same problem introduced in [CLT14]. The only difference is change of message space from $\mathbb{Z}_2$ to $\mathbb{Z}_t$, so we omit this part.

Conditions on the Parameters

The parameters must satisfy the following conditions for security parameter λ and message space $\mathbb{Z}_t$:

- $\rho = \Omega(\lambda)$ to avoid brute force attacks on noise [CN12, CNT12],
- $\eta \geqslant \rho + O(L(\log \lambda + \log t))$, where L is the depth of multiplication of the circuits to be evaluated,
- $\gamma \geqslant \omega((2\eta - \rho)^2 \cdot \log \lambda)$ to avoid lattice-based attacks [vDGHV10, CMNT11],

- $\Theta^2 \geqslant \gamma \cdot \omega(\log \lambda)$ to avoid lattice attacks on the subset sum problem [CMNT11],
- $\tau \geqslant \gamma + 2\lambda$ to apply the leftover hash lemma.

5.3.2 Homomorphic Operations with $\mathcal{M} = \mathbb{Z}_{t^a}$

During bootstrapping, we use homomorphic addition and multiplication between ciphertexts on the message space $\mathcal{M} = \mathbb{Z}_{t^a}$ for $1 \leqslant a \leqslant \log_t \lambda$. Homomorphic addition and multiplication are described below. Note that x_0 is defined in the same manner as in the previous section, and the definition of Eval is non-deterministic since the method of the evaluation depends on the formation of a given polynomial.

- $\mathsf{Add}_t^a(\mathsf{pk}, c_1, c_2)$. Output $c_1 + c_2 \bmod x_0$.
- $\mathsf{Mult}_t^a(\mathsf{pk}, c_1, c_2)$. Output $\mathsf{Convert}_t(\mathsf{pk}, t^{a-1} \cdot c_1 \cdot c_2)$
- $\mathsf{Eval}_t^a(\mathsf{pk}, f, c)$. Output the homomorphic evaluation of the ciphertext c with the polynomial f by operations defined above.

A ciphertext $c = q \cdot p^2 + (t^a r^* + m) \cdot \lfloor p/t^a \rceil + r$ has two kinds of errors, r and r^*. We call c a ciphertext with noise (ρ, ρ^*) if $|r| < 2^\rho$ and $|r^*| < 2^{\rho^*}$. Lemma 5.3.1 shows the correctness of Add_t^a and Mult_t^a as well as analysis on noise growth during the homomorphic operations. We notice that the proof of Lemma 3 is definitely not new one compared to the proof in [CLT14]; we only generalize it from the case of $t = 2$ to the case of arbitrary prime t.

Lemma 5.3.1. *(Noise growth analysis) Let c_1 and c_2 be ciphertexts with noise (ρ_1, ρ_1^*) and (ρ_2, ρ_2^*), respectively. Let $\rho = \mathsf{max}(\rho_1, \rho_2)$ and $\rho^* = \mathsf{max}(\rho_1^*, \rho_2^*)$. Then,*

- $\mathsf{Add}_t^a(\mathsf{pk}, c_1, c_2)$ *is a ciphertext with noise* $(\rho + 2, \rho^* + 1)$

- $\mathsf{Mult}_t^a(\mathsf{pk}, c_1, c_2)$ *is a ciphertext with noise* $(\rho + \rho^* + a \log t + 8, \log \Theta)$

Proof. Let c_1, c_2 as below.

$$c_1 = q_1 \cdot p^2 + \lfloor p/t^a \rceil \cdot (m_1 + t^a r_1^*) + r_1,$$

$$c_2 = q_2 \cdot p^2 + \lfloor p/t^a \rceil \cdot (m_2 + t^a r_2^*) + r_2.$$

Then addtion of c_1 and c_2 is

$$\begin{aligned} c_1 + c_2 &= (q_1 + q_2) \cdot p^2 + \lfloor p/t^a \rceil \cdot ([m_1 + m_2]_{t^a} \\ &\quad + t^a (r_1^* + r_2^* + 1/0)) + r_1 + r_2 \\ &= q_3 \cdot p^2 + \lfloor p/t^a \rceil \cdot (m_3 + t^a r_3^*) + r_3 \end{aligned}$$

for $r_3^* < 2^{\rho_1^*} + 2^{\rho_2^*} + 1$ and $r_3 < 2^{\rho_1} + 2^{\rho_2}$. The ciphertext of $[m_1 + m_2]_{2^a}$ is $c_3 = [c_1 + c_2]_{x_0} = c_1 + c_2 - k \cdot x_0$ for $k \in \{0, 1\}$ since $c_1,\ c_2 < x_0$. Therefore, $c_3 \leftarrow \mathsf{Add}_t^a(\mathsf{pk}, c_1, c_2)$ is a ciphertext $c_3 = q \cdot p^2 + \lfloor p/t^a \rceil (m + t^a r^*) + r$ satisfying $r^* < 2^{\rho_1^*} + 2^{\rho_2^*} + 1$ and $r < 2^{\rho_1} + 2^{\rho_2} + 2^{\rho_0}$.

Let c_1, c_2 as defined above, and $k,\ l$ be integers such that $\lfloor p/t^a \rceil = (p-k)/t^a$ and $\lfloor p^2/t^a \rceil = (p^2 - l)/t^a$. Then the following equation holds,

$$\begin{aligned} c_3 &= q_3 \cdot p^2 + ((p-k)^2/t^a)(m_1 + t^a r_1^*)(m_2 + t^a r_2^*) + R \\ &= q_3 \cdot p^2 + ((p^2 - l)/t^a) \cdot (m_1 m_2 \bmod t^a) + R + R' \\ &= q_3 \cdot p^2 + \left\lfloor p^2/t^a \right\rceil \cdot (m_1 m_2 \bmod t^a) + r_3 \end{aligned}$$

where $|R| < 3{\cdot}2^{\eta}{\cdot}t^{a}{\cdot}2^{\rho^*+\rho}$ and $|R'| < 2{\cdot}2^{\eta}{\cdot}t^{2a}{\cdot}2^{2\rho^*}+t^{2a}{\cdot}2^{2\rho^*} < 3{\cdot}2^{\eta}{\cdot}t^{2a}{\cdot}2^{2\rho^*}$.

Therefore, the inequality $|r_3| < 6\cdot 2^{\eta+a\log t+\rho+\rho^*}$ holds when assuming $a\log t+\rho^* < \rho$.

Now we will analyze the error of ciphertext after processing $\mathsf{Convert}$ procedure. We followed the proof of lemma 1 in [CLT14].

Let $\lceil \log r_3 \rfloor = \rho_3 < \eta + 2a\log t + \rho + \rho^* + 3$ and $c \leftarrow \mathsf{Convert}(c_3/t)$, then from the equation

$$\boldsymbol{\sigma} = p^2\cdot\boldsymbol{q} + \boldsymbol{r} + \lfloor \boldsymbol{s}'\cdot\frac{p}{2^{\eta+1}}\rceil$$

Let $\boldsymbol{c}' = \mathsf{BitDecomp}_{\eta}(\boldsymbol{c})$, then we have:

$$c = 2\langle\boldsymbol{\sigma},\boldsymbol{c}'\rangle = 2p^2\cdot\langle\boldsymbol{q},\boldsymbol{c}'\rangle + 2\langle\boldsymbol{r},\boldsymbol{c}'\rangle + 2\langle\lfloor\boldsymbol{s}'\cdot\frac{p}{2^{\eta+1}}\rceil,\boldsymbol{c}'\rangle.$$

since the components of $\boldsymbol{c}'$ are bits,

$$2\langle\lfloor\boldsymbol{s}'\cdot\frac{p}{2^{\eta+1}}\rceil,\boldsymbol{c}'\rangle = \langle\frac{p}{2^{\eta}}\cdot\boldsymbol{s}',\boldsymbol{c}'\rangle + \nu_2 = \frac{p}{2^{\eta}}\langle\boldsymbol{s}',\boldsymbol{c}'\rangle + \nu_2,$$

where $|\nu_2| < \Theta\cdot\eta$. From the definition of $\mathsf{BitDecomp}$ and $\mathsf{PowersofTwo}$, we have $\langle\boldsymbol{s}',\boldsymbol{c}'\rangle = \langle\boldsymbol{s},\boldsymbol{c}\rangle \bmod 2^{\eta} = \langle\boldsymbol{s},\boldsymbol{c}\rangle + q_2\cdot 2^{\eta}$. Moreover

$$\langle\boldsymbol{s},\boldsymbol{c}\rangle = \sum s_i\left\lfloor\frac{c_3}{t}\cdot z_i\right\rceil + \Delta\cdot 2^{\eta} = \sum\frac{s_i\cdot c_3\cdot z_i}{t} + \delta_1 + \Delta\cdot 2^{\eta}$$

$$= \frac{c_3}{t}\cdot\langle\boldsymbol{s},\boldsymbol{z}\rangle + \delta_1 + \Delta\cdot 2^{\eta},$$

for some $\Delta\in\mathbb{Z}$ and $|\delta_1| \leqslant \Theta/2$. Using $\langle\boldsymbol{s},\boldsymbol{z}\rangle = 2^{\eta}\cdot t/p^2 - \epsilon - \mu\cdot 2^{\eta}\cdot t$ for some $\mu\in\mathbb{Z}$, and $c_3 = r_3 + \lfloor p^2/t^a\rceil\cdot m + q_3\cdot p^2$, this gives

$$\langle \boldsymbol{s}, \boldsymbol{c} \rangle = q_3 \cdot 2^\eta + \frac{2^\eta}{t^a} m - \frac{\ell \cdot 2^\eta}{p^2 \cdot t^a} m + \frac{2^\eta}{p^2} r_3 - \frac{c_3}{t} \epsilon + \delta_1 + (\Delta - c_3 \cdot \mu) \cdot 2^\eta.$$

Therefore we can write

$$\langle \boldsymbol{s}, \boldsymbol{c} \rangle = q_1 \cdot 2^\eta + m \cdot \frac{2^\eta}{t^a} + r^*$$

for some $r^* \in \mathbb{Z}$, with $|r^*| \leqslant 2^{\rho_3 - \eta + 3}$. Now we get an equation below:

$$2 \left\langle \left\lfloor \frac{p}{2^{\eta+1}} \cdot \boldsymbol{s}' \right\rceil, \boldsymbol{c}' \right\rangle = q_4 \cdot p + m \cdot \frac{p}{t^a} + r^* \cdot \frac{p}{2^\eta} + \nu_2$$

with $|q_4| \leqslant \Theta$; namely the components of $(p/2^{\eta+1}) \cdot \boldsymbol{s}'$ are smaller than p and $\boldsymbol{c}'$ is a binary vector. This gives

$$2 \left\langle \left\lfloor \frac{p}{2^{\eta+1}} \cdot \boldsymbol{s}' \right\rceil, \boldsymbol{c}' \right\rangle = (t^a q_4 + m) \cdot \left\lfloor \frac{p}{t^a} \right\rfloor + r_2^*$$

with $|r_2^*| \leqslant 2^{\rho_3 - \eta + 4}$. Then we obtain

$$\begin{aligned} c &= 2p^2 \cdot \langle \boldsymbol{q}, \boldsymbol{c}' \rangle + 2 \langle \boldsymbol{r}, \boldsymbol{c}' \rangle + (t^a q_4 + m) \cdot \left\lfloor \frac{p}{t^a} \right\rfloor + r_2^* \\ &= 2q'' \cdot p^2 + (t^a q_4 + m) \cdot \left\lfloor \frac{p}{t^a} \right\rfloor + r' \end{aligned}$$

where $|r'| \leqslant |r_2^*| + \eta \Theta 2^{\rho+1} \leqslant 2^{\rho_3 - \eta + 4} + \eta \Theta 2^{\rho+1} < 2^{a \log t + \rho + \rho^* + 7} + \eta \Theta 2^{\lambda+1}$. Therefore, c is ciphertext with noise $(\rho + \rho^* + a \log t + 8, \log \Theta)$ if $a \log t + \rho + \rho^* + 5 > \log \eta + \log \Theta + \lambda$. $\square$

5.3.3 Homomorphic Digit Extraction for CLT scheme

During homomorphic digit extraction, we use various message spaces from $\mathbb{Z}_t$ to $\mathbb{Z}_{t^{n+1}}$. Let $\mathrm{Enc}_{\mathbb{Z}_{t^k}}(m)$ be a ciphertext of m with message space $\mathbb{Z}_{t^k}$ in the form of $q \cdot p^2 + \lfloor p/t^k \rceil \cdot (m + t^k \cdot r^*) + r$. The following algorithm represents homomorphic digit extraction with CLT scheme. Note that the polynomial $F_{t,n}$ is explained in the section 5.2.2.

HomExt Algorithm (Homomorphic digit extraction):

```
    **Input**: A ciphertext $c$ of message space $\mathbb{Z}_{t^{n+1}}$

    **Compute** $c_{i,j}$ for $0 \leqslant i \leqslant n$, $0 \leqslant j \leqslant n-i$ :

        $c_{0,0} \leftarrow c$
        For $0 \leqslant i \leqslant n-1$,
            For $0 \leqslant j \leqslant n-i-1$,
            $c_{i,j+1} \leftarrow \mathsf{Eval}_t^{n-i+1}(\mathsf{pk}, F_{t,n}, c_{i,j})$
        $c_{i+1,0} \leftarrow c_{0,0} - c_{0,i+1} - c_{1,i} - \cdots - c_{i,1}$

    **Output**: $c_{n,0}$
```

Figure 5.3: HomExt Algorithm

To understand the above algorithm, we need to check when we can change the message space for a fixed ciphertext. In the scale invariant HE over the integer, since PSDil and PSCon are trivial mapping, $\mathrm{Enc}_{\mathbb{Z}_{t^k}}(m)$ can be treated as $\mathrm{Enc}_{\mathbb{Z}_{t^\ell}}(t^{\ell-k}m)$ for $k < \ell$. Conversely, if m is a multiple of $t^{\ell-k}$, $\mathrm{Enc}_{\mathbb{Z}_{t^\ell}}(m)$ can be treated as $\mathrm{Enc}_{\mathbb{Z}_{t^k}}(m/t^{\ell-k})$.

The following lemma shows the correctness of the proposed homomorphic digit extraction algorithm.

Lemma 5.3.2. *(Correctness of* HomExt*) For given* $m = b_{0,0}$*, define* $b_{i,j}$*:*

$$b_{i,0} = (b_{0,0} - \sum_{j=0}^{i-1} t^j \cdot b_{j,i-j} \bmod t^{n+1})/t^i \textit{ for } 1 \leqslant i \leqslant n,$$

$$b_{i,j+1} = F_{t,n}(b_{i,j}) \textit{ for } 0 \leqslant i < n,\ 0 \leqslant j \leqslant n-i.$$

When we set $c_0 = Enc_{\mathbb{Z}_{t^{n+1}}}(b_{0,0})$ *and define* $(c_{i,j})$ *following the* HomExt *algorithm, then* $c_{i,0} = Enc_{\mathbb{Z}_{t^{n-i+1}}}(b_{i,0})$ *for* $0 \leqslant i \leqslant n$ *so that the equality* $c_{n,0} = Enc_{\mathbb{Z}_t}(m^{(t)}\langle n\rangle)$ *holds.*

Proof. We use induction on i. The statement is clear when $i = 0$. Suppose the proposition is true for $i < m$. Then we have

$$\begin{aligned}
c_{m,0} &= c_{0,0} - \sum_{j=0}^{m-1} c_{j,m-j} \\
&= c_0 - \sum_{j=0}^{m-1} \mathrm{Enc}_{\mathbb{Z}_{t^{n-j+1}}}\left(F_{t,n}^{m-j}(b_{j,0})\right) \\
&= c_0 - \sum_{j=0}^{m-1} \mathrm{Enc}_{\mathbb{Z}_{t^{n+1}}}\left(t^j F_{t,n}^{m-j}(b_{j,0})\right) \\
&= \mathrm{Enc}_{\mathbb{Z}_{t^{n+1}}}\left(b_{0,0} - \sum_{j=0}^{m-1} t^j F_{t,n}^{m-j}(b_{j,0})\right) \\
&= \mathrm{Enc}_{\mathbb{Z}_{t^{n+1}}}\left(b_{0,0} - \sum_{j=0}^{m-1} t^j b_{j,m-j} \bmod t^{n+1}\right) \\
&= \mathrm{Enc}_{\mathbb{Z}_{t^{n+1}}}\left(t^m b_{m,0}\right) = \mathrm{Enc}_{\mathbb{Z}_{t^{n+1-m}}}(b_{m,0}).
\end{aligned}$$

Therefore, this lemma holds for any positive $i \leqslant n$, and this means $c_{n,0} = \mathrm{Enc}_{\mathbb{Z}_t}(b_{n,0}) = \mathrm{Enc}_{\mathbb{Z}_t}(m^{(t)}\langle n\rangle)$, so this lemma shows the correctness of our bootstrapping procedure. □

To sum up, we can homomorphically evaluate digit-extraction, so CLT scheme satisfies all conditions in section 5.2.3; namely, our method can be applied to CLT scheme. Now we introduce the explicit explanation of the application of our method on the scheme.

5.3.4 Our Method on the CLT scheme

For an η-bit odd integer p and integer q_0 in $[0, 2^\gamma/p^2)$, we define the set

$$\mathcal{D}^\rho_{p,q_0} = \{q \leftarrow [0, q_0), r \leftarrow (-2^\rho, 2^\rho) : \textsf{Output } p^2 q + r\}.$$

- $\textsf{KeyGen}^*_t(1^\lambda)$. Generate $\textsf{pk} = \{x_0, x_1, \cdots, x_\tau, \boldsymbol{\sigma}, \boldsymbol{z}\}$ as in Section 5.3.1. Choose a random a Θ'-bit vector $\boldsymbol{s'}$ with Hamming weight θ', and let $S' = \{i : s'_i = 1\}$. Choose a random integer $u_i \in [0, t^{\kappa+1})$ such that $\sum_{i \in S'} u_i = \lfloor t^{\kappa+1}/p \rceil$. For $n = \lceil \log_t \theta' \rceil + 3$, generate

  $$v_i = q_i \cdot p^2 + \left\lfloor \frac{p}{t^{n+1}} \right\rceil \cdot s'_i + r_i$$

  and $v_0 = q \cdot p^2 + \lfloor \frac{p}{t^{n+1}} \rceil \cdot \frac{t^n}{2} + r$, where $q, q_i \in [0, q_0)$ and $r, r_i \in (-2^\rho, 2^\rho)$ for $1 \leqslant i \leqslant \Theta'$. The secret key is $\textsf{sk} = \{p\}$ and the public key is $\textsf{pk}^* = \{\textsf{pk}, \boldsymbol{u}, \boldsymbol{v}\}$.

- $\textsf{HomSum}_t(c, \boldsymbol{u}, \boldsymbol{v})$. Generate $w_0 = 1$, $w_i = \lfloor c \cdot u_i / t^{\kappa - n} \rceil \bmod t^{n+1}$ for $n = \lceil \log_t \theta' \rceil + 3$, and output

  $$c' \leftarrow \sum_{i=0}^{\Theta'} v_i \cdot w_i \bmod x_0.$$

- $\textsf{Bootstrap}_t(c, \boldsymbol{u}, \boldsymbol{v})$. For $c' \leftarrow \textsf{HomSum}_t(c, \boldsymbol{u}, \boldsymbol{v})$, output the new ciphertext $\mathsf{HomExt}(c')$.

Conditions on the Parameters

The security of the squashed scheme has been studied in [vDGHV10, CMNT11, CNT12]. Here, λ is a security parameter, and γ is as in the previous section.

- $n = \lceil \log_t \theta \rceil + 3$ for the correctness of squashed decryption function,
- $\kappa' > (\gamma+\lambda)/\log t$ for the correctness of squashed decryption function,
- $\Theta'^2 \geqslant \gamma \cdot \omega(\log \lambda)$ to avoid a lattice-based attack on the subset sum problem [CMNT11, CNT12],
- $\binom{\Theta'}{\theta'/2} \geqslant 2^\lambda$ to avoid an attack on the sparse subset sum problem [BIWX11].

5.3.5 Analysis of Proposed Bootstrapping Method

Our analysis can be more tight for binary message space, since the evaluation of the polynomial $F_{t,n}(X)$ for $t > 2$ is relatively hard due to its complicated form. In this section, we first check the correctness of our bootstrapping method and analyze the noise growth during bootstrapping procedure. Also, we compute the number of homomorphic multiplications in our method, which directly implies the efficiency of our method.

Theorem 5.3.1. *For $c^* \leftarrow \mathsf{Bootstrap}(c, \boldsymbol{u}, \boldsymbol{v})$, c^* is ciphertext with noise*

$$(\rho_2, \rho_2^*) = (\rho + \delta + n \log t(\log t + \log \Theta + 8)(1+\epsilon),\ \log \Theta + n)\,,$$

and ciphertexts c and c^* have same message if ρ^* and ρ_2^* is smaller than p. Here $\epsilon = \left(\frac{n+1}{2} \cdot \log t + t + \frac{n+2}{\log t}\right) / (\log t + \log \Theta + 8)$ and $\delta = (n+1)\log t + \log(\Theta' + 1)$.

Proof. (HomSum) Note that $v_i = q_i \cdot p^2 + \lfloor p/t^{n+1} \rfloor \cdot s_i + r_i$ and $v_0 = q \cdot p^2 + \lfloor p/t^{n+1} \rfloor \cdot \lfloor t^n/2 \rfloor + r$ with $q,\ q_i \in [0, q_0)$ and $r,\ r_i \in (-2^\rho, 2^\rho)$ for $1 \leqslant i \leqslant \Theta'$. So if $c_{0,0} \leftarrow \mathsf{HomSum}(c, \boldsymbol{u}, \boldsymbol{v})$, then $c_{0,0} = q' \cdot p^2 + \lfloor p/t^{n+1} \rfloor \cdot ((\sum s_i w_i + \lfloor t^n/2 \rfloor) \bmod t^{n+1} + r^* t^{n+1}) + r'$ for $|r'| = |\sum_{i=1}^{\Theta'} w_i r_i + r| < (\Theta' + 1)2^{\rho + (n+1)\log t}$ and $|r^*| \leqslant \Theta'$. Therefore, $c_{0,0}$ is a ciphertext with noise $(\rho_1, \rho_1^*) = (\rho + (n+1)\log t + \log(\Theta'+1),\ \log \Theta')$ whose message space is $\mathbb{Z}_{t^{n+1}}$.

(HomExt) Let $c_{i,j}$ is a ciphertext with noise $(\rho_{i,j}, \rho_{i,j}^*)$, then the equations $\rho_{0,0} = \rho + (n+1)\log t + \log(\Theta' + 1)$ and $\rho_{0,0}^* = \log \Theta'$ holds by above HomSum procedure. By applying Lemma 5.3.1, we can set

$$\rho_{i,0} = \mathsf{max}\{\rho_{0,i}, \cdots, \rho_{i-1,1}\} + 2i,\ \rho_{i,0}^* = \log \Theta + i \log t$$

for $1 \leqslant i \leqslant n$. First, we will show the equality

$$\mathsf{max}\{\rho_{0,i+1}, \cdots, \rho_{i,1}\} = \rho_{i,1}$$

holds for $0 \leqslant i \leqslant n-1$. Since $c_{j,i-j+1} = \mathsf{Eval}_t^{n-j+1}(\mathsf{pk}, F_{t,n}, c_{j,i-j})$ for $0 \leqslant j \leqslant i$, it is sufficient to compare noise increase of $c_{j,i-j}$ after Mult_t^a. For $1 \leqslant j \leqslant i-1$, the increase of first noise of $c_{j,i-j}$ is less than or equal to $\log \Theta + (n+1)\log t + 8$, and the increase of noise of $c_{i,0}$ is $\rho_{i,0}^* + (n-i+1)\log t + 8 = \log \Theta + (n+1)\log t + 8$. Therefore, the equality $\mathsf{max}\{\rho_{0,i+1}, \cdots, \rho_{i,1}\} = \rho_{i,1}$ holds and we can get

$$\rho_{i,0} = \rho_{i-1,1} + 2i.$$

Second, we will analyze the noise increase in while evaluating $F_{t,n}$. Note that the polynomial $F_{t,n}$ is of degree t and its coefficients are bounded by t^{n+1}. Then, we can regard each term of $F_{t,n}$ is contained by at most t times of multiplications, so we get $\rho_{i-1,1} = \rho_{i-1,0} + \lceil \log t \rceil \cdot (\log t \cdot (n-i+2) + \log \Theta + 8) + t\lceil \log t \rceil$. Now, we obtain a recursion formula:

$$\rho_{i,0} = \rho_{i-1,0} + \lceil \log t \rceil \cdot (\log t \cdot (n-i+2) + \log \Theta + 8)$$

$$+t\lceil \log t \rceil + 2i.$$

The consequence of the recursion formula is

$$\begin{aligned} \rho_{n,0} &= \rho_{0,0} + \log^2 t \cdot \frac{n^2+3n}{2} + n \log t(\log \Theta + 8) \\ &\quad +nt \log t + n^2 + 2n \\ &= \rho_{0,0} + n \log t(\log t + \log \Theta + 8)(1+\epsilon) \end{aligned}$$

for $\epsilon = \left(\frac{n+1}{2} \cdot \log t + t + \frac{n+2}{\log t}\right) / (\log t + \log \Theta + 8)$.
(Correctness) Let $\mathrm{Enc}^{\rho,\rho^*}_{\mathbb{Z}_{t^k}}(m)$ be a set of ciphertext with message $m \in \mathcal{M} = \mathbb{Z}_{t^k}$ and error (ρ, ρ^*). Then our bootstrapping process can be described as below diagram.

$$\begin{tikzcd}
Enc^{\rho,\log\Theta}_{\mathbb{Z}_t}(m) \arrow[r, "HomExt"] \arrow[d, Rightarrow, "Bootstrap"] & Enc^{\rho_1,\rho_1^*}_{\mathbb{Z}_{t^{n+1}}}(\sum s_i w_i + \lfloor t^n/2 \rfloor \bmod t^{n+1}) \arrow[d, "HomMsb"] \\
Enc^{\rho_2,\rho_2^*}_{\mathbb{Z}_t}(m) \arrow[r, equal] & Enc^{\rho_2,\rho_2^*}_{\mathbb{Z}_t}((\sum s_i w_i + \lfloor t^n/2 \rfloor \bmod t^{n+1})^{(t)}\langle n \rangle)
\end{tikzcd}$$

Top side of the diagram was proved in 1. HomSum. Also, Lemma 5.3.2

and 2. HomExt exactly signify the right side of the diagram, and the discussion in Section 5.3.3 shows the equality $m = (\sum s_i w_i + \lfloor t^n/2 \rfloor \bmod t^{n+1})^{(t)}\langle n \rangle$ holds so that bottom side of the diagram is proved.

□

Since the first noise grows approximately $(\log t + \log \Theta + 8)$ per each multiplication, we can think of the degree of Bootstrap function is

$$2^{n \log t(1+\epsilon)+\epsilon_1} = O(\lambda^{1+\epsilon+\frac{\epsilon_1}{n \log t}}) = O(\lambda^{1+\epsilon_2})$$

where $\epsilon_1 = \{(n+1)\log t + \log(\Theta' + 1)\}/(\log t + \log \Theta + 8)$.

Theorem 5.3.2. *The number of multiplication operations in our bootstrapping algorithm is $O(n(n+1)/2) = O(\log^2 \lambda)$.*

Proof. We will treat t as a constant, so the number of multiplication while evaluating polynomial $F_{t,n}$ is constant. The number of evaluation k is equal to $1+2+\cdots+n = n(n+1)/2$; thus, the number of multiplication operations is $O(n(n+1)/2)$. □

As a result, in our bootstrapping method, the number of homomorphic multiplications is $O(\log^2 \lambda)$ and multiplicative degree is $O(\lambda^{1+\epsilon})$. Comparing to the previous methods including the result in [NK15], $\tilde{O}(\lambda^4)$ multiplications, our method shows significantly improved result within the framework of efficiency. In addition to theoretical analysis, we will explain the implementation result of our bootstrapping method applying to the CLT scheme in next section.

5.4 Implementation Result

While implementing our bootstrapping method, we use word decomposition and the powers of word instead of BitDecomp and PowersofTwo with word size $w = 32$. Moreover, in order to use a public key of reasonable size, we compress the ciphertext using the same method as in [CMNT11]. We implement our bootstrapping method and check the running time of Bootstrap. Furthermore, for precise comparison with other FHEs, we implement the homomorphic evaluation of the AES-128 circuit, which has emerged lately as a standard homomorphic evaluation circuit. We encrypt messages bit-wisely while AES evaultion as in [CLT14].

1. **Parameters ($\ell = 500, \lambda = 72$).**
   - AGCD parameters: $\eta = 192$, $\gamma = 3.8 \times 10^5$, $\rho = 52$
   - Convert parameters: $\Theta = 1500$, $\theta = 100$
   - Bootstrap parameters: $\Theta' = 8000$, $\theta' = 15$
2. **Efficiency.**
   - The number of Add: $8000 + 10$
   - The number of Mult: 8
   - Error size after bootstrapping: 122 bit
3. **AES evaluation.**
   - Bootstrap Time : 6.7×128 sec (128 ciphertexts)
   - SubByte Time : 128 sec

- **Total AES** Time : 4020 sec
- **Relative** Time (**Total AES** Time / ℓ): 8 sec

Remark 5.4.1. Implementations of our bootstrapping method and homomorphic evaluation of AES circuit were progressed on a desktop with eight core Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz processors and 16GB RAM using C++ and GMP 6.0.0[G+96].

This result shows that bootstrapping process can be done with only 8 number of homomorphic multiplications. Our bootstrapping procedure for one ciphertext takes about 6 seconds. This result is faster than previous results in FHE over the integers [CMNT11, vDGHV10, CCK+13], and also compatable with the result in [HS15], 320 seconds for 16000-bit message space. Comparing to the results of homomorphic evaluation of AES circuit in [CCK+13, CLT14], 13 minutes and 23 seconds per block at security level $\lambda = 72$, homomorphic evaluation of AES circuit applying our bootstrapping method takes 8 seconds per block on a 8-core machine at 3.4 GHz for the same security level. This implementation of homomorphic evaluation of AES circuit is the first case that using small depth parameter with bootstrapping can be faster than using large depth without bootstrapping.

Chapter 6

Logistic Regression on Large Encrypted Data

In this chapter, we present an efficient algorithm for logistic regression on encrypted data, and demonstrate its practical feasibility against realistic size datasets, for the first time to the best of our knowledge. We evaluate our algorithm against a real, private financial dataset consisting of 422,108 samples over 200 features. Our implementation successfully learned a quality model in $\sim$17 hours on a single machine, where we tested it against a validation set of 844,217 samples and obtained a sufficient Kolmogorov Smirnov statistic value of 50.84. The performance is "only" two to three orders of magnitude slower than that of plaintext learning, which is encouraging, considering the inherent computational overhead of HEs. We also executed our algorithm on the public MNIST dataset for more detailed evaluation, and it took $\sim$2 hours to learn an encrypted model with 96.4% accuracy. Below we describe the principal techniques used in our efficient logistic regression algorithm on a large encrypted dataset.

Approximate HE. Our algorithm leverages the recent advances of (word-wise) *approximate* HE schemes and the *approximate* bootstrapping method to reduce the computational overhead. The approximate HE can quickly compute approximated results of complex operations, avoiding the bit-manipulation overhead. Similarly, the approximate bootstrapping can efficiently bootstrap a ciphertext at the cost of additional approximation noise.

While both the approximate HE and the approximate bootstrapping can reduce the computational overheads, they have the disadvantage of introducing an additional noise for each computation step. Even if it is small, the noise may affect the overall machine learning performance (e.g., the convergence rate and accuracy), but it had not been clear how critical the small noise is. We empirically show that the additional noise is not significant to deteriorate the accuracy of a learned model and the convergence rate. Indeed, our finding is consistent with the results of low-precision training approaches in the literature [DSFRO17, ZLK$^+$16, GAGN15, CBD14] which have also empirically shown that small approximation (round-off) errors due to the low-precision are manageable.

HE-Optimized, Vectorized Logistic Regression Algorithm. The approximate HE scheme we use also supports the packing method [CKKS17] which can further reduce the computation overhead. In the packed HEs, a single ciphertext represents an encryption of a vector of plaintexts, and ciphertext operations correspond to point-wise operations on plaintext vectors, so-called single instruction multiple data (SIMD) operations.

To maximize the benefits of the packed scheme, we vectorize our logistic regression algorithm to utilize the SIMD operations as much as possible. For example, the inner product operation is represented as a SIMD-multiplication followed by a sequence of rotations and SIMD-additions. Moreover, we carefully tune the vectorized algorithm to minimize redundant computations caused by the use of the SIMD operations, reduce the depth of nested multiplications, and minimize the approximation noises by reordering operations.

Parallelized Bootstrapping. One of the most expensive operations of HEs is the bootstrapping operation (even with the approximate bootstrapping method). This operation needs to be periodically executed during the entire computation. In logistic regression, for example, it should be executed every few iterations, and dominates the overall training time. It is critical for performance to optimize the bootstrapping operation.

We design our algorithm to parallelize the bootstrapping operation. It splits a ciphertext into multiple smaller chunks and executes bootstrapping on each chunk in parallel, achieving a significant speedup of the overall performance. Moreover, we carefully design the packing of training data (see below) so that our algorithm continues to use the chunks without merging them in the next training iterations, which additionally saves time it takes to reconstruct a ciphertext from the chucks.

HE-Optimized, Efficient Partition of Training Data. As mentioned above, we pack multiple plaintexts in a single ciphertext, and it is critical for performance how to pack (i.e., partition) the training dataset. The training data can be seen as an $n \times m$ matrix with n samples and m features. A naive encoding would pack each row (or column) into a ciphertext, resulting in a total of n (or m) ciphertexts. This encoding, however, is not efficient, since it either does not utilize the maximum capacity of the ciphertexts, or requires too much capacity, increasing the computation overhead drastically.

We design an efficient partition of training data in which a sub $n' \times m'$ matrix is packed into a single ciphertext, where the size of the matrix is set to the maximum capacity of each ciphertext, and m' is set to align with the aforementioned parallelization technique, avoiding an extra overhead of the ciphertext reconstruction.

Approximating Non-Polynomial Functions As mentioned earlier, non-polynomial functions are computationally expensive in HEs. We mitigate this performance overhead issue by approximating them as polynomials. A sigmoid function, for example, is replaced by its polynomial approximation in our training algorithm. Note that, however, an approximation at a point such as Taylor expansion is not adequate for logistic regression (and machine learning in general) since the deviation could be too large at other points. Instead, we use an interval approximation whose difference on the interval is minimized in terms of least squares. Combined with a proper input normalization, the interval approximation has provided sufficient precision for logistic regression in our experiment.

6.1 Basis of Logistic Regression

Logistic regression is a machine learning algorithm to learn a model for classification. We focus on the binary classification throughout this paper for the simplicity of the presentation. In logistic regression, we consider the following model:

$$\log\left[\frac{\Pr(Y=0|X=\boldsymbol{x})}{\Pr(Y=1|X=\boldsymbol{x})}\right]=\langle \boldsymbol{w},(1,\boldsymbol{x})\rangle$$

where:*

$$\Pr(Y=1|X=\boldsymbol{x})=\frac{1}{1+e^{-\langle \boldsymbol{w},(1,\boldsymbol{x})\rangle}}$$
$$\Pr(Y=0|X=\boldsymbol{x})=\frac{e^{-\langle \boldsymbol{w},(1,\boldsymbol{x})\rangle}}{1+e^{-\langle \boldsymbol{w},(1,\boldsymbol{x})\rangle}}$$

for an input vector X of d features, a class Y, a weight vector $\boldsymbol{w}\in\mathbb{R}^{d+1}$.

The goal of the logistic regression training, given m samples $\{(\boldsymbol{x}_i,y_i)\}_m$, is to find a weight vector $\boldsymbol{w}$ that minimizes the negative log likelihood function $\ell(\boldsymbol{w})=-\frac{1}{m}\cdot\log L(\boldsymbol{w})$, where:

$$L(\boldsymbol{w})=\prod_{i=1}^{m}h_{\boldsymbol{w}}(\boldsymbol{x}_i)^{y_i}\cdot(1-h_{\boldsymbol{w}}(\boldsymbol{x}_i))^{1-y_i}$$

with $h_{\boldsymbol{w}}(\boldsymbol{x}_i)=\sigma(\langle \boldsymbol{w},(1,\boldsymbol{x}_i)\rangle)$ and $\sigma(x)=1/(1+e^{-x})$. Since $\ell(\boldsymbol{w})$ is convex, we can use the gradient descent method to find the vector $\boldsymbol{w}$ that minimizes $\ell(\boldsymbol{w})$. The gradient descent method for logistic regression is formulated as the following recurrence relation:

$$\boldsymbol{w}_{i+1} \quad = \quad \boldsymbol{w}_i-\alpha\cdot\Delta_w\ell(\boldsymbol{w}_i)$$

for a learning rate α. The gradient of the log likelihood function is as follows:

$$\Delta_w\ell(\boldsymbol{w})=-\frac{1}{m}\sum_{i=1}^{m}\sigma(-\langle \boldsymbol{z}_i,\boldsymbol{w}\rangle)\cdot\boldsymbol{z}_i$$

*We write $\langle\cdot,\cdot\rangle$ for the inner product, and $(1,\boldsymbol{x})$ for a vector extended with 1 from $\boldsymbol{x}$.

where $\boldsymbol{z}_i = y'_i \cdot (1, \boldsymbol{x}_i) \in \mathbb{R}^{d+1}$, and $y'_i = 2y_i - 1 \in \{-1, 1\}$.

6.2 Logistic Regression on Encrypted Data

In this section, we explain our algorithm for efficient logistic regression on encrypted data. We first present a baseline (plaintext) algorithm of the logistic regression training, designed to be friendly to homomorphic evaluation (Section 6.2.1). Then we explain how to optimize the baseline algorithm to be efficiently evaluated in HEs.

6.2.1 HE-friendly Logistic Regression Algorithm

We first explain our baseline algorithm of the logistic regression training, as shown in Algorithm 3, that we will further optimize in the next section. We design the baseline algorithm to be friendly to homomorphic evaluation by avoiding the use of certain types of computations that are expensive in HEs.

Mini-Batch Gradient Descent. We adopt the mini-batch gradient descent method, where we set the mini-batch size according to the number of slots in a packed ciphertext. We do not consider the stochastic gradient descent method since it does not utilize the maximum capacity of the packed ciphertext. Also, we do not consider the full-batch gradient descent method since it requires too many and/or large ciphertexts for each iteration when the training dataset is large.

Nesterov Accelerated Gradient Optimizer. We adopt Nesterov accelerated gradient (NAG) as the gradient descent optimization method. We

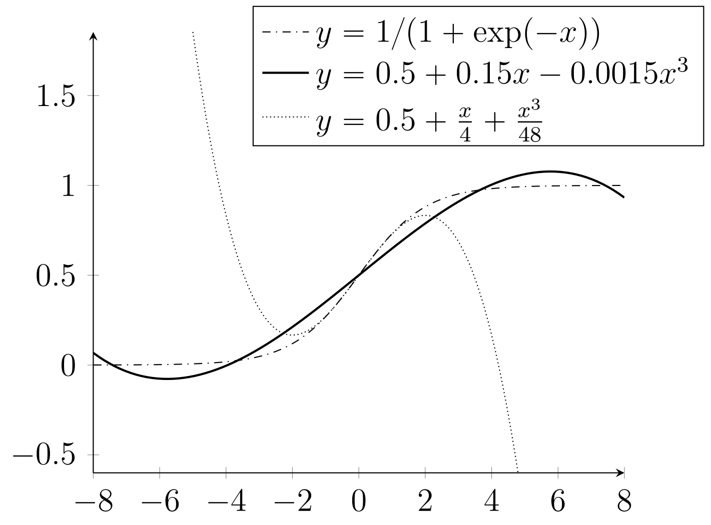

Figure 6.1: Sigmoid (the first) and its two approximations using the least squares fitting method (the second) and the Taylor expansion (the third).

choose NAG among the various optimization methods, since it provides decent optimization performance without using the division operation that is expensive in HEs. The NAG can be formulated as follows:

$$\begin{aligned} w_{i+1} &= v_i - \gamma \cdot \Delta_w \ell(v_i) \\ v_{i+1} &= (1-\eta) \cdot w_{i+1} + \eta \cdot w_i \end{aligned}$$

where w_i and v_i are two weight vectors to be updated for each iteration i, $\Delta_w \ell(v_i)$ is the gradient of the log likelihood function (as given in Section 6.1), and γ and η are parameters.

Polynomial Approximation of Activation Function. An essential step of the logistic regression training is to apply an activation function, e.g., the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$. Since non-polynomials are very expensive to evaluate in HEs, we consider its (low-degree) polynomial approximation σ' as an alternative in our algorithm. We use the least squares fitting method to approximate the sigmoid function. The least squares fitting polynomial provides a sufficient approximation within the given interval. Figure 6.1, for example, plots the original sigmoid function, its least squares fitting polynomial (of degree 3) within the interval $[-8, 8]$, and its Taylor expansion (of degree 3) at the point $x = 0$. Note that the Taylor polynomial provides an accurate approximation only around the given point, while the least squares fitting polynomial provides a good approximation in a wider range.

```
Algorithm 3: HE-friendly logistic regression algorithm
   Data: Mini-batches of training data $\{Z_i\}$ where $Z_i \in \mathbb{R}^{m \times f}$ (i.e., the
         mini-batch size is $m$), parameters $\gamma$ and $\eta$, the number of
         iterations $K$, and a polynomial approximation of sigmoid $\sigma'$
   Result: Weight vectors $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{R}^f$
 1 Initialize weight vector: $\boldsymbol{w}, \boldsymbol{v} \leftarrow \mathbf{0}$ for $k$ in $[1..K]$ do
 2     Select a mini-batch $Z_i$ (in order, or at random);
 3     $\boldsymbol{a} = Z_i \cdot \boldsymbol{v}$;
 4     for $j$ in $[1..m]$ do
 5         $b_j = \sigma'(a_j)$
 6     end
 7     $\boldsymbol{\Delta} = \sum_{j=0}^{m-1} b_j \cdot Z_i[j]$ $\boldsymbol{w}^+ = \boldsymbol{v} - \gamma \cdot \boldsymbol{\Delta}$ $\boldsymbol{v}^+ = (1 - \eta) \cdot \boldsymbol{w}^+ + \eta \cdot \boldsymbol{w}$
        $\boldsymbol{w} = \boldsymbol{w}^+$, $\boldsymbol{v} = \boldsymbol{v}^+$
 8 end
```

Baseline Algorithm. The Algorithm 3 shows the resulting baseline algorithm. Note that each sample (row) $\boldsymbol{z}_i$ of the training data Z_i is structured by $\boldsymbol{z}_i = y_i' \cdot (1, \boldsymbol{x}_i) \in \mathbb{R}^f$, where $y_i' = 2y_i - 1 \in \{-1, 1\}$, and $\boldsymbol{x}_i$ and y_i are the original input samples and its class output, respectively (as described in Section 6.1).

6.2.2 HE-Optimized Logistic Regression Algorithm

Now we optimize the baseline algorithm (Algorithm 3) to be efficiently evaluated in HEs against large encrypted data. Specifically, we optimize the body of the main iteration loop (lines 2–7 of Algorithm 3). Conceptually, the optimization consists of two parts: vectorization using homomorphic SIMD operations, and fine-tuning the evaluation order. In this section, we explain the first part, which will result in the vectorized body of the main iteration loop as shown in Algorithm 6. We will explain the second part in the next section.

Let us first define some notations. For two matrices A and B, we write $A+B$ and $A \circ B$ to denote the addition and the element-wise multiplication (i.e., Hadamard product) of A and B, respectively. Also, we write $A^{\circ k}$ to denote the element-wise exponentiation, i.e., $A^{\circ k} = \{a_{i,j}^k\}$ for $A = \{a_{i,j}\}$.

Partition and Encryption of Training Data

Assume that the training data $\{x_{i,j}\}$ consists of n samples over $f-1$ features, throughout this section. This data can be seen as an $n \times f$ matrix Z including the target $\{y_i\}$ as follows:

$$Z = \begin{bmatrix} z[0][0], & z[0][1], & \cdots, & z[0][f-1] \\ z[1][0], & z[1][1], & \cdots, & z[1][f-1] \\ & \vdots & & \\ z[n-1][0], & z[n-1][1], & \cdots, & z[n-1][f-1] \end{bmatrix}$$

where $z[i][0] = y_i$ and $z[i][j+1] = y_i \cdot x_{i,j}$ for $0 \leqslant i < n$ and $0 \leqslant j < f-1$.[†]

We divide Z into multiple $m \times g$ sub-matrices $Z_{i,j}$ (for $0 \leqslant i < n/m$ and $0 \leqslant j < f/g$) as follows:

$$Z_{i,j} = \begin{bmatrix} z[mi][gj], & \cdots, & z[mi][gj+(g-1)] \\ z[mi+1][gj], & \cdots, & z[mi+1][gj+(g-1)] \\ & \vdots & \\ z[mi+(m-1)][gj], & \cdots, & z[mi+(m-1)][gj+(g-1)] \end{bmatrix}$$

$Z_{i,j}$ is supposed to be packed into a single ciphertext, and thus we set m and g in a way that utilizes the maximum ciphertext slots, $N/2$, that is, $m \times g = N/2$. Also, we set g to the same size of the partition of a weight vector for the bootstrapping parallelization, which in turn decides m, the size of a mini-batch block.

To encrypt $Z_{i,j}$ in a single ciphertext, we first represent it in a vector $\boldsymbol{p}_{i,j}$:

$$\boldsymbol{p}_{i,j}[k] = Z_{i,j}[\lfloor k/g \rfloor][k \bmod g] \quad (0 \leqslant k < g \cdot m)$$

and encrypt $\boldsymbol{p}_{i,j}$ using the scheme described in Section 4.1:

$$\texttt{encZ}[i][j] = \mathsf{encrypt}(\boldsymbol{p}_{i,j}; \Delta_z)$$

Note that we have nf/mg ciphertexts to encrypt the whole training data.

Partition and Encryption of Weight Vectors

We have two weight vectors, $\boldsymbol{w}$ and $\boldsymbol{v}$, of size f in our logistic regression algorithm due to the NAG optimization (as shown in Section 6.2.1). We divide each of them into multiple sub-vectors, $\boldsymbol{w}_i$ and $\boldsymbol{v}_i$, for the purpose

[†]We have $y_i \cdot x_{i,j}$ instead of $x_{i,j}$ for a simpler representation of the gradient descent method, as described in Section 6.1. This representation also has an advantage for computing a gradient $\Delta_w \ell(\boldsymbol{v}_i)$ over ciphertexts.

of the bootstrapping parallelization. Then we construct matrices, W_i and V_i, each of which consists of m duplicates of each of sub-vectors, $\boldsymbol{w}_i$ and $\boldsymbol{v}_i$, as follows:

$$W_i = \begin{bmatrix} w[gi], & w[gi+1], & \cdots, & w[gi+(g-1)] \\ w[gi], & w[gi+1], & \cdots, & w[gi+(g-1)] \\ & & \vdots & \\ w[gi], & w[gi+1], & \cdots, & w[gi+(g-1)] \end{bmatrix}$$

$$V_i = \begin{bmatrix} v[gi], & v[gi+1], & \cdots, & v[gi+(g-1)] \\ v[gi], & v[gi+1], & \cdots, & v[gi+(g-1)] \\ & & \vdots & \\ v[gi], & v[gi+1], & \cdots, & v[gi+(g-1)] \end{bmatrix}$$

We write $\texttt{encW}[i]$ and $\texttt{encV}[i]$ to denote encryptions of these matrices. We initialize them to be an encryption of a zero vector.

Homomorphic Evaluation of Inner Product

One of the essential operations of logistic regression is the inner product. If we have m samples over g features, then for each iteration, we have to compute m inner products on vectors of size g, where each inner product requires g^2 multiplication and $g-1$ addition operations, that is, $m \cdot (g^2 \cdot \texttt{mult} + g \cdot \texttt{add})$ operations in total. Now we will show an optimized, batch inner product method using SIMD-addition, SIMD-multiplication, and rotation operations, which requires only *two* SIMD-multiplication operations and $2\log g$ rotation-and-SIMD-addition operations to compute the m inner products, that is, $2 \cdot \texttt{SIMDmult} + 2\log g \cdot (\texttt{rot} + \texttt{SIMDadd})$ in total. This batch method is extremely efficient in the packed HEs where SIMD operations provide high throughput at no additional cost compared to non-SIMD operations.

The batch inner product method is as follows. Suppose we want to compute $Z \cdot \boldsymbol{v}$ where $Z \in \mathbb{R}^{m \times g}$ and $\boldsymbol{v} \in \mathbb{R}^{g}$. Assume that g is a power of two.[‡] First, we construct a matrix V that consists of m duplicate row-vectors of $\boldsymbol{v}$ as described in Section 6.2.2. Then, we can compute the Hadamard product, $Z \circ V$, by conducting a *single* SIMD-multiplication as follows:

$$Z \circ V = \begin{bmatrix} Z[1][1] \cdot v[1], & Z[1][2] \cdot v[2], & \cdots, & Z[1][g] \cdot v[g] \\ Z[2][1] \cdot v[1], & Z[2][2] \cdot v[2], & \cdots, & Z[2][g] \cdot v[g] \\ \vdots & \vdots & \ddots & \vdots \\ Z[m][1] \cdot v[1], & Z[m][2] \cdot v[2], & \cdots, & Z[m][g] \cdot v[g] \end{bmatrix}$$

Now, we need to compute the summation of the columns, which becomes the inner product result. We can compute the summation by repeating the rotation-and-addition operations $\log g$ times as follows. Let $\mathsf{Lrot}_i(A)$ be a matrix obtained by rotating each element of A to the left by i. Then, recursively evaluating the following recurrence relation starting from $A^{(0)} = A$ will give us $A^{(g)}$, *in* $\log g$ *steps*, whose first column is the summation of the columns of A:

[‡]Otherwise, we can pad zero columns in the end to make it a power of two.

$$A^{(2^{k+1})} = A^{(2^k)} + \mathsf{Lrot}_{2^k}(A^{(2^k)}) = \begin{bmatrix} \Sigma_{i=1}^{2^{k+1}} Z[1][i] \cdot v[i] & \cdots & - \\ \Sigma_{i=1}^{2^{k+1}} Z[2][i] \cdot v[i] & \cdots & - \\ \vdots & \ddots & \vdots \\ \Sigma_{i=1}^{2^{k+1}} Z[m][i] \cdot v[i] & \cdots & - \end{bmatrix}$$

Note that the other columns except the first are garbage, denoted by $-$, in the above. We can clean up the garbage columns by multiplying the zero vectors, and then duplicate the first column by applying the rotation-and-addition method. See Algorithm 5 for the complete details.

Algorithm 4: $\mathsf{SumRowVec}$: summation of row-vectors

```
  Data: Matrix $A$ with size $f \times g$ for a power of two $f$
  Result: Matrix $R$ with size $f \times g$
1 $R := A$;
2 for $0 \leqslant i < \log_2 f$ do
3     $R = \mathsf{Lrot}_{g \cdot 2^i}(R) + R$;
4 end
5 return $R$
```

Note that we can compute the summation of row-vectors in a similar way, as shown in Algorithm 4. Below we illustrate the results of two procedures, $\mathsf{SumRowVec}$ and $\mathsf{SumColVec}$:

$$\mathsf{SumRowVec}(A) = \begin{bmatrix} \sum_i a[i][1], & \cdots, & \sum_i a[i][g] \\ \sum_i a[i][1], & \cdots, & \sum_i a[i][g] \\ \vdots, & \ddots, & \vdots \\ \sum_i a[i][1], & \cdots, & \sum_i a[i][g] \end{bmatrix}$$

Algorithm 5: SumColVec: summation of column-vectors

```
  Data: Matrix $A$ with size $f \times g$ for a power of two $g$
  Result: Matrix $R$ with size $f \times g$
1 $R := A$ for $0 \leqslant i < \log_2 g$ do
2 |   $R = \mathsf{Lrot}_{2^i}(R) + R$
3 end
4 $D = \{D_{i,j}\}$, where $D_{i,j} = 1$ if $j = 0$ and 0 otherwise;
5 $R = R \circ D$;
6 for $0 \leqslant i < \log_2 g$ do
7 |   $R := \mathsf{Rrot}_{2^i}(R) + R$;
8 end
9 return $R$
```

$$\mathsf{SumColVec}(A) = \begin{bmatrix} \sum_j a[1][j], & \cdots, & \sum_j a[1][j] \\ \sum_j a[2][j], & \cdots, & \sum_j a[2][j] \\ \vdots, & \ddots, & \vdots \\ \sum_j a[f][j], & \cdots, & \sum_j a[f][j] \end{bmatrix}$$

for $A = \{a_{i,j}\} \in \mathbb{R}^{f \times g}$.

Vectorized Algorithm

Algorithm 6 shows the resulting vectorized body of the main iteration loop using the approaches described so far in this section. At line 6, we use the least squares fitting polynomial approximation of sigmoid, $y = 0.5 + 0.15x - 0.0015x^3$ (depicted in Figure 6.1). The bold symbols and numbers denote $m \times g$ matrices that consist of duplicates of corresponding elements. Note that the approximated sigmoid function is evaluated only once per iteration even with the partitioned weight vectors. Also, note that the two loops of iterating over the partitioned weight vectors can be run

Algorithm 6: Vectorized body of the iteration loop

```
   Data: Matrices $Z_j$, $W_j$, and $V_j$ for $0 \leqslant j < f/g$
   Result: Matrices $W_j^+$ and $V_j^+$ for $0 \leqslant j < f/g$
1  for $0 \leqslant j < f/g$ do
2      $M_j = Z_j \circ V_j$ ;
3      $M_j = \mathsf{SumColVec}(M_j)$ ;
4  end
5  $M = \sum_{j=0}^{f/g} M_j$ ;
6  $S = \mathbf{0.5} + \mathbf{0.15} \circ M - \mathbf{0.0015} \circ M^{\circ 3}$ ;
7  for $0 \leqslant j < f/g$ do
8      $S_j = S \circ Z_j$ ;
9      $\Delta_j = \mathsf{SumRowVec}(S_j)$ ;
10     $W_j^+ = V_j - \boldsymbol{\gamma} \circ \Delta_j$ ;
11     $V_j^+ = (\mathbf{1} - \boldsymbol{\eta}) \circ W_j^+ + \boldsymbol{\eta} \circ W_j$ ;
12 end
13 return $W_j^+$ and $V_j^+$ for $0 \leqslant j < f/g$
```

in parallel.

6.2.3 Further Optimization

Now we explain the further optimization made on the top of Algorithm 6 by fine-tuning the evaluation order to minimize both the depth and the noise of multiplications. Our final HE-optimized algorithm is given in Algorithm 7.[§]

Minimizing Multiplication Depth

In homomorphic evaluation, minimizing the depth of nested multiplications is critical to optimize the performance. The larger the multiplica-

[§]The definitions of encSumRowVec and encSumColVec are provided in Appendix.

tion depth, the larger the ciphertext modulus and/or the more often the bootstrapping operation needs to be executed. A large ciphertext modulus significantly increases the computation overhead, and the bootstrapping operation is very expensive. For example, when computing x^n, a naive method would require the nested multiplications of depth $n-1$, but an optimized method such as the square-and-multiply method would require only the multiplication depth of $\log n$.

We further optimize Algorithm 6 by minimizing the multiplication depth. A naive evaluation of Algorithm 6 requires the multiplication depth of 7. We reduce the depth to 5, by using the square-and-multiply method with further adjusting the evaluation order. This depth reduction allows us to reduce the size of the ciphertext modulus, improving the performance. Note that our depth minimization method will achieve a bigger depth reduction as a larger-degree polynomial is used in the sigmoid approximation (at line 6 in Algorithm 6).

$$\begin{tikzcd}
\text{Depth} & & & & & \\
0 & V_j \arrow[ddddr, dotted] \arrow[dr] \arrow[dddd, dotted] & & Z_j \arrow[dl] \arrow[d] \arrow[dr] & & W_j \arrow[dddd, dotted] \\
1 & & M_j = Z_j \circ V_j \arrow[d] & Z_j''' = (\gamma \cdot c_3) \circ Z_j \arrow[d, dotted] & Z_j' = (-\gamma \cdot c_1) \circ Z_j \arrow[ddd, dotted] & \\
2 & & M = \sum_j \mathsf{SumColVec}(M_j) \arrow[d] \arrow[dr] & Z_j''' \arrow[d] & & \\
3 & & M'' = M^{\circ 2} - \frac{c_2}{c_3} \arrow[dr] & M' = M \circ Z_j''' \arrow[d] & & \\
4 & V_j & W_j^+ = V_j + \mathsf{SumRowVec}(G_j) \arrow[l] \arrow[d, dotted] \arrow[dr] & G_j = M' \circ M'' + Z_j' \arrow[l] & Z_j' \arrow[l] & W_j \arrow[dll] \\
5 & & W_j^+ & V_j^+ = (1-\eta) \circ W_j^+ + \eta \circ W_j & &
\end{tikzcd}$$

Figure 6.2: An optimized evaluation circuit of Algorithm 6

Figure 6.2 illustrates our optimized evaluation of Algorithm 6 using the depth minimization method. The circuit is layered by the multiplication depth (in the left-hand side), where each layer consists of either normal multiplication (mult) or constant multiplication (constMult), with zero or more addition (add) operations. The solid arrow denotes the input wiring, and the dotted arrow denotes the value propagation. Since the circuit is layered by only the multiplication depth, the inputs of the addition operation are put in the same layer (e.g., as shown in the fourth layer). Algorithm 7 incorporates this optimized evaluation circuit.

For the given inputs V_j, Z_j, and W_j,¶ the first layer computes $M_j = Z_j \circ V_j$ (corresponding to the line 2 in Algorithm 6), and $Z' = (-\gamma \cdot c_1) \circ Z_j$ and $Z''' = (\gamma \cdot c_3) \circ Z_j$ (corresponding to the partial computation of the lines 6, 8, and 10). The second layer computes $M = \Sigma_j(\mathsf{SumColVec}(M_j))$ (corresponding to the lines 3 and 5). The third layer computes $M' = M \circ Z'''$ and $M'' = M^{\circ 2} - \frac{c_2}{c_3}$. The fourth layer computes $G = M' \circ M'' + Z'$ and $W_j^+ = V_j + \mathsf{SumRowVec}(G)$. The fifth layer computes $V_j^+ = (1 - \eta) \circ W_j^+ + \eta \circ W_j$. Note that $\mathsf{SumRowVec}(G)$ computed in the fourth layer effectively computes $-\gamma \circ \Delta_j$ (at line 10 in Algorithm 6).‖ Also note that the computation of $\mathsf{SumRowVec}(G)$ requires only the multiplication depth of 3, while a naive evaluation of $-\gamma \circ \Delta_j$ would require the multiplication depth of 5. In general, if we use a degree n polynomial approximation (at line 6 in Algorithm 6), our depth minimization method will reduce the multiplication depth from $O(n)$ to $O(\log n)$.

¶Indeed, the whole evaluation circuit consists of duplicates of the presented circuit for each j being arranged side-by-side, which effectively parallelizes the loops in Algorithm 6.

‖$\mathsf{SumRowVec}\ (\mathrm{G}) = \mathsf{SumRowVec}(M' \circ M'' + Z') = \mathsf{SumRowVec}((M \circ (\gamma \cdot c_3) \circ Z_j) \circ (M^{\circ 2} - \frac{c_2}{c_3}) + (-\gamma \cdot c_1) \circ Z_j) = \mathsf{SumRowVec}(-\gamma \cdot Z_j \circ (c_1 + c_2 \circ M - c_3 \circ M^{\circ 2})) = \mathsf{SumRowVec}(-\gamma \cdot Z_j \circ S) = \mathsf{SumRowVec}(-\gamma \cdot S_j) = -\gamma \circ \mathsf{SumRowVec}(S_j) = -\gamma \circ \Delta_j$

Minimizing Approximation Noise

Recall that the approximate HE used in our algorithm introduces an additional noise for each homomorphic operation. Even the homomorphic rotation and rescaling operations introduce the noise. We further optimize our algorithm to minimize the noise by reordering the evaluation order of homomorphic operations. For example, the rescaling operation has an effect of reducing the previously introduced noise. Reordering the rescaling operations, thus, can reduce the overall accumulated noise. Let us illustrate the approach. Suppose we want to multiply two ciphertexts $c_1 = \mathsf{Enc}(\boldsymbol{m_1})$ and $c_2 = \mathsf{Enc}(\boldsymbol{m_2})$, and rotate the multiplication result. Let $\boldsymbol{m}_3 = (\boldsymbol{m}_1 \circ \boldsymbol{m}_2)$. A naive way of computing that would have the following evaluation order:

$$\begin{aligned}
c_3 &= \mathsf{Mult}(\mathsf{Enc}(\boldsymbol{m}_1), \mathsf{Enc}(\boldsymbol{m}_2)) = \mathsf{Enc}(\boldsymbol{m}_3 \cdot \Delta + \epsilon_1) \\
c_4 &= \mathsf{Rescale}(c_3, \Delta) = \mathsf{Enc}(\boldsymbol{m}_3 + \epsilon_1/\Delta + \epsilon_2) \\
c_5 &= \mathsf{Rotate}(c_4, i) = \mathsf{Enc}(\mathsf{Lrot}_i(\boldsymbol{m}_3) + \epsilon_1/\Delta + \epsilon_2 + \epsilon_3)
\end{aligned}$$

where Δ is the scaling factor, and ϵ_i is the noise. However, we can reduce the final noise by adjusting the evaluation order, i.e., by swapping the rescaling operation and the rotation operation, as follows:

$$\begin{aligned}
c_3 &= \mathsf{Mult}(\mathsf{Enc}(\boldsymbol{m}_1), \mathsf{Enc}(\boldsymbol{m}_2)) = \mathsf{Enc}(\boldsymbol{m}_3 \cdot \Delta + \epsilon_1) \\
c_4' &= \mathsf{Rotate}(c_3, i) = \mathsf{Enc}(\mathsf{Lrot}_i(\boldsymbol{m}_3) \cdot \Delta + \epsilon_1 + \epsilon_2) \\
c_5' &= \mathsf{Rescale}(c_4', \Delta) = \mathsf{Enc}(\mathsf{Lrot}_i(\boldsymbol{m}_3) + (\epsilon_1 + \epsilon_2)/\Delta + \epsilon_3)
\end{aligned}$$

Note that the final noise is reduced from $\epsilon_1/\Delta + \epsilon_2 + \epsilon_3$ to $(\epsilon_1 + \epsilon_2)/\Delta + \epsilon_3$. Since $\epsilon_2 \ll \Delta$, this optimization effectively removes ϵ_2.

```
Algorithm 7: HE-optimized body of the iteration loop
  Data: Ciphertexts $\mathsf{encZ}_j$, $\mathsf{encW}_j$, and $\mathsf{encV}_j$ for $0 \leqslant j < f/g$, and
        parameters $\mathsf{wBits}$ and $\mathsf{pBits}$
  Result: Ciphertexts $\mathsf{encW}_j^+$ and $\mathsf{encV}_j^+$ for $0 \leqslant j < f/g$
1 for $0 \leqslant j < f/g$ do
2     $\mathsf{encM}_j = \mathsf{rescale}(\mathsf{mult}(\mathsf{encZ}_j, \mathsf{encV}_j), \mathsf{wBits})$;
3     $\mathsf{encM}_j = \mathsf{encSumColVec}(\mathsf{encM}_j, \mathsf{pBits})$;
4 end
5 $\mathsf{encM} = \sum_{j=0}^{f/g} \mathsf{encM}_j$;
6 $\mathsf{encM}'' = \mathsf{rescale}(\mathsf{mult}(\mathsf{encM}, \mathsf{encM}), \mathsf{wBits})$
   $\mathsf{encM}'' = \mathsf{cAdd}(\mathsf{encM}'', -\mathbf{100}, \mathsf{wBits})$ for $0 \leqslant j < f/g$ do
7     $\mathsf{encZ}' = \mathsf{constMult}(\mathsf{encZ}_j, -\boldsymbol{\gamma} \circ \mathbf{0.5}, \mathsf{wBits})$
       $\mathsf{encZ}''' = \mathsf{constMult}(\mathsf{encZ}_j, \boldsymbol{\gamma} \circ \mathbf{0.0015}, \mathsf{wBits})$
       $\mathsf{encZ}''' = \mathsf{modDownTo}(\mathsf{encZ}''', \mathsf{encM})$
       $\mathsf{encM}' = \mathsf{rescale}(\mathsf{mult}(\mathsf{encM}, \mathsf{encZ}'''), \mathsf{wBits})$
       $\mathsf{encG} = \mathsf{rescale}(\mathsf{mult}(\mathsf{encM}', \mathsf{encM}''), \mathsf{wBits})$
       $\mathsf{encG} = \mathsf{add}(\mathsf{encG}, \mathsf{modDownTo}(\mathsf{encZ}', \mathsf{encG}))$
       $\mathsf{encG} = \mathsf{encSumRowVec}(\mathsf{encG})$
       $\mathsf{encW}_j^+ = \mathsf{add}(\mathsf{encG}, \mathsf{modDownTo}(\mathsf{encV}_j, \mathsf{encG}))$
       $\mathsf{encW}_j = \mathsf{modDownTo}(\mathsf{encW}_j, \mathsf{encW}_j^+)$
       $\mathsf{encW}_{j,1}^+ = \mathsf{constMult}(\mathsf{encW}_j^+, \mathbf{1} - \boldsymbol{\eta}, \mathsf{pBits})$
       $\mathsf{encW}_{j,2}^+ = \mathsf{constMult}(\mathsf{encW}_j, \boldsymbol{\eta}, \mathsf{pBits})$
       $\mathsf{encV}_j^+ = \mathsf{add}(\mathsf{encW}_{j,1}^+, \mathsf{encW}_{j,2}^+)$ $\mathsf{encV}_j^+ = \mathsf{rescale}(\mathsf{encV}_j^+, \mathsf{pBits})$
       $\mathsf{encW}_j^+ = \mathsf{modDownTo}(\mathsf{encW}_j^+, \mathsf{encV}_j^+)$
8 end
9 return $\mathsf{encV}_j^+$ and $\mathsf{encW}_j^+$ for $0 \leqslant j < f/g$
```

6.3 Evaluation

We evaluate our algorithm of logistic regression on encrypted data against both a real financial training dataset and the MNIST dataset. Our artifact is publicly available at [Han18].

6.3.1 Logistic Regression on Encrypted Financial Dataset

We executed our algorithm on a private, real financial dataset to evaluate the efficiency and the scalability of our algorithm on a large dataset.

Training Dataset

The *encrypted* dataset we consider to evaluate our logistic regression algorithm is the real consumer credit information maintained by a credit reporting agency. The dataset (for both training and validation), randomly sampled by the agency, consists of 1,266,325 individuals' credit information over 200 features that are used for credit rating. Examples of the features are the loan information (such as the number of credit loans and personal mortgages), the credit card information (such as the average amount of credit card purchases and cash advances in the last three months), and the delinquency information (such as the days of credit card delinquency). The samples are labeled with a binary classification that refers to whether each individual's credit rating is below the threshold.

HE Scheme Parameters

We use two scaling factors $\Delta = 2^{30}$ and $\Delta_c = 2^{15}$, where Δ is the regular scaling factor (for mult) and Δ_c is the constant scaling factor (for $\mathsf{constMult}$) that is used for multiplying constant matrices and scalars such as $\boldsymbol{\eta}$ and $\boldsymbol{\gamma}$. We have the number of ciphertext slots $N/2 = 2^{15}$.

We set the initial ciphertext modulus Q for the weight vectors W and V as follows:

$$\log_2 Q = 5 + \texttt{wBits} + \texttt{I} \cdot (3 \cdot \texttt{wBits} + 2 \cdot \texttt{pBits})$$

where `wBits` $= \log_2 \Delta$ and `pBits` $= \log_2 \Delta_c$. The above formula is derived from the fact that each iteration reduces $(3 \cdot \texttt{wBits} + 2 \cdot \texttt{pBits})$-bits of the ciphertext modulus (see Section 6.2.3 and Figure 6.2 for more details). Here, `I` is the number of iterations per bootstrapping operation; that is, the bootstrapping operation is executed every `I` iterations. We have `I` $=$ 5. Also, we set the largest ciphertext modulus Q' for the bootstrapping, according to the HeaAn scheme [CHK+18, snu18], as follows: $\log_2 Q' = \log_2 Q + 24 + 14 \cdot (9 + \texttt{wBits})$.

Experimental Results

Data			Performances	
Financial	No. Samples (training)	422,108	Accuracy	80%
	No. Samples (validation)	844,217	AUROC	0.8
	No. Features	200	K-S value	50.84
	No. Iterations	200	Public Key Size	$\approx$ 2 GB
	Learning Rate	0.01	Encrypted Block Size	4.87 MB
	Block Size (mini-batch)	512	Running Time	1060 min

Table 6.1: Result of machine learning on encrypted data

We executed our logistic regression algorithm on the encrypted training set of 422,108 samples over 200 features. Having 200 iterations, it took 1,060 minutes to learn an *encrypted* model, i.e., ~5 minutes per iteration on average, in a machine with IBM POWER8 (8 cores, 4.0GHz) and 256GB RAM. We sent the learned model to the data owner, and they decrypted and evaluated it on the validation set of 844,217 samples, having 80% accuracy and the KS value of 50.84. They confirmed that it provides a sufficient accuracy compared to their internal model learned using the plaintext dataset[**] and also our learned model gives appropriate weights

[**]According to their report, it took several minutes to learn a model on the plaintext

on the important features (e.g., delinquency, loan, and credit card information) as expected.

Tabel 6.1 shows the detailed result of our experiment. We set the learning rate to be 0.01, and the mini-batch size to be 512. The ciphertext size of each mini-batch block is 4.87 MB, and thus the total size of the encrypted dataset is $\sim$4 GB = 4.87 MB $\times$ (422,108 / 512). The public key size is $\sim$2 GB.

6.3.2 Logistic Regression on Encrypted MNIST Dataset

We executed our logistic regression algorithm on the public MNIST dataset to provide a more detailed evaluation.

Training Dataset and Parameters

We took the MNIST dataset [LCB99], and restructured it for the binary classification problem between 3 and 8. We compressed the original images of 28×28 pixels into 14×14 pixels, by compressing 2×2 pixels to their arithmetic mean. The restructured dataset consists of 11,982 samples of the training dataset and 1,984 samples of the validation dataset.

We use the same principle for setting the HE scheme parameters as shown in Section 6.3.1. We set $\Delta = 2^{40}$, $\Delta_c = 2^{15}$, and $\mathtt{I} = 3$. Also, we approximate the sigmoid function with the interval $[-16, 16]$ by the least squares fitting polynomial of degree 3, $y = 0.5 - 0.0843x + 0.0002x^3$.

Data			Performances	
MNIST	No. Samples (training)	11,982	Accuracy	96.4%
	No. Samples (validation)	1,984	AUROC	0.99
	No. Features	196	K-S value	N/A
	No. Iterations	32	Public Key Size	$\approx$ 1.5 GB
	Learning Rate	1.0	Encrypted Block Size	3.96 MB
	Block Size (mini-batch)	1024	Running Time	132 min

Table 6.2: Result of machine learning on encrypted data

Experimental Results

We encrypted the MNIST dataset and executed our logistic regression algorithm. Table 6.2 shows the result. With 32 iterations, our logistic algorithm took 132 minutes to learn an encrypted model. The average time for each iteration is $\sim$4 minutes, which is similar to that of the financial dataset, as expected. We decrypted the learned model and evaluated it on the validation dataset, obtaining 96.4% accuracy.[††]

Microbenchmarks

We also executed our logistic regression algorithm on the plaintext dataset, and compared the result to that of the ciphertext learning. Recall that the approximate HE used in our algorithm introduces the approximation noise for each computation step, but it had not been clear how much the noise affects the overall training process. To evaluate the impact of the approximation noise on the overall learning performance (e.g., the convergence rate and accuracy), we measured the accuracy for each iteration for both plaintext and ciphertext training, and compared those results. Figure 6.3

using the same algorithm, and the model provides the KS value of 51.99.

[††]The accuracy seems to be lower than the usual, but the difference is mainly due to the image compression, not because of the approximation noise. See Section 6.3.2 and Figure 6.3 for more details.

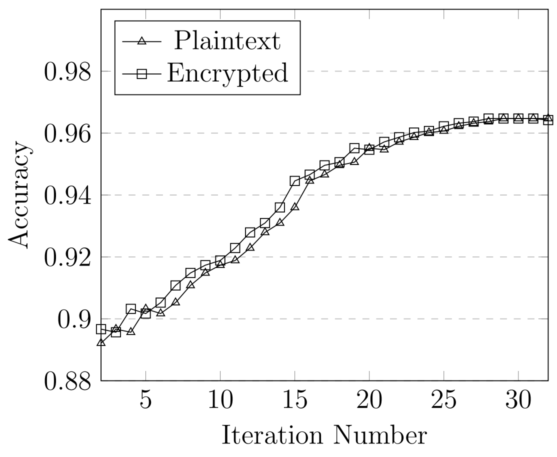

Figure 6.3: Comparison between encrypted and plaintext training

shows the comparison result. It shows that the accuracy for each iteration in the ciphertext training is marginally different from that of the plaintext, especially in the early stage of the training process, but they eventually converged at the final step. This result implies that the additional noise introduced by the approximate HE evaluation is not significant to deteriorate the accuracy of a learned model and the training performance.

We also evaluate the effect of the precision of the polynomial approximation of sigmoid. We executed the same algorithm (on the plaintext) with three different sigmoid approximations: the original sigmoid (i.e., no approximation), the least squares fitting polynomial, and the Taylor expansion polynomial (depicted in Figure 6.1). Figure 6.4 and 6.5 show the comparison of accuracy between them. It shows that the approximation error of the least fitting polynomial is not significant, resulting in only the marginal difference of accuracy. However, the approximation error of the Taylor expansion polynomial is so large that it fails to learn a model; that is, the accuracy decreases as the number of iteration increases, and eventually it becomes 0 (i.e., an invalid model).

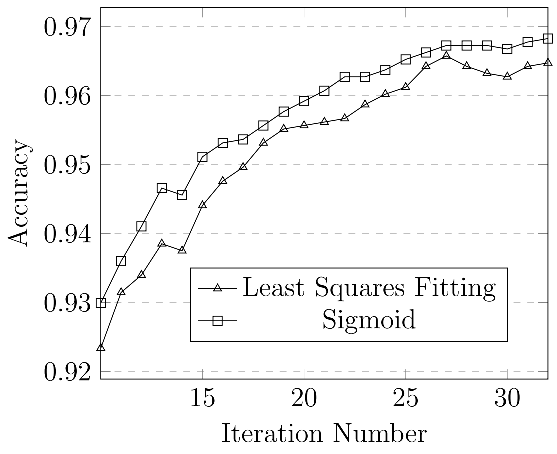

Figure 6.4: Comparison between sigmoid and least squares fitting (of degree 3)

6.3.3 Discussion

It is not straightforward to provide the fair comparison of our performance with those of the related works, since the previous HE-based approaches are *not* capable of admitting such realistic size training datasets considered in this paper, and the MPC-based approaches do not support the same flexibility in the usage scenarios as ours. As a rough comparison, however, the recent MPC-based approach [MZ17] will take minutes[‡‡] to learn a model on the MNIST dataset used in this paper, which is one or two orders of magnitude faster than ours. We note that, however, the MPC-based approach requires the additional assumption in the usage scenarios

[‡‡]The time is obtained by extrapolating their experimental result on the MNIST dataset.

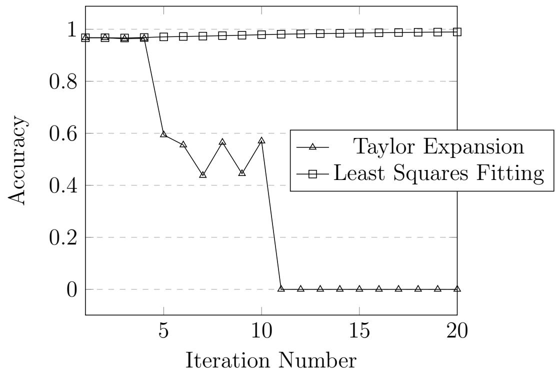

Figure 6.5: Comparison between Taylor expansion between least squares fitting (both of degree 3)

that either the number of participants is small, or the two servers do not collude.

Our algorithm requires the number of iterations to be provided in advance, which is inevitable due to the security of the underlying HE schemes. In our experiment on the financial data, the number was obtained by asking the data owner to provide a rough bound. We note that, however, one can use our algorithm in an interactive way that the data owners decrypt the learned model periodically (e.g., every 100 iterations), and decide whether to proceed further or not, depending on the quality of the model at the moment.

Chapter 7

Conclusions

In this paper, we propose various bootstrapping methods for fully homomorphic encryption. In case of Ring-based FHE such as BGV and FV, we propose new homomorphic lower digit removal algorithm. And, this algorithm is used as one of the key part in bootstrapping. If we compare with previous one, the complexity reduce from $O(\log^2 \lambda)$ to $O(\log^{1.5} \lambda)$. We also improve linear transformation part in bootstrapping. we decompose the given linear transformation by $\log n$ number of sparse matrices for number of slots n. Multiplying this sparse matrix to encrypted vector only needs $O(1)$ number of homomorphic operations, and this leads the complexity of the linear transformation to $O(\log n)$ which was $O(n)$ in the previous method. In case of FHE over the integers, we represent the decryption function using digit extraction algorithm which can be homomorphically evaluated only with $O(\log^2 \lambda)$ operations. With those efficient bootstrapping processes, we made a privacy preserving logistic regression algorithm for large scaled data. Training for $400,000 \times 200$ financial data which is encrypted takes 16 hours with 80% accuracy.

Bibliography

[AMBG$^+$16] 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.

[APS15] 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.

[BGV12] 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.

[BIWX11] Arnab Bhattacharyya, Piotr Indyk, David P Woodruff, and Ning Xie. The complexity of linear dependence problems in vector spaces. In *ICS*, pages 496–508, 2011.

[BLLN13] 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.

[BLN14] Joppe W Bos, Kristin Lauter, and Michael Naehrig. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics*, 50:234–243, 2014.

[BMMP17] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. Cryptology ePrint Archive, Report 2017/1114, 2017.

[BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, pages 868–886, 2012.

[BV11] 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.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[CBD14] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[CCK$^+$13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2013*, pages 315–335. Springer, 2013.

[CCS18] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. *IACR Cryptology ePrint Archive*, 2018:1043, 2018.

[CdWM$^+$17] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive*, 2017:35, 2017.

[CH18] Hao Chen and Kyoohyung 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.

[CHH18] Jung Hee Cheon, Kyoohyung Han, and Minki Hhan. Faster Homomorphic DFT and Improved Bootstrapping for FHE. *IACR Cryptology ePrint Archive*, 2018:1073, 2018.

[CHK17] Jung Hee Cheon, Kyoohyung Han, and Duhyeong Kim. Faster bootstrapping of FHE over the integers. *IACR Cryptology ePrint Archive*, 2017:79, 2017.

[CHK$^+$18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on* *the Theory and Applications of Cryptographic Techniques*, pages 360–384. Springer, 2018.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[CLT14] 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.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Advances in Cryptology–CRYPTO 2011*, pages 487–504. Springer, 2011.

[CN12] Yuanmi Chen and Phong Q Nguyen. Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers. In *Advances in Cryptology–EUROCRYPT 2012*, pages 502–519. Springer, 2012.

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2012*, pages 446–464. Springer, 2012.

[CS15] Jung Hee Cheon and Damien Stehlé. Fully homomophic encryption over the integers revisited. In *Advances in Cryptology–EUROCRYPT 2015*, pages 513–536. Springer, 2015.

[CSV17] Anamaria Costache, Nigel P Smart, and Srinivas Vivek. Faster homomorphic evaluation of discrete fourier transforms. In *International Conference on Financial Cryptography and Data Security*, pages 517–529. Springer, 2017.

[CSVW16] Anamaria Costache, Nigel P Smart, Srinivas Vivek, and Adrian Waller. Fixed-point arithmetic in she schemes. In *International Conference on Selected Areas in Cryptography*, pages 401–422. Springer, 2016.

[CT65] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[DM15] 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.

[DSFRO17] Christopher De Sa, Matthew Feldman, Christopher Ré, and Kunle Olukotun. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 561–574. ACM, 2017.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[G$^+$96] Torbjörn Granlund et al. The gnu multiple precision arithmetic library. *TMG Datakonsult, Boston, MA, USA*, 2(2), 1996.

[GAGN15] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[GBDL$^+$16] 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.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[GHS12a] 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.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO 2012*, pages 850–867. Springer, 2012.

[GLN12] 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.

[Gri17] Michael Griffin. Lowest degree of polynomial that removes the first digit of an integer in base p. `https://mathoverflow.net/q/269282` (version: 2017-05-08), 2017.

[Han18] Kyoohyung Han. Homomorphic Logistic Regression on Encrypted Data. `https://github.com/HanKyoohyung/HELR`, 2018.

[HHCP18] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Efficient Logistic Regression on Large Encrypted Data. *IACR Cryptology ePrint Archive*, 2018:662, 2018.

[HS15] Shai Halevi and Victor Shoup. Bootstrapping for HElib. In *Advances in Cryptology–EUROCRYPT 2015*, pages 641–670. Springer, 2015.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, 2018. USENIX Association.

[KSK$^+$18] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. 2018.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM, 2012.

[LCB99] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST Database of Handwritten Digits, 1999.

[LLH$^+$17] Ping Li, Jin Li, Zhengan Huang, Chong-Zhi Gao, Wen-Bin Chen, and Kai Chen. Privacy-preserving outsourced classification in cloud computing. *Cluster Computing*, pages 1–10, 2017.

[LP16] Kim Laine and Rachel Player. Simple encrypted arithmetic library-SEAL (v2. 0). Technical report, Technical report, September, 2016.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[MZ17] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 19–38. IEEE, 2017.

[NK15] 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.

[PS73] 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.

[RAD78] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. 1978.

[snu18] snucrypto. HEAAN. `https://github.com/snucrypto/HEAAN`, 2018.

[SV14] Nigel P Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Designs, codes and cryptography*, pages 1–25, 2014.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in cryptology–EUROCRYPT 2010*, pages 24–43. Springer, 2010.

[ZLK$^+$16] Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. The zipml framework for training models with end-to-end low precision: The cans, the cannots, and a little bit of deep learning. *arXiv preprint arXiv:1611.05402*, 2016.

국문초록

2009년 Gentry에 의해서 완전동형암호가 처음 설계된 이후로 최적화와 고속화를 위해서 다양한 기법들과 스킴들이 설계되어 왔다. 하지만 동형암호의 연산횟수를 무제한으로 늘리기 위해서 필수적인 재부팅 기법의 효율성 문제로 실제 응용에 적용하기에는 부적합하다는 평가를 많이 받아왔다. 본 논문에서는 재부팅 기법의 고속화를 위한 다양한 기법을 제시하고 이를 실제로 응용분야에 적용하였다.

본 논문에서는 대표적인 동형암호 스킴들에 대한 재부팅 기법에 대한 연구를 수행하였는데, 첫 번째로는 Microsoft Research와 IMB에서 만든 동형암호 라이브러리인 SEAL과 HElib에 적용가능한 재부팅 기법에 대한 연구를 수행하였다. 해당 재부팅 기법에서 핵심적이 과정은 암호화된 상태에서 복호화 함수를 계산하는 부분이다. 암호된 상태에서 최하위 비트를 추출하는 새로운 방법을 제시하여 재부팅 과정에서 소모되는 계산량과 표현되는 다항식의 차수를 줄이는데에 성공하였다. 두 번째로는, 비교적 최근에 개발된 근사계산 동형암호인 HEAAN 스킴의 재부팅 기법을 개선하는 연구를 수행하였다. 2018년에 삼각함수를 이용한 근사법을 통해서 처음 해당 스킴에 대한 재부팅 기법이 제시되었는데, 많은 데이터를 담고있는 암호문에 대해서는 전처리, 후처리 과정이 계산량의 대부분을 차지하는 문제가 있었다. 해당 과정들을 여러 단계로 재귀적인 함수들로 표현하여 계산량이 데이터 사이즈에 대해서 로그적으로 줄이는 것에 성공하였다. 추가로, 다른 스킴들에 비해서 많이 사용되지는 않지만, 정수기반 동형암호들에 대해서도 재부팅 기법을 개선하는 연구를 수행하였고 그 결과 계산량을 로그적으로 줄이는 것에 성공하였다.

마지막으로, 재부팅 기법의 활용성과 사용 가능성을 보이기 위해 실제 데이터 보안을 필요로 하는 기계학습 분야에 적용해보았다. 실제로 400,000건의 금융 데이터를 이용한 회귀분석을 암호화된 데이터를 이용해서 수행하였다. 그 결과 약 16시간 안에 80% 이상의 정확도와 0.8 정도의 AUROC 값을 가지는 유의미한 분석 모델을 얻을 수 있었다.

주요어휘: 동형암호, 정보보호, 재부팅

학번: 2013-20250

감사의 글

2009년 3월 학부에 입학한 이후로 10년 동안 학부와 대학원 생활에 도움을 준 모든 분들께 감사를 표합니다. 대학원 입학 전에는 생활지도 교수님으로써 많은 조언을 해주시고 대학원에서는 지도 교수님으로써 부족한 저를 학문적으로 그리고 생활적으로 지도해주신 천정희 교수님께 진심으로 감사드립니다. 바쁘신 와중에도 논문 심사를 해주신 김명환 교수님께 감사드리며, 심형보 교수님, 현동훈 교수님, 김명선 교수님께 감사드립니다.

기나긴 10년이라는 시간동안 묵묵히 지켜봐주신 아버지 어머님께 감사드립니다. 저에게 너가 하고싶은 일을 하라며 응원해주시고 논문 게제와 같은 좋은 소식이 있을 때마다 자신의 일처럼 기뻐해주셔서 감사합니다. 앞으로 기대를 저버리지 않도록 부족하겠지만 보답하는 아들이 되도록 하겠습니다. 그리고 제가 뜻하는 바를 달성하기 위해서 졸업 이후에도 더 열심히 연구와 공부에 정진하여 도록 하겠습니다. 그리고 학부 1학년부터 함께 대학원 입학준비, 장학금 준비, 그리고 논문 자격 시험 준비를 함께하면서 생활적으로 학문적으로 큰 도움이 되어준 석범, 대준에게도 감사를 표합니다.

연구실 생활을 하면서 처음 함께 동형암호 논문을 읽었던 삼성에 계신 진수 형부터 지금 연구실 막내 기우까지 다들 많은 도움 감사합니다. 부족했던 저와 첫 논문을 함께 했던 한솔 누나와 창민이, 연구실의 큰 형님으로 힘들어도 미소를 잃지 않았던 김진수 소령님, 항상 웃으면서 많은 조언을 해주시는 미란 누나, 가장 가까운 선배이면서 같은 주제를 연구해서 가장 많은 이야기를 나눈 용수 형에게 감사의 말을 전합니다. 디스크로 고생하면서 연구하고 있는 진혁, 최근에 결혼해서 신혼여행에 있는 랩장님 주희, 작년에 랩장을 하면서 많은 도움을 받은 분석팀의 리더 지승, 동형암호 연구의 새로운 리더로 기대되는 용하, 건강이 나아지기를 바라는 재윤, 많은 술을 함께 기울였던 두형, 앞으로 기대되는 민기, 의도치 않게 많은 일들을 시켰던 승완, 원희, 잘생긴 막내 기우까지 모두에게 감사의 말을 전합니다.