이학 박사 학위논문

Multivariate Homomorphic Encryption for Approximate Matrix Arithmetics

(근사 행렬연산을 위한 다변수 동형암호)

2019년 8월

서울대학교 대학원

수리과학부

김안드레이

Multivariate Homomorphic Encryption for Approximate Matrix Arithmetics

(근사 행렬연산을 위한 다변수 동형암호)

지도교수 천정희

이 논문을 이학 박사 학위논문으로 제출함

2019년 4월

서울대학교 대학원

수리과학부

김안드레이

김안드레이 의 이학 박사 학위논문을 인준함

2019년 6월

Multivariate Homomorphic Encryption for Approximate Matrix Arithmetics

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

Andrey Kim

Dissertation Director : Professor Jung Hee Cheon

Department of Mathematical Sciences
Seoul National University

August 2019

Abstract

Multivariate Homomorphic Encryption for Approximate Matrix Arithmetics

Andrey Kim
Department of Mathematical Sciences
The Graduate School
Seoul National University

Homomorphic Encryption for Arithmetics of Approximate Numbers (HEAAN) is a homomorphic encryption (HE) scheme for approximate arithmetics intoroduced by Cheon et.al. [CKKS17]. Its vector packing technique proved its potential in cryptographic applications requiring approximate computations, including data analysis and machine learning.

Multivariate Homomorphic Encryption for Approximate Matrix Arithmetics (MHEAAN) is a generalization of HEAAN to the case of a tensor structure of plaintext slots. Our design takes advantage of the HEAAN scheme, that the precision losses during the evaluation are limited by the depth of the circuit, and it exceeds no more than one bit compared to unencrypted approximate arithmetics, such as floating point operations. Due to the multi-dimensional structure of plaintext slots along with rotations in various dimensions, MHEAAN is a more natural choice for applications involving matrices and tensors.

The concrete two-dimensional construction shows the efficiency of the MHEAAN scheme on matrix operations, and was applied to several Machine Learning algorithms on encrypted data and encrypted model such as Logistic Regression (LR) training algorithm, Deep Neural Network (DNN) and Recurrent Neural Network (RNN) classification algorithms. With the efficient bootstrapping, the implementation can be easily be scaled to the case of arbitrary LR, DNN or RNN structures.

Key words: homomorphic encryption, privacy protection, machine learning
Student Number: 2014-31408

Contents

Chapter 1

Introduction

Homomorphic Encryption (HE) [RAD78] allows to perform certain arithmetics operations in encrypted state. Following Gentry's blueprint [Gen09], a numerous HE schemes have been proposed [DGHV10, BV11a, BV11b, Bra12, BGV12, GHS12, LATV12, BLLN13, GSW13, CLT14, CS15, DM15, DHS16, CKKS17, CGGI18]. The most asymptotically efficient HE schemes are based on the hardness of RLWE, and they normally have a common structure of ciphertexts with noised encryption for security.

In calculations, floating-point arithmetic (FP) is arithmetic using the formal representation of real numbers as an approximation to maintain a compromise between range and accuracy. For this reason, floating point calculations are often found in systems that include very small and very large real numbers (e.g. floating point numbers) that require fast processing time. The number, as a rule, is presented approximately to a fixed number of significant digits (values) and is scaled using the exponent in some fixed base. Over the years, a variety of floating-point representations have been used in computers systems.

Recently Cheon et. al. [CKKS17] presented a method of constructing an HE scheme for arithmetics of approximate numbers (called `HEAAN`). The idea of the construction is to treat encryption noise as a part of error occurring during approximate computations. In other words, a ciphertext ct of a plaintext $m \in \mathcal{R}$ encrypted by a secret key sk for an underlying ciphertext modulus q will have a decryption structure of the form $\langle \mathsf{ct}, \mathsf{sk} \rangle = m + e \pmod{\mathcal{R}/q\mathcal{R}}$ for some small error e. `HEAAN` is based on an RLWE structure over a power-of-two $M = 2 \cdot N$ cyclotomic ring modulo q, $\mathcal{R}/q\mathcal{R} = \mathbb{Z}_q[X]/(X^N + 1)$. A vector of complex values of size up to $N/2$ can be encoded using a variant of canonical embedding map.

`HEAAN` showed its potential by providing the winning solution of Track 3 (Homomorphic Encryption Based Logistic Regression Model Learning) at the iDASH privacy and security competition in 2017 [KSK+]. In the iDASH 2018, all the participants used `HEAAN` scheme as an underlying scheme for the Secure Parallel Genome Wide Association Studies using Homomorphic Encryption (Track 2) [CKK+17].

In both years in their solutions authors packed a matrix of inputs in a vector. Even though the authors could provide all computations using matrix to vector packing in that particular task, due to absence of row-wise matrix rotation functionality they had to circumvent and consume an additional level during the computations. With the growth of more complex algorithms, such as deep learning and recommendation systems which require lots of matrix operations, the possibility of performing matrix operations is becoming crucial for homomorphic encryptions. Despite the diversity of HE schemes that achieve a variety of circuit evaluations, practical matrix operations such as matrix multiplications is still a problem in HE.

1.1 Multidimensional Variant of HEAAN

We present generalization of HEAAN with a tensor packing method, along with natural rotations in various dimensions, which is, called the hypercube structure, also applied in HElib [HS14, HS15, HS18]. The straightforward attempt could be based on the Multivariate RLWE (m-RLWE) problem as an underlying hardness problem, introduced by Pedrouzo-Ulloa et al. [PUTPPG15, PUTPPG16] as a multivariate variant of RLWE problem with an underlying ring $\mathbb{Z}[x_0, x_1]/(x_0^{N_0} + 1, x_1^{N_1} + 1)$ where both N_0 and N_1 are powers-of-two. However this problem succumbs to the following evaluation attack: without loss of generality assume $N_0 \geqslant N_1$, and substitute $x_1 = x_0^{N_0/N_1}$, then the RLWE problem over $\mathbb{Z}[x_0, x_1]/(x_0^{N_0} + 1, x_1^{N_1} + 1)$ reduces to a problem over $\mathbb{Z}[x_0]/(x_0^{N_0} + 1)$.

So instead, we provide a scheme MHEAAN based on the m-RLWE problem with indeterminates x_0 and x_1 (or in general case $x_0, \ldots, x_s$) satisfying relations given by cyclotomic polynomials corresponding to relatively prime orders. The hardness of the m-RLWE problem over this ring is shown to have reduction from the origina RLWE problem. MHEAAN enjoys all the benefits of HEAAN such as a rescaling procedure, which enables us to preserve the precision of the message after approximate computations and to reduce the size of ciphertexts significantly. Thus, the scheme can be a reasonable solution for approximate computation over the complex values. Moreover, with a multivariable structure of m-RLWE, we provide a general technique for tensor plaintext slots packing in a single ciphertext. We provide a concrete two-dimensional construction which supports matrix operations as well as standard HE operations.

For two-dimensional case corresponding to natural matrix structure of plaintext slots, matrix multiplication in MHEAAN is achieved in very simple way using Fox matrix multiplication algorithm [FO87]. In contrast to the method of Mishra et al. [MRDY] our method does not require exponentially large degree of the base ring and we can use matrix multiplication as a part of more complex algorithms. The matrix size is also not a problem, as our method preserves matrix structure, and can combined with divide-and-conquer algorithm. Moreover `MHEAAN` enjoys other matrix related operations, like matrix transposition.

`MHEAAN` supports faster bootstrapping procedure than that of `HEAAN` when number of slots is sufficiently large. For base ring degree N, the bootstrapping procedure for large number of slots in `MHEAAN` is approximately requires $O(N^{\frac{1}{2(s+1)}})$ of ciphertext rotations and $O(N^{\frac{1}{s+1}})$ of constant multiplications where $s+1$ is the number of factors of base ring. The original `HEAAN` requires about $O(\sqrt{N})$ of ciphertext rotations and $O(N)$ of constant multiplications. In our implementation s is equal to 1 and the degree of ring is factored into values close to $\sqrt{N}$, so the bootstrapping complexity is reduced from $O(\sqrt{N})$ to $O(\sqrt[4]{N})$ rotations and from $O(N)$ to $O(\sqrt{N})$ constant multiplications.

1.2 Applications To Machine Learning

Machine Learning is a class of artificial intelligence methods whose characteristic feature is not a direct solution of a problem, but learning in the process of applying solutions to a multitude of similar tasks. To build such methods, the tools of mathematical statistics, numerical methods, optimization methods, probability theory, graph theory, and various techniques of working with data in digital form are used.

The scope of ML applications is constantly expanding, however, with the rise of ML, the security problem has become an important issue. For example, many medical decisions rely on logistic regression model, and biomedical data usually contain confidential information about individuals which should be treated carefully. Therefore, privacy and security of data are the major concerns, especially when deploying the outsource analysis tools. In most of the ML based online services, model-service providers have a common strategy that a trained model resides on a server and returns computed values of data uploaded by the user instead of releasing the trained model in public. This is because not only the trained models with the massive amount of data have high economic values, but also publicly available models are vulnerable to adversarial attacks. On the other hand, in perspectives of such service users, one of the major concerns is about privacy of their data. Users lose control over the data after uploading it to the online services. In other words, it is impossible for users to know who will access their data and how the data will be used. And Even if model-service providers are honest, there is always a risk of information leakage due to external adversaries. For this reason, users become reluctant to use such services, despite how helpful those services are. Therefore it is essential to execute inferences of the trained ML models while preserving data privacy.

Homomorphic encryption can be a solution to this problem, which is an encryption scheme that allows calculations on several operations on encrypted data without decryption. We show several applications of `MHEAAN` to different machine learning algorithms.

Logistic Regression Training Phase.

Before iDASH 2017 several papers already discussed ML with HE techniques. Wu et al. [WS13] used Paillier cryptosystem [LP13] and approximated the logistic function using polynomials, but it required an exponentially growing computational cost in the degree of the approximation polynomial. Aono et al. [AHTPW16] and Xie et al. [XWBB16] used an additive HE scheme to aggregate some intermediate statistics. However, the scenario of Aono et al. relies on the client to decrypt these intermediary statistics and the method of Xie et al. requires expensive computational cost to calculate the intermediate information. The most related research of our approach is the work of Kim et al. [KSW$^+$] which also used HE based ML. However, the size of encrypted data and learning time were highly dependent on the number of features, so the performance for a large dataset was not practical in terms of storage and computational cost.

We propose a general practical solution for `MHEAAN` based logistic regression learning algorithm over encrypted data. Our approach demonstrates good performance and low storage costs. In practice, our output quality is comparable to the one of an unencrypted learning case. To improve the performance, we apply several additional techniques including a matrix packing method, which reduce the required storage space and optimize the computational time. We also adapt Nesterov's accelerated gradient [Nes83] to improve the convergence rate. As a result, we used less number of iterations than the other solutions, resulting in a much faster time to learn a model.

Deep Neural Network Classification.

A deep neural network is an artificial neural network with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship.

Previous implementations of encrypted prediction [BMMP17, HTG17] are done over the plain models, and limited number of hidden layers. The result of [BMMP17] has an impressive performance, but it is restricted only for a binary model, and is expected to have huge drowning in the efficiency when expanding to a non-binary model.

We constructed `MHEAAN` based Deep Neural Network classification algorithm with 2 and 6 number of layers. With matrix packing we with the rotation technique we optimized the storage space and computational time. The encrypted predictions achieve the accuracy similar to the accuracy of the predictions on the plain data. With our practical bootstrapping method our approach is flexible and can be generalized to the DNN architecture with large number of hidden layers.

Recurrent Neural Network Classification.

Recurrent Neural Networks (RNNs) are popular models that have shown great promise in many sequential data and among others used by Apples Siri and Googles Voice Search. Their great advantage is that the algorithm remembers its input, due to an internal memory. RNN model has much more complex structure than standard DNN model, thus it is much harded to adapt it with HE.

We chose as an application a model designed in deepTarget (Lee et al.) [LBPY16] as a validation of `MHEAAN` scheme. We evaluate the scalability of `MHEAAN` on a sequential model with RNA sequences, where privacy is critical. As far as our knowledge, this is the first attempt to implement RNN using FHE.

1.3 List Of Papers

Andrey Kim was a co-author for original `HEAAN` papers. The contribution of Andrey Kim was researching and drafting the source code. The original papers for `HEAAN` scheme are:

- [CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, Yongsoo Song, *Homomorphic Encryption for Arithmetic of Approximate Numbers*, 978-3-319-70693-1, ASIACRYPT 2017, Part 1, LNCS 10624.
- [CHK$^+$18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, Yongsoo Song, *Bootstrapping for Approximate Homomorphic Encryption*, EUROCRYPT 2018.

This thesis is based on the following papers:

- [KSK$^+$] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, Jung Hee Cheon, Logistic Regression Model Training based on the Approximate Homomorphic Encryption, BMC Medical Genomics 2018, vol. 11 (suppl. 4) :83, (SCI) doi: 10.1186/s12920-018-0401-7
- [CKY18] Jung Hee Cheon, Andrey Kim, Donggeon Yhee Multi-dimensional Packing for HEAAN for Approximate Matrix Arithmetics
- [JKN$^+$19] Jaehee Jang, Andrey Kim, Byunggook Na, Byunghan Lee, Sungroh Yoon and Junghee Cheon. Privacy-Preserving Inference for Gated RNNs with Matrix Homomorphic Encryptions

  In [KSK$^+$] and [CKY18] Andrey Kim was the main author and contributor. In [JKN$^+$19] Andrey Kim designed the source code.

Chapter 2

Background Theory

To avoid an ambiguity, we define tensors following linear algebras :

Definition 2.0.1. *A* tensor *is an assign from a multi-indices set to values. A tensor is of rank k if the multi-indices set consists of k-tuple of indices. A vector is a rank 1 tensor and a matrix is a rank 2 tensor.*

2.1 Basic Notations

All logarithms are base 2 unless otherwise indicated. We denote vectors in bold, e.g. $\mathbf{a}$, and every vector in this paper will be a column vector. For vectors $\mathbf{a}$ and $\mathbf{b}$ we denote by $\langle \mathbf{a}, \mathbf{b} \rangle$ the usual dot product. We denote matrices by bold capital letters, e.g. $\mathbf{A}$, and general tensors by $\hat{\mathbf{a}}$. For a real number r, $\lfloor r \rceil$ is the nearest integer to r, rounding upwards in case of a tie. For an integer q, we identify the ring $\mathbb{Z}_q$ with $(-q/2, q/2]$ as a representative interval and for integer r we denote by $\lfloor r \rceil_q$ the reduction of r modulo q into that interval. We use $a \leftarrow \chi$ to denote the sampling a according to a distribution χ. If χ is a uniform distribution on a set $\mathcal{D}$, we use $a \leftarrow \mathcal{D}$ rather than $a \leftarrow \chi$. For rank k tensors $\hat{\mathbf{a}}, \hat{\mathbf{b}} \in \mathbb{C}^{n_1 \times \cdots \times n_k}$ we denote a component-wise product by $\hat{\mathbf{a}} \odot \hat{\mathbf{b}}$. For vectors $\mathbf{r} = (r_1, \ldots, r_k)$ and $\mathbf{g} = (g_1, \ldots, g_k)$ we denote by $\mathbf{g}^{\mathbf{r}} = (g_1^{r_1}, \ldots, g_k^{r_k})$ component powers, and by $\mathsf{rt}(\hat{\mathbf{a}}, \mathbf{r})$ a tensor obtained from $\hat{\mathbf{a}}$ by cyclic rotating by r_i in corresponding index i. For example, in case of matrices i.e. rank 2 tensors, we have:

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n_1-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n_1-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_0-1,0} & a_{n_0-1,1} & \cdots & a_{n_0-1,n_1-1} \end{bmatrix}$$

$$\mathsf{rt}(\mathbf{A}, (r_0, r_1)) = \begin{bmatrix} a_{r_0,r_1} & a_{r_0,r_1+1} & \cdots & a_{r_0,r_1-1} \\ a_{r_0+1,r_1} & a_{r_0+1,r_1+1} & \cdots & a_{r_0+1,r_1-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r_0-1,r_1} & a_{r_0-1,r_1+1} & \cdots & a_{r_0-1,r_1-1} \end{bmatrix}$$

where indices are taken modulus n_i. Denote the security parameter throughout the paper: all known valid attacks against the cryptographic scheme under scope should take bit operations.

2.2 Machine Learning Algorithms

2.2.1 Logistic Regression

Logistic regression or logit model is a ML model used to predict the probability of occurrence of an event by fitting data to a logistic curve [Har01]. It is widely used in various fields including machine learning, biomedicine [LL90], genetics [LK12], and social sciences [GL09].

Throughout this paper, we treat the case of a binary dependent variable, represented by ± 1. Learning data consists of pairs $(\mathbf{x}_i, y_i)$ of a vector of co-variates $\mathbf{x}_i = (x_{i1}, ..., x_{if}) \in \mathbb{R}^f$ and a dependent variable $y_i \in \{\pm 1\}$. Logistic regression aims to find an optimal $\beta \in \mathbb{R}^{f+1}$ which maximizes the likelihood estimator

$$\prod_{i=1}^{n} \Pr(y_i|\mathbf{x}_i) = \prod_{i=1}^{n} \frac{1}{1 + \exp(-y_i(1, \mathbf{x}_i)^T \beta)},$$

or equivalently minimizes the loss function, defined as the negative log-likelihood:

$$J(\beta) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-\mathbf{z}_i^T \beta))$$

where $\mathbf{z}_i = y_i \cdot (1, \mathbf{x}_i)$ for $i = 1, \ldots, n$.

Gradient Descent

Gradient Descent (GD) is a method for finding a local extremum (minimum or maximum) of a function by moving along gradients. To minimize the function in the direction of the gradient, one-dimensional optimization methods are used.

For logistic regression, the gradient of the cost function with respect to β is computed by

$$\nabla J(\beta) = -\frac{1}{n} \sum_{i=1}^{n} \sigma(-\mathbf{z}_i^T \beta) \cdot \mathbf{z}_i$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$. Starting from an initial β_0, the gradient descent method at each step t updates the regression parameters using the equation

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \frac{\alpha_t}{n} \sum_{i=1}^{n} \sigma(-\mathbf{z}_i^T \beta^{(t)}) \cdot \mathbf{z}_i$$

where α_t is a learning rate at step t.

Nesterov's Accelerated Gradient

The method of GD can face a problem of zig-zagging along a local optima and this behavior of the method becomes typical if it increases the number of variables of an objective function. Many GD optimization algorithms are widely used to overcome this phenomenon. Momentum method, for example, dampens oscillation using the accumulated exponential moving average for the gradient of the loss function.

Nesterov's accelerated gradient [Nes83] is a slightly different variant of the momentum update. It uses moving average on the update vector and evaluates the gradient at this "looked-ahead" position. It guarantees a better rate of convergence $O(1/t^2)$ (vs. $O(1/t)$ of standard GD algorithm) after t steps theoretically, and consistently works slightly better in practice. Starting with a random initial $\mathbf{v}_0 = \beta_0$, the updated equations for Nesterov's Accelerated GD are as follows:

$$\begin{cases} \beta^{(t+1)} &= \mathbf{v}^{(t)} - \alpha_t \cdot \bigtriangledown J(\mathbf{v}^{(t)}), \\ \mathbf{v}^{(t+1)} &= (1-\gamma_t) \cdot \beta^{(t+1)} + \gamma_t \cdot \beta^{(t)}, \end{cases} \tag{2.2.1}$$

where $0 < \gamma_t < 1$ is a moving average smoothing parameter.

2.2.2 Deep Learning

Deep Learning is a set of machine learning algorithms (with a training, with partial involvement of a training, without a training, with reinforcement), based on the representation learning, rather than specialized algorithms for specific tasks. Many deep learning methods were known as early as the 1980s, but the results were unimpressive, while advances were made in the theory of artificial neural networks. And the computational power of the mid-2000s did not allow creating complex technological architectures of neural networks with sufficient productivity and did not allow to solve a wide range of tasks in computer vision, machine translation, speech recognition. However nowadays deep learning has shown amazing performance in diverse areas including academic research as well as industrial developments, and is applied to the increasing number of real-life applications.

DNN Classification Algorithm

We briefly describe the flow of DNN classification algorithm. DNN model consists of $\mathcal{L}+1$ number of fully connected (FC) layers. For simplicity we enumerate the layers starting from 0. Each layer contains n_l number of nodes for $l = 0, \ldots \mathcal{L}$. The layer 0 is input layer, the layer l is output layer, and the others are hidden layers. Each of the hidden layers and the output layer has a corresponding weight matrix $\mathbf{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$ and a bias vector $\mathbf{b}_l \in \mathbb{R}^{n_l}$. For the input vector $\mathbf{a}_0 \in \mathbb{R}^{n_0}$, we consecutively calculate the linear transformation part

$$\mathbf{z}_l = \mathbf{W}_l \mathbf{a}_{l-1} + \mathbf{b}_l$$

and for acitivation function g_l the activation part

$$\mathbf{a}_l = g_l(\mathbf{z}_l)$$

at the each hidden layer. For the output layer we calculate the linear transformation part $\mathbf{z}_{\mathcal{L}} = \mathbf{W}_{\mathcal{L}}\mathbf{a}_{\mathcal{L}-1} + \mathbf{b}_{\mathcal{L}}$ and the index of largest value in $\mathbf{z}_{\mathcal{L}}$ is the classification output.

RNN Classification Algorithm

Most neural networks currently used in research based on deep learning are deep sequential models. In the deep sequential models, a prediction value or vector corresponding to input data is computed by going through their critical operations (i.e., matrix multiplication, activation function). A RNN is one of the most popular deep sequential model. The RNN has recurrent operations to get knowledge from sequence data. Connections of neurons in the RNNs form computational directed graphs, and types of the directed graphs can diverse. RNN can learn dynamic temporal representation of input data by recurrently calculating internal states of the neurons.

We briefly describe the flow of RNN classification algorithm based on gated recurrent units (GRU). RNN model consists of $\mathcal{T}$ number of GRU layers and $\mathcal{L}+1$ number of FC layers.

Each GRU layer has hidden state $\mathbf{h}_{t-1}$ and input $\mathbf{x}_t$ as inputs, and hidden state $\mathbf{h}_t$ as output. Each of the GRU layers has a corresponding weight matrices $\mathbf{W}_z$, $\mathbf{U}_z$, $\mathbf{W}_r$, $\mathbf{U}_r$, $\mathbf{W}_h$, $\mathbf{U}_h$ and a bias vectors $\mathbf{b}_{\mathbf{W}_z}$, $\mathbf{b}_{\mathbf{U}_z}$, $\mathbf{b}_{\mathbf{W}_r}$, $\mathbf{b}_{\mathbf{U}_r}$, $\mathbf{b}_{\mathbf{W}_h}$, $\mathbf{b}_{\mathbf{U}_h}$.

$$\text{(update gate) } \mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{b}_{W_z} + \mathbf{U}_z\mathbf{x}_t + \mathbf{b}_{\mathbf{U}_z}) \tag{2.2.2}$$

$$\text{(reset gate) } \mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{b}_{\mathbf{W}_r} + \mathbf{U}_r\mathbf{x}_t + \mathbf{b}_{\mathbf{U}_r}) \tag{2.2.3}$$

$$\text{(hidden cell) } \tilde{\mathbf{h}_t} = \tanh(\mathbf{W}_h\mathbf{x}_t + \mathbf{b}_{\mathbf{W}_h} + \mathbf{r}_t * (\mathbf{U}_h\mathbf{x}_t + \mathbf{b}_{\mathbf{U}_h})) \tag{2.2.4}$$

$$\text{(output) } \mathbf{h}_t = \mathbf{z}_t * \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) * \tilde{\mathbf{h}_t} \tag{2.2.5}$$

For FC layers the algorithm is same as DNN case. Our input vector for FC is $\mathbf{h}_{\mathcal{T}}$.

2.3 The Cyclotomic Ring and Canonical Embedding

For an integer M consider its decomposition into primes $M = 2^{k_0} \cdot p_1^{k_1} \cdot \cdots \cdot p_s^{k_s} = \prod_{i=0}^{s} M_i$, where $M_0 = 2^{k_0}$, and $M_i = p_i^{k_i}$ for $i = 1, \ldots, s$. We will consider the cases $k_0 > 2$. Let $N_i = \phi(M_i) = (1 - \frac{1}{p_i})M_i$ for $i = 0, \ldots, s$, and $N = \phi(M) = \prod_{i=0}^{s} N_i$. Denote tensors $\hat{\mathbf{N}} = N_0 \times N_1 \times \cdots \times N_s$, $\hat{\mathbf{N}}_h = N_0/2 \times N_1 \times \cdots \times N_s$, and vectors $\mathbf{N} = (N_0, N_1, \ldots, N_s)$, $\mathbf{N}_h = (N_0/2, N_1, \ldots, N_s)$. Let $\Phi_M(x)$ be M-th cyclotomic polynomial. Let $\mathcal{R} = \mathbb{Z}[x]/\Phi_M(x)$ and $\mathcal{S} = \mathbb{R}[x]/\Phi_M(x)$. The canonical embedding τ_M of $a(x) \in \mathbb{Q}[x]/(\Phi_M(x))$ into $\mathbb{C}^N$ is the vector of evaluation values of $a(x)$ at the roots of $\Phi_M(x)$. We naturally extend it to the set of real polynomials $\mathcal{S}$, $\tau_M : \mathcal{S} \to \mathbb{C}^N$, so $\tau_M(a(x))$ will be defined as $(a(\xi_M^j))_{j \in \mathbb{Z}_M^\star} \in \mathbb{C}^N$ for any $a \in \mathcal{R}$ where $\xi_M = \exp(-2\pi i/M)$ is a primitive M-th roots of unity. The ℓ_∞-norm of $\tau_M(a(X))$ is called the *canonical embedding norm* of a, denoted by $\|a\|_\infty^{\mathsf{can}} = \|\tau_M(a)\|_\infty$. The canonical embedding norm $\|\cdot\|_\infty^{\mathsf{can}}$ satisfies the following properties:

- For all $a, b \in \mathcal{R}$, we have $\|a \cdot b\|_\infty^{\mathsf{can}} \leqslant \|a\|_\infty^{\mathsf{can}} \cdot \|b\|_\infty^{\mathsf{can}}$
- For all $a \in \mathcal{R}$, we have $\|a\|_\infty^{\mathsf{can}} \leqslant \|a\|_1$.
- For all $a \in \mathcal{R}$, we have $\|a\|_\infty \leqslant \|a\|_\infty^{\mathsf{can}}$.

Refer [DPSZ12] for more details.

2.4 m-RLWE Problem

Here we set up an underlying hardness problem.

Proposition 2.4.1. *If $M_0, M_1, \cdots, M_s$ are pairwisely coprime, then there is a ring isomorphism*

$$\mathcal{S} = \mathbb{R}[x]/\Phi_M(x) \cong \mathbb{R}[x_0, \ldots, x_s]/(\Phi_{M_0}(x_0), \ldots \Phi_{M_s}(x_s)) = \mathcal{S}'$$

and the map induces a ring isomorphism

$$\mathcal{R} = \mathbb{Z}[x]/\Phi_M(x) \cong \mathbb{Z}[x_0, \ldots, x_s]/(\Phi_{M_0}(x_0), \ldots \Phi_{M_s}(x_s)) = \mathcal{R}'.$$

We refers [BGV12] for RLWE-problem.

Definition 2.4.1. *A decisional RLWE problem $\mathsf{RLWE}_{\mathcal{R},\sigma}$ is a distinguishing problem between uniform distribution $(a(x), b(x))$ and a distribution $(a(x), a(x)s(x) + e(x))$ such that $a(x), b(x), s(x) \leftarrow \mathcal{R}/q\mathcal{R}$ and $e(x)$ is given by the image of a sample in $\mathcal{R}$ whose canonical embedding has components following a Gaussian distribution of variance σ^2 independently.*

Definition 2.4.2. *A decisional m-RLWE problem $\mathsf{m\text{-}RLWE}_{\mathcal{R}',\sigma'}$ is a distinguishing problem between uniform distribution $(a(\mathbf{x}), b(\mathbf{x}))$ and a distribution $(a(\mathbf{x}), a(\mathbf{x})s(\mathbf{x}) + e(\mathbf{x}))$ such that $a(\mathbf{x}), b(\mathbf{x}), s(\mathbf{x}) \in \mathcal{R}'/q\mathcal{R}'$ and $e(\mathbf{x})$ is* *given by the image of a sample in $\mathcal{R}'$ whose coefficients follow a Gaussian distribution of variance σ'^2 independently.*

The m-RLWE problem is suspected to be weak under evaluation attacks such as in case of the ring $\mathbb{Z}[x_0, x_1]/(\Phi_{M_0}(x_0), \Phi_{M_1}(x_1))$ for the powers-of-two M_0, M_1. The attack also seems to be expanding at least partially to the case $\gcd(M_i, M_j) > 1$. We design our scheme using relatively prime M_i's to avoid this case. Further we show the hardness of our case by devising a reduction from the original RLWE problem to m-RLWE problem with relatively prime M_i's.

Lemma 2.4.1. *(Hardness of m-RLWE) Let $\mathcal{R}$ and $\mathcal{R}'$ be given as proposition 2.4.1. Then $\mathsf{RLWE}_{\mathcal{R},\sigma}$ reduces to $\mathsf{m\text{-}RLWE}_{\mathcal{R}',c\sigma}$, where*

$$c^2 = \prod_{i=1}^{s} \left(\frac{p_i - 1}{p_i} \times (2 + \frac{12\pi}{p_i}) \right)$$

In particular, c is less than $\sqrt{3}$ if $p_i \geqslant 41 > 12\pi$ or $p_i = 3, 37$. As p increases, c tends to be $\sqrt{2}$. The followings are approximations of c :

$$\begin{aligned} (p_i, c) \quad = \quad & (5, 2.8), (7, 2.6), (11, 2.3), (13, 2.2), (17, 2.0), \\ & (19, 2.0), (23, 1.9), (29, 1.8), (31, 1.9) \end{aligned}$$

For $p_i = 3$ and 37, the norm is given $2/\sqrt{3}$ and bounded by 1.72, respectively.

Remark 2.4.1. *Our implementation covers cases of $s = 1$ and $p = 17, 257$. In these cases, c^2 is approximately $2.06, 2.01$, respectively.*

Remark 2.4.2. *Since $\|a\|_2 \leqslant \|a\|_\infty$, the distinguishing problem given by ℓ_∞ norm is at least as hard as the problem given by ℓ_2 norm. In other words,* *m-RLWE sample can be chosen by an error distribution following ℓ_∞ norm rather than ℓ_2 norm. From now on, the norm of m-RLWE samples, or their errors, are measured by ℓ_∞ norm.*

2.5 HEAAN Scheme

The following is the instantiation of the RLWE-based `HEAAN` scheme [CKKS16, CKKS17]. For a power-of-two $N > 4$ and $M = 2N$, denote $\Phi_M(x) = (x^N + 1)$, $\mathcal{R} = \mathbb{Z}[x]/\Phi_M(x)$. For a positive integer ℓ, denote $\mathcal{R}_\ell = \mathcal{R}/2^\ell\mathcal{R} = \mathbb{Z}_{2^\ell}[x]/\Phi_M(x)$ the residue ring of $\mathcal{R}$ modulo 2^ℓ. The variant of the canonical embedding map defined as

$$\tau'_{N/2} : m(x) \to \mathbf{z} = (z_0, \ldots, z_{N/2-1})$$

such that $z_j = m(\xi_M^{5^j})$.

Sparse packing. For a power-of-two $n \leqslant N/2$ consider a subring $\mathcal{R}^{(n)} = \mathbb{Z}[x']/(x'^{2n} + 1) \subset \mathcal{R}$ where $x' = x^{N/(2n)}$. For $\mathcal{R}^{(n)}$ define an isomorphism $\tau'_n : m(x') = m(x^{N/(2n)}) \to \mathbf{z} = (z_0, \ldots, z_{n-1})$ such that $z_j = m(\xi'_j)$, where $\xi'_j = \xi_j^{N/(2n)}$. We can pack n complex values via isomorphism τ'^{-1}_n. In this case if we apply $\tau'_{N/2}$ to $m(x') \in \mathcal{R}$ we will get a vector obtained from $\mathbf{z}$ by concatenating itself N/n times. For a message $m(x)$ encoding a vector $\mathbf{z}$ and a ciphertext ct encrypting $m(x)$, ct is also said to be encrypting vector $\mathbf{z}$.

- $\underline{\texttt{HEAAN.KeyGen}(1^\lambda)}$.
    - For an integer L that corresponds to the largest ciphertext modulus level, given the security parameter λ, output the ring dimension N which is a power of two.

  - Set the small distributions $\chi_{key}, \chi_{err}, \chi_{enc}$ over $\mathcal{R}$ for secret, error, and encryption, respectively.
  - Sample a secret $s \leftarrow \chi_{key}$, a random $a \leftarrow \mathcal{R}_L$ and an error $e \leftarrow \chi_{err}$. Set the secret key as $\mathsf{sk} \leftarrow (s, 1)$ and the public key as $\mathsf{pk} \leftarrow (a, b) \in \mathcal{R}_L^2$ where $b \leftarrow -as + e \pmod{2^L}$.

- <u>`HEAAN.KSGen`$_{\mathsf{sk}}(s')$</u>. For $s' \in \mathcal{R}$, sample a random $a' \leftarrow \mathcal{R}_{2\cdot L}$ and an error $e' \leftarrow \chi_{err}$. Output the switching key as $\mathsf{swk} \leftarrow (a', b') \in \mathcal{R}_{2\cdot L}^2$ where $b' \leftarrow -a's + e' + 2^L s' \pmod{2^{2\cdot L}}$.

  - Set the evaluation key as $\mathsf{evk} \leftarrow$ `HEAAN.KSGen`$_{\mathsf{sk}}(s^2)$.

- <u>`HEAAN.Encode`$(\mathbf{z}, p)$</u>. For a vector $\mathbf{z} \in \mathbb{C}^n$, with of a power-of-two $n \leqslant N/2$ and an integer $p < L$ corresponding to precision bits, output the polynomial $m \leftarrow \tau'^{-1}_n(2^p \cdot \mathbf{z}) \in \mathcal{R}$.

- <u>`HEAAN.Decode`(m, p)</u>. For a plaintext $m \in \mathcal{R}$, the encoding of a vector consisting of a power-of-two $n \leqslant N/2$ complex messages and precision bits p, output the vector $\mathbf{z} \leftarrow \tau'_n(m/2^p) \in \mathbb{C}^n$.

- <u>`HEAAN.Enc`$_{\mathsf{pk}}(m)$</u>. For $m \in \mathcal{R}$, sample $v \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$. Output $v \cdot \mathsf{pk} + (e_0, e_1 + m) \pmod{2^L}$.

- <u>`HEAAN.Dec`$_{\mathsf{sk}}(\mathsf{ct})$</u>. For $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}_\ell^2$, output $c_0 \cdot s + c_1 \pmod{2^\ell}$.

- <u>`HEAAN.Add`$(\mathsf{ct}_1, \mathsf{ct}_2)$</u>. For $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{R}_\ell^2$, output $\mathsf{ct}_{\mathsf{add}} \leftarrow \mathsf{ct}_1 + \mathsf{ct}_2 \pmod{2^\ell}$.

- <u>`HEAAN.CMult`$_{\mathsf{evk}}(\mathsf{ct}, \mathbf{c}, p)$</u>. For $\mathsf{ct} \in \mathcal{R}_\ell^2$ and $\mathbf{c} \in \mathbb{C}^n$, compute $c \leftarrow$ `HEAAN.Encode`$(\mathbf{c}; p)$ and output $\mathsf{ct}' \leftarrow c \cdot \mathsf{ct} \pmod{2^\ell}$.

- <u>`HEAAN.PolyMult`$_{\mathsf{evk}}(\mathsf{ct}, g, p)$</u>. For $\mathsf{ct} \in \mathcal{R}_\ell^2$ and $g \in \mathcal{R}_\ell$, output $\mathsf{ct}' \leftarrow g \cdot \mathsf{ct} \pmod{2^\ell}$.

- $\underline{\texttt{HEAAN.Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)}$. For $\mathsf{ct}_1 = (a_1, b_1), \mathsf{ct}_2 = (a_2, b_2) \in \mathcal{R}_\ell^2$, let $(d_0, d_1, d_2) = (a_1 \cdot a_2, a_1 \cdot b_2 + a_2 \cdot b_1, b_1 \cdot b_2) \pmod{2^\ell}$. Output $\mathsf{ct}_{\mathsf{mult}} \leftarrow (d_1, d_2) + \lfloor 2^{-L} \cdot d_0 \cdot \mathsf{evk} \rceil \pmod{2^\ell}$.

- $\underline{\texttt{HEAAN.ReScale}(\mathsf{ct}, p)}$. For a ciphertext $\mathsf{ct} \in \mathcal{R}_\ell^2$ and an integer p, output $\mathsf{ct}' \leftarrow \lfloor 2^{-p} \cdot \mathsf{ct} \rceil \pmod{2^{\ell - p}}$.

- $\underline{\texttt{HEAAN.ModDown}(\mathsf{ct}, p)}$. For a ciphertext $\mathsf{ct} \in \mathcal{R}_\ell^2$ and an integer p, output $\mathsf{ct}' \leftarrow \mathsf{ct} \pmod{2^{\ell - p}}$.

For an integer k co-prime with M, let $\kappa_k : m(x) \to m(x^k) \pmod{\Phi_M(x)}$. This transformation can be used to provide more functionalities on plaintext slots.

- $\underline{\texttt{HEAAN.Conjugate}_{\mathsf{cjk}}(\mathsf{ct})}$. Set the conjugation key as $\mathsf{cjk} \leftarrow \texttt{HEAAN.KSGen}_{\mathsf{sk}}(\kappa_{-1}(s))$. For $\mathsf{ct} = (a, b) \in \mathcal{R}_\ell^2$ encrypting vector $\mathbf{z}$, let $(a', b') = (\kappa_{-1}(a), \kappa_{-1}(b)) \pmod{2^\ell}$. Output $\mathsf{ct}_{\mathsf{cj}} \leftarrow (0, b') + \lfloor 2^{-L} \cdot a' \cdot \mathsf{cjk} \rceil \pmod{2^\ell}$. $\mathsf{ct}_{\mathsf{cj}}$ is a ciphertext encrypting $\bar{\mathbf{z}}$ - the conjugated plaintext vector of ct.

- $\underline{\texttt{HEAAN.Rotate}_{\mathsf{rtk}}(\mathsf{ct}; r)}$. Set the rotation key as $\mathsf{rtk} \leftarrow \texttt{HEAAN.KSGen}_{\mathsf{sk}}(\kappa_{5^r}(s))$. For $\mathsf{ct} = (a, b) \in \mathcal{R}_\ell^2$ encrypting vector $\mathbf{z}$, let $(a', b') = (\kappa_{5^r}(a), \kappa_{5^r}(b)) \pmod{2^\ell}$. Output $\mathsf{ct}_{\mathsf{rt}} \leftarrow (0, b') + \lfloor 2^{-L} \cdot a' \cdot \mathsf{rtk} \rceil \pmod{2^\ell}$. $\mathsf{ct}_{\mathsf{rt}}$ is a ciphertext encrypting $\mathsf{rt}(\mathbf{z}, r) = (z_r, \ldots, z_{n-1}, z_0, \ldots, z_{r-1})$ - rotated by r positions plaintext vector of ct.

Refer [CKKS17, CHK+18] for the technical details and noise analysis.

2.5.1 Bootstrapping for HEAAN

Consider a ciphertext $\mathsf{ct} \in \mathcal{R'}_\ell^2$, an encryption of message $m(x)$ encoding a vector of size n. Then the coefficients of $m(x)$ are non-zero only at degrees

$k \cdot \frac{N}{2n}$ for $k = 1, 2, \cdots, 2n-1$. Consider ct as an element of $\mathcal{R}'^2_L$ for $L \gg \ell$. We can treat ct as an encryption of $m(x) + 2^\ell \cdot I(x)$ in $\mathcal{R}'_L$ i.e. $\mathtt{Dec}(\mathsf{ct}) = m(x) + e(x) + 2^\ell \cdot I(x) \pmod{\mathcal{R}}$ for some polynomial $I(x)$ of degree $< N$. With a choice of sparse sk, coefficients of $I(x)$ are bounded with some constant. Now the bootstrapping procedure is defined as followings.

- $\underline{\mathtt{HEAAN.SubSum}(\mathsf{ct}, n)}$ As the number of slots is n, then nonzero coefficients of $m(x)$ are only at degrees $k \cdot \frac{N}{2n}$. The output encrypts a message $m(x) + 2^\ell \cdot I'(x)$ where $I'(x)$ derived from $I(x)$ by vanishing the coefficients at degrees other than multiples of $\frac{N}{2n}$.

Algorithm 1 SubSum procedure

```
1: procedure SubSum($\mathsf{ct} \in \mathcal{R}'^2_L, n \mid N/2, n \geqslant 1$)
2:     $\mathsf{ct}' \leftarrow \mathsf{ct}$
3:     for $j = 0$ to $\log(\frac{N}{2n}) - 1$ do
4:         $\mathsf{ct}_j \leftarrow$ HEAAN.Rotate($\mathsf{ct}'; 2^j \cdot n$)
5:         $\mathsf{ct}' \leftarrow$ HEAAN.Add($\mathsf{ct}', \mathsf{ct}_j$)
6:     end for
7:     $\mathsf{ct}'' \leftarrow$ HEAAN.ReScale($\mathsf{ct}'; \log(\frac{N}{2n})$)
8:     return $\mathsf{ct}''$
9: end procedure
```

Let $m(x) + 2^\ell \cdot I'(x) = \sum_{j=0}^{N-1} t_j x^j$ encoding vector $\mathbf{z} = (z_0, \ldots, z_{n-1})$. Then for the following matrix Σ we have equation:

$$\Sigma \cdot \mathbf{z} = \begin{bmatrix} \xi_0^0 & \xi_0^1 & \cdots & \xi_0^{n-1} \\ \xi_1^0 & \xi_1^1 & \cdots & \xi_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \xi_{n-1}^0 & \xi_{n-1}^1 & \cdots & \xi_{n-1}^{n-1} \end{bmatrix} \cdot \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} t'_0 & + & i \cdot t'_n \\ t'_1 & + & i \cdot t'_{n+1} \\ & \vdots & \\ t'_{n-1} & + & i \cdot t'_{2n-1} \end{bmatrix} \tag{2.5.6}$$

where $\xi_j = \exp(\frac{2\pi i \cdot 5^j}{2n})$ and $t'_k = t_{k \cdot \frac{N}{2n}}$.

- <u>`HEAAN.SlotToCoeff(ct)`</u>. Multiply `ct` by a matrix Σ^{-1}. The output is the ciphertext that encrypts coefficients of $m(x) + 2^\ell \cdot I'(x)$ in real and imaginary parts: $t_{k \cdot \frac{N}{2n}} + i \cdot t_{(k+n) \cdot \frac{N}{2n}}$ in slot k for $k = 1, 2, \cdots, n-1$.
- <u>`HEAAN.RemoveIPart(ct)`</u> Extract real and imaginary parts of slots and evaluate the polynomial function, close to $f(x) = \frac{1}{2\pi i} \exp(\frac{2\pi i x}{2^\ell})$ for both parts. Combine the two ciphertexts to obtain a ciphertext that encrypts coefficients of $m(x)$ in real and imaginary parts: $m_{k \cdot \frac{N}{2n}} + i \cdot m_{(k+n) \cdot \frac{N}{2n}}$ in slot k for $k = 1, 2, \cdots, n-1$.
- <u>`HEAAN.CoeffToSlot(ct)`</u> Multiply `ct` by a matrix Σ^{-1}. The result is a ciphertext that encrypts $m(x)$ in a higher power-of-two modulus $L' \gg \ell$

`SlotToCoeff` and `CoeffToSlot` parts of the algorithm require $O(\sqrt{n})$ ciphertext rotations and $O(n)$ constant multiplications when performing so-called 'baby-giant step' optimization. The algorithm also requires to store $O(\sqrt{n})$ rotations keys, which is impractical for large number of slots. For more details refer to [CHK+18, CKKS16].

Chapter 3

MHEAAN Scheme

3.1 MHEAAN Scheme

3.1.1 Structure of MHEAAN

In this section we will use notations from Section 2.3. MHEAAN is a generalization of HEAAN to a case of non power-of-two M. The encryption process in MHEAAN scheme can be shown in the following outline: we encode a tensor of complex values of size $\hat{\mathbf{N}}$ using ${\tau'}^{-1}_{\mathbf{N}_h}$ into $m(\mathbf{x}) \in \mathcal{R}'$. We mask the result with m-RLWE instance $\big(a(\mathbf{x}), b(\mathbf{x})\big)$ in the corresponding ring $\mathcal{R}'_\ell$. For a message $m(\mathbf{x})$ encoding a tensor $\hat{\mathbf{z}}$ and a ciphertext ct encrypting $m(\mathbf{x})$, we also say that ct encrypts tensor $\hat{\mathbf{z}}$.

sparse packing. For divisors n_0 of $N_0/2$ and n_i of N_i for $i = 1, \ldots, s$, denote $\hat{\mathbf{n}} = n_0 \times n_1 \times \cdots \times n_s$, $\mathbf{n} = (n_0, n_1, \ldots, n_s)$. We can imitate sparse tensor packing similar to the HEAAN case. We can encode a sparse tensor of complex values of size $\hat{\mathbf{n}}$ using ${\tau'}^{-1}_{\mathbf{N}_h}$ applied to a tensor of size $\hat{\mathbf{N}}_h$ consisting of same blocks of size $\hat{\mathbf{n}}$. We denote this embedding as ${\tau'}^{-1}_{\mathbf{n}}$.

We can treat HEAAN scheme as a special case of MHEAAN with $s = 0$:

$$\mathbf{z} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{n_0-1} \end{bmatrix} \xrightarrow[\texttt{Encode}]{\tau'^{-1}_{n_0}} m(x) \xrightarrow[\texttt{Enc}]{\mathsf{RLWE}} \mathsf{ct}$$

and for two-dimensional packing ($s = 1$) we have:

$$\mathbf{Z} = \begin{bmatrix} z_{0,0} & z_{0,1} & \cdots & z_{n_1-1} \\ z_{1,0} & z_{1,1} & \cdots & z_{1,n_1-1} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n_0-1,0} & z_{n_0-1,1} & \cdots & z_{n_0-1,n_1-1} \end{bmatrix} \xrightarrow[\texttt{Encode}]{\tau'^{-1}_{n_0,n_1}} m(x_0, x_1) \xrightarrow[\texttt{Enc}]{m-\mathsf{RLWE}} \mathsf{ct}$$

3.1.2 Concrete Construction

For a positive integer ℓ denote $\mathcal{R}'_\ell = \mathcal{R}'/2^\ell\mathcal{R}'$ the residue ring of $\mathcal{R}'$ modulo 2^ℓ. For a real $\sigma > 0$, $\mathcal{DG}(\sigma^2)$ samples a multivariate polynomial in $\mathcal{R}'$ by drawing its coefficient independently from the discrete Gaussian distribution of variance σ^2. For an positive integer h, $\mathcal{HWT}(h)$ is the set of signed binary tensors in $\{0, \pm 1\}^{\hat{\mathbf{N}}}$ whose Hamming weight is exactly h. For a real $0 \leqslant \rho \leqslant 1$, the distribution $\mathcal{ZO}(\rho)$ draws each entry in the tensor from $\{0, \pm 1\}^{\hat{\mathbf{N}}}$, with probability $\rho/2$ for each of -1 and $+1$, and probability being zero $1 - \rho$.

- MHEAAN.KeyGen(1^λ).

  - Given the security parameter λ, set an integer M that corresponds to a cyclotomic ring, an integer L that corresponds to the largest ciphertext modulus level and distribution parameters (ρ, σ, h).

- Set the distributions $\chi_{enc} = \mathcal{ZO}(\rho)$, $\chi_{err} = \mathcal{DG}(\sigma)$, $\chi_{key} = \mathcal{HWT}(h)$ over $\mathcal{R}$ for secret, error, and encryption, respectively.
- Sample a secret $s \leftarrow \chi_{key}$, a random $a \leftarrow \mathcal{R}'_L$ and an error $e \leftarrow \chi_{err}$. Set the secret key as $\mathsf{sk} \leftarrow (s, 1)$ and the public key as $\mathsf{pk} \leftarrow (a, b) \in \mathcal{R}'^2_L$ where $b \leftarrow -a \cdot s + e \pmod{2^L}$.

- $\underline{\texttt{MHEAAN.KSGen}_{\mathsf{sk}}(s)}$. For $\mathsf{s} \in \mathcal{R}'$, sample a random $a \leftarrow \mathcal{R}'_{2\cdot L}$ and an error $e \leftarrow \chi_{err}$. Output the switching key as $\mathsf{swk} \leftarrow (a, b) \in \mathcal{R}'^2_{2\cdot L}$ where $b \leftarrow -a \cdot s + e + 2^L s \pmod{\mathcal{R}'_{2\cdot L}}$.

  - Set the evaluation key as $\mathsf{evk} \leftarrow \texttt{MHEAAN.KSGen}_{\mathsf{sk}}(s^2)$.

- $\underline{\texttt{MHEAAN.Encode}(\hat{\mathbf{z}}, p)}$. For a tensor $\hat{\mathbf{z}} \in \mathbb{C}^{\hat{\mathbf{n}}}$, an integer $p < L - 1$ corresponding to precision bits, output the two-degree polynomial $m \leftarrow \tau'_{\mathbf{n}}(2^p \cdot \hat{\mathbf{z}}) \in \mathcal{R}'$.

- $\underline{\texttt{MHEAAN.Decode}(m, p)}$. For a plaintext $m \in \mathcal{R}'$, the encoding of a tensor of complex messages $\hat{\mathbf{z}} \in \mathbb{C}^{\hat{\mathbf{n}}}$, precision bits p, output the tensor $\hat{\mathbf{z}}' \leftarrow \tau'^{-1}_{\mathbf{n}}(m/2^p) \in \mathbb{C}^{\hat{\mathbf{n}}}$.

- $\underline{\texttt{MHEAAN.Enc}_{\mathsf{pk}}(m)}$. For $m \in \mathcal{R}'$, sample $v \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$. Output $\mathsf{ct} = v \cdot \mathsf{pk} + (e_0, e_1 + m) \pmod{\mathcal{R}'_L}$.

- $\underline{\texttt{MHEAAN.Dec}_{\mathsf{sk}}(\mathsf{ct})}$. For $\mathsf{ct} = (c_0, c_1) \in \mathcal{R}'^2_\ell$, output $c_0 \cdot s + c_1 \pmod{\mathcal{R}'_\ell}$.

- $\underline{\texttt{MHEAAN.Add}(\mathsf{ct}_1, \mathsf{ct}_2)}$. For $\mathsf{ct}_1, \mathsf{ct}_2 \in \mathcal{R}'^2_\ell$ - encryption of tensors $\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2 \in \mathbb{C}^{\hat{\mathbf{n}}}$ output $\mathsf{ct}_{\mathsf{add}} \leftarrow \mathsf{ct}_1 + \mathsf{ct}_2 \pmod{2^\ell}$. $\mathsf{ct}_{\mathsf{add}}$ is a ciphertext encrypting tensor $\hat{\mathbf{z}}_1 + \hat{\mathbf{z}}_2$.

- $\underline{\texttt{MHEAAN.CMult}_{\mathsf{evk}}(\mathsf{ct}, \mathbf{C}, p)}$. For $\mathsf{ct} \in \mathcal{R}_\ell^2$ - encryption of $\hat{\mathbf{z}} \in \mathbb{C}^{\hat{\mathbf{n}}}$, and a constant tensor $\hat{\mathbf{c}} \in \mathbb{C}^{\hat{\mathbf{n}}}$, compute $c \leftarrow \texttt{MHEAAN.Encode}(\hat{\mathbf{c}}, p)$ the encoding of $\hat{\mathbf{c}}$ and output $\mathsf{ct}_{\mathsf{cmult}} \leftarrow c \cdot \mathsf{ct} \pmod{\mathcal{R}'_\ell}$. $\mathsf{ct}_{\mathsf{cmult}}$ is a ciphertext encrypting tensor $\hat{\mathbf{z}} \odot \hat{\mathbf{c}}$.

- $\underline{\texttt{MHEAAN.PolyMult}_{\mathsf{evk}}(\mathsf{ct}, g, p)}$. For $\mathsf{ct} \in \mathcal{R}_\ell^2$ - encryption of $\hat{\mathbf{z}} \in \mathbb{C}^{\hat{\mathbf{n}}}$, and a constant $g \in \mathcal{R}_\ell$ output $\mathsf{ct}_{\mathsf{cmult}} \leftarrow c \cdot \mathsf{ct} \pmod{\mathcal{R}'_\ell}$. $\mathsf{ct}_{\mathsf{cmult}}$ is a ciphertext encrypting tensor $\hat{\mathbf{z}} \odot \hat{\mathbf{c}}$, where $\hat{\mathbf{c}}$ is decoding of g.

Multiplication by polynomial is similar to a constant multiplication, however in the next section we will show why it is important to define it separately.

- $\underline{\texttt{MHEAAN.Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)}$. For $\mathsf{ct}_1 = (a_1, b_1), \mathsf{ct}_2 = (a_2, b_2) \in \mathcal{R}'^2_\ell$ - encryptions of tensors $\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2 \in \mathbb{C}^{\hat{\mathbf{n}}}$, let $(d_0, d_1, d_2) = (a_1 a_2,\ a_1 b_2 + a_2 b_1, b_1 b_2) \pmod{\mathcal{R}'_\ell}$. Output

  $$\mathsf{ct}_{\mathsf{mult}} \leftarrow (d_1, d_2) + \lfloor 2^{-L} \cdot d_0 \cdot \mathsf{evk} \rceil \pmod{\mathcal{R}'_\ell}$$

  $\mathsf{ct}_{\mathsf{mult}}$ is a ciphertext encrypting tensor $\hat{\mathbf{z}}_1 \odot \hat{\mathbf{z}}_2$.

- $\underline{\texttt{MHEAAN.ReScale}(\mathsf{ct}, p)}$. For a ciphertext $\mathsf{ct} \in \mathcal{R}'^2_\ell$ and an integer p, output $\mathsf{ct}' \leftarrow \lfloor 2^{-p} \cdot \mathsf{ct} \rceil \pmod{\mathcal{R}'_{\ell - p}}$.

For an integer vector $\mathbf{k} = (k_0, \ldots, k_s)$ with k_i co-prime with M_i, let

$$\kappa_{\mathbf{k}} : m'(\mathbf{x}) \to m'(\mathbf{x}^{\mathbf{k}}) \pmod{\mathcal{R}'_\ell}$$

This transformation can be used to provide conjugation and rotations in different dimensions on the plaintext matrix.

- $\underline{\texttt{MHEAAN.Conjugate}_{\mathsf{cjk}}(\mathsf{ct})}$. Set the conjugation key as

  $$\mathsf{cjk} \leftarrow \texttt{MHEAAN.KSGen}_{\mathsf{sk}}(\kappa_{-\mathbf{1}}(s))$$

  For $\mathsf{ct} = (a, b) \in \mathcal{R}'^2_\ell$ encrypting matrix $\mathbf{Z}$, let

  $$(a', b') = (\kappa_{-\mathbf{1}}(a), \kappa_{-\mathbf{1}}(b)) \pmod{\mathcal{R}'_\ell}$$

  Output

  $$\mathsf{ct}_{\mathsf{cj}} \leftarrow (0, b') + \lfloor 2^{-L} \cdot a' \cdot \mathsf{cjk} \rceil \pmod{\mathcal{R}'_\ell}$$

  $\mathsf{ct}_{\mathsf{cj}}$ is a ciphertext encrypting $\bar{\hat{\mathbf{z}}}$ - the conjugated plaintext tensor of ct.

- $\underline{\texttt{MHEAAN.Rotate}_{\mathsf{rtk}}(\mathsf{ct}; \mathbf{r})}$. Set the rotation key as

  $$\mathsf{rtk} \leftarrow \texttt{MHEAAN.KSGen}_{\mathsf{sk}}(\kappa_{\mathbf{g}^{\mathbf{r}}}(s))$$

  For $\mathsf{ct} = (a, b) \in \mathcal{R}'^2_\ell$ encrypting matrix $\mathbf{Z}$, let $(a', b') = (\kappa_{\mathbf{g}^{\mathbf{r}}}(a), \kappa_{\mathbf{g}^{\mathbf{r}}}(b))$ $\pmod{\mathcal{R}'_\ell}$. Output $\mathsf{ct}_{\mathsf{rt}} \leftarrow (0, b') + \lfloor 2^{-L} \cdot a' \cdot \mathsf{rtk} \rceil \pmod{\mathcal{R}'_\ell}$. $\mathsf{ct}_{\mathsf{rt}}$ is a ciphertext encrypting $\mathsf{rt}(\hat{\mathbf{z}}, \mathbf{r})$ - cyclic rotated plaintext tensor by r_i in i-th dimension.

Throughout this paper, we use real polynomials as plaintexts for convenience of analysis. A ciphertext $\mathsf{ct} \in \mathcal{R}'^2_\ell$ will be called a valid encryption of $m \in \mathcal{S}$ with the encryption noise bounded by δ, and plaintext bounded by μ, if $\langle \mathsf{ct}, \mathsf{sk} \rangle = m + e \pmod{\mathcal{R}'_\ell}$ for some polynomial $e \in \mathcal{S}$ with $\|e\|_\infty^{\mathsf{can}} < \delta$ and $\|m\|_\infty^{\mathsf{can}} < \mu$. We will use a corresponding tuple $(\mathsf{ct}, \delta, \mu, \ell)$ for such an encryption of m. The following lemmas give upper bounds on noise growth after encryption, rescaling and homomorphic operations. Refer to

Appendix A for proofs.

Lemma 3.1.1 (Encoding & Encryption). *For* $m \leftarrow \texttt{MHEAAN.Encode}(\hat{\mathbf{z}}, p)$ *and* $\mathsf{ct} \leftarrow \texttt{MHEAAN.Enc}_{\mathsf{pk}}(m)$ *the encryption noise is bounded by* $\delta_{\mathsf{clean}} = 8\sqrt{2} \cdot \sigma N + 6\sigma\sqrt{N} + 16\sigma\sqrt{hN}$.

Lemma 3.1.2 (Rescaling). *Let* $(\mathsf{ct}, \delta, \mu, \ell)$ *be a valid encryption of* m *and* $\mathsf{ct}' \leftarrow \texttt{MHEAAN.ReScale}(\mathsf{ct}, p)$. *Then* $(\mathsf{ct}', \delta/2^p + \delta_{\mathsf{scale}}, \mu/2^p, \ell - p)$ *is a valid encryption of* $m/2^p$ *where* $\delta_{\mathsf{scale}} = 6\sqrt{N/12} + 16\sqrt{hN/12}$

Remark 3.1.1. *We can slightly change the public key generation and the encryption process to obtain a ciphertext with initial noise reduced from* δ_{clean} *to almost* δ_{scale}. *For this we generate public key in* $\mathcal{R}'^2_{2L}$ *instead of* $\mathcal{R}'^2_L$. *Also in the encryption process we encode the plaintext* m *with* $p + L$ *precision bits, instead of* p *bits with the following rescaling of the encryption* ct *of* m *by* L *bits. With a slightly slower encryption process we end up with a valid encryption in* $\mathcal{R}'^2_L$, *with the initial noise bounded by* $\delta_{\mathsf{clean}}/2^L + \delta_{\mathsf{scale}} \approx \delta_{\mathsf{scale}}$.

Lemma 3.1.3 (Addition & Multiplication). *Let* $(\mathsf{ct}_i, \delta_i, \mu_i, \ell)$ *be encryptions of* $m_i \in \mathcal{R}'$ *and let*

$$\mathsf{ct}_{\mathsf{add}} \leftarrow \texttt{MHEAAN.Add}(\mathsf{ct}_1, \mathsf{ct}_2)$$

and

$$\mathsf{ct}_{\mathsf{mult}} \leftarrow \texttt{MHEAAN.Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)$$

then

$$(\mathsf{ct}_{\mathsf{add}}, \delta_1 + \delta_2, \mu_1 + \mu_2, \ell)$$

and

$$(\mathsf{ct}_{\mathsf{mult}}, \mu_1 \cdot \delta_2 + \mu_2 \cdot \delta_1 + \delta_1 \cdot \delta_2 + \delta_{\mathsf{mult}}, \mu_1 \cdot \mu_2, \ell)$$

are valid encryptions of $m_1 + m_2$ *and* $m_1 \cdot m_2$*, respectively, where* $\delta_{\mathsf{ks}} = 8\sigma N/\sqrt{3}$ *and* $\delta_{\mathsf{mult}} = 2^{\ell-L} \cdot \delta_{\mathsf{ks}} + \delta_{\mathsf{scale}}$.

Lemma 3.1.4 (Conjugation & Rotation)**.** *Let* $(\mathsf{ct}, \delta, \mu, \ell)$ *be encryption of* $m \in \mathcal{R}'$ *that encodes tensor* $\hat{\mathbf{z}}$*,* $\mathbf{r}$*- integer vector, and let*

$$\mathsf{ct}_{\mathsf{rt}} = \mathit{MHEAAN.Rotate}_{\mathsf{rtk}}(\mathsf{ct}; \mathbf{r})$$

$$\mathsf{ct}_{\mathsf{cj}} = \mathit{MHEAAN.Conjugate}_{\mathsf{cjk}}(\mathsf{ct})$$

then $(\mathsf{ct}_{\mathsf{rt}}, \delta + \delta_*, \mu, \ell)$ *and* $(\mathsf{ct}_{\mathsf{cj}}, \delta + \delta_*, \mu, \ell)$ *are valid encryptions of tensors* $\mathsf{rt}(\hat{\mathbf{z}}, \mathbf{r})$ *and* $\bar{\hat{\mathbf{z}}}$ *respectively where where* $\delta_{\mathsf{ks}} = 8\sigma N/\sqrt{3}$ *and* $\delta_* = 2^{\ell-L} \cdot \delta_{\mathsf{ks}} + \delta_{\mathsf{scale}}$

Relative Error As discussed in [CKKS17] the decryption of a ciphertext is an approximate value of plaintext, so it needs to dynamically manage the bound of noise of ciphertext. It is sometimes convenient to consider the relative error defined by $\beta = \delta/\mu$. When two ciphertexts with relative errors $\beta_i = \delta_i/\mu_i$ are added the output ciphertext has a relative error bounded by $\max_i(\beta_i)$. When two ciphertexts are multiplied with the following rescaling by p bits the output ciphertext has a relative error bounded by

$$\beta' = \beta_1 + \beta_2 + \beta_1\beta_2 + \frac{\delta_{\mathsf{mult}} + 2^{-p} \cdot \delta_{\mathsf{scale}}}{\mu_1\mu_2}$$

according to Lemmas 3.1.2 and 3.1.3. This relative error is close to $\beta_1 + \beta_2$ which is similar to the case of unencrypted floating-point multiplication under an appropriate choice of parameters.

For convenience of analysis, we will assume that for two ciphertexts with relatives errors β_1 and β_2 the relative error after multiplication and rescaling is bounded by $\beta_1 + \beta_2 + \beta^*$ for some fixed β^*

3.2 Bootstrapping for MHEAAN

Similar to HEAAN scheme, consider a ciphertext $\mathtt{ct} \in \mathcal{R}'^2_\ell$ as an element of $\mathcal{R}'^2_L$ for $L \gg \ell$, with $\mathtt{Dec}(\mathtt{ct}) = m(\mathbf{x}) + e(\mathbf{x}) + 2^\ell \cdot I(\mathbf{x}) \pmod{\mathcal{R}'_L}$. For simplicity we only consider boostrapping for full packing. However some cases of sparse packing (as sparse packing in dimension corresponding to M_0) could be achieved using similar to HEAAN case techniques.

- MHEAAN.SlotToCoeff(ct). From the equation A.0.1 (in appendix) we notice that linear transformation can be split into consecutive linear transformations consisting of Σ from the equation 2.5.6 and Σ'_i from the equations A.0.2 applying to different dimensions i of $m(\mathbf{x})$. Output is the ciphertext that encrypts coefficients of $m(\mathbf{x})+e(\mathbf{x})+2^\ell \cdot I(\mathbf{x})$ in real and imaginary parts.

- MHEAAN.RemoveIPart(ct) This part of algorithm is same to HEAAN. Extract real and imaginary parts of slots, evaluate polynomial function, close to $f(x) = \frac{1}{2\pi i}\exp(\frac{2\pi i x}{2^\ell})$ for both parts. Combine two ciphertexts to obtain ciphertext that encrypts coefficients of $m(\mathbf{x})$ in real and imaginary parts.

- HEAAN.CoeffToSlot(ct) Apply consecutively linear transformations Σ^{-1} and Σ_i^{-1}. The result is a ciphertext that encrypts same vector as initial ct in a higher modulus $\mathcal{R}'^2_{L'}$ with $L' \gg \ell$.

The noise, correctness and performance analysis are similar to [CHK+18] with the differences that now SlotToCoeff and CoeffToSlot parts of the algorithm require $O(\sum_{i=0}^{s} \sqrt{N_i})$ ciphertext rotations and $O(\sum_{i=0}^{s} N_i)$ constant multiplications when performing 'baby-giant step' optimization. This is much smaller than $O(\sqrt{N})$ and $O(N)$ corresponding to HEAAN case for a full slot packing $N/2$. We now also have to store only $O(\sum_{i=0}^{s} \sqrt{N_i})$ rotations keys instead of $O(\sqrt{N})$ keys for HEAAN case. The only drawback is that when applying consecutively linear transformations, we use more rescaling operations. For small s such as $s = 1$, however, it is not a big issue.

3.3 Homomorphic Evaluations of Matrix Operations

One of the purposes to design MHEAAN is to run the matrix operations naturally. Since a matrix multiplication consists of multiplications and additions for each components, every HE scheme should support the operation. However, the there is no known general practical result yet. With the structure of MHEAAN we provide algorithms for homomorphic evaluation of approximate matrix multiplication, transposition and inverse functions.

Let n be a divisor of both of $N_0/2$ and N_1, in particular n is a power-of-two. For simplicity we will consider only square power-of-two size matrix case for multiplication, transposition and inverse. One can keep in mind parameters $(s, M_0, M_1) = (1, 2^k, 257)$, in which case n can be up to $min(2^{k-2}, 256)$, and parameters $(s, M_0, M_1) = (1, 2^k, 17)$, in which case n can be up to $min(2^{k-2}, 16)$. We start with several simple auxiliary algorithms.

Remark 3.3.1. *Multiplication and transposition algorithms can be extended to a non-square matrices case. Also for bigger matrices we can split them into smaller ones and use divide-and-conquer algorithm. We will omit the details as we need to consider many cases, although they are essentially similar.*

Row and Column Sums Let $\mathtt{ct}_{\mathbf{A}}$ - encryption of matrix $\mathbf{A} \in \mathbb{C}^{n \times n}$. Then the algorithm 2 return the ciphertext encrypting row sums of $\mathbf{A}$. Similarly we can define algorithm `ColSum` for column sums of $\mathbf{A}$.

Algorithm 2 Row Sum

```
1: procedure MHEAAN.RowSum($\mathtt{ct}_{\mathbf{A}} \in \mathcal{R}'^2_\ell$)
2:     $\mathtt{ct}_{\mathbf{S}} \leftarrow \mathtt{ct}_{\mathbf{A}}$
3:     for $j = 0$ to $\log n$ do
4:         $\mathtt{ct}_j \leftarrow$ MHEAAN.Rotate($\mathtt{ct}_{\mathbf{S}}, 2^j, 0$)
5:         $\mathtt{ct}_{\mathbf{S}} \leftarrow$ MHEAAN.Add($\mathtt{ct}_{\mathbf{S}}, \mathtt{ct}_j$)
6:     end for
7:     return $\mathtt{ct}_{\mathbf{S}}$
8: end procedure
```

Diagonal Extraction Let $\mathbf{I} \in \mathbb{C}^{n \times n}$ be the identity matrix with $\mathbf{I}_k = \mathsf{rt}(\mathbf{I}, (k, 0))$. We can obtain encryption of shifted diagonal of $\mathbf{A}$ by multiplying $\mathtt{ct}_{\mathbf{A}}$ with $\mathbf{I}_k$. The procedure is described in Algorithm 3.

Algorithm 3 Diagonal Extraction

```
1: procedure MHEAAN.Diag($\mathtt{ct}_{\mathbf{A}} \in \mathcal{R}'^2_\ell, k, p$)
2:     $\mathtt{ct}_{\mathbf{A}_k} \leftarrow$ MHEAAN.CMult($\mathtt{ct}_{\mathbf{A}}, \mathbf{I}_k$)
3:     $\mathtt{ct}_{\mathbf{A}_k} \leftarrow$ MHEAAN.ReScale($\mathtt{ct}_{\mathbf{A}_k}, p$)
4:     return $\mathtt{ct}_{\mathbf{A}_k}$
5: end procedure
```

3.3.1 Matrix by Vector Multiplication

Let ciphertext $\mathtt{ct}_{\mathbf{v}}$ encrypts vector $\mathbf{v}$ as a matrix of size $n \times 1$. Remind that $\mathtt{ct}_{\mathbf{v}}$ can be viewed as encryption of matrix of size $n \times n$, consisting of same columns $\mathbf{v}$. If we multiply $\mathtt{ct}_{\mathbf{A}^T}$ by $\mathtt{ct}_{\mathbf{v}}$ and apply `ColSum` algorithm we obtain ciphertext encrypting $\mathbf{w}^T = (\mathbf{A}\mathbf{v})^T$ as a matrix of size $1 \times n$. Matrix by vector multiplication is stated in algorithm 4. Similarly for $\mathbf{w}^T$ of size $n \times 1$ we can define `VecMatMult` algorithm that evaluates encryption of $\mathbf{A}\mathbf{w}$.

Algorithm 4 Matrix by Vector Multiplication

```
1: procedure MHEAAN.MatVecMult($\mathsf{ct}_{\mathbf{A}^T}, \mathsf{ct}_{\mathbf{v}} \in \mathcal{R}'^2_{\ell}, p \in \mathbb{Z}$)
2:     $\mathsf{ct}_{(\mathbf{Av})^T} \leftarrow$ MHEAAN.Mult($\mathsf{ct}_{\mathbf{A}^T}, \mathsf{ct}_{\mathbf{v}}$)
3:     $\mathsf{ct}_{(\mathbf{Av})^T} \leftarrow$ MHEAAN.ReScale($\mathsf{ct}_{(\mathbf{Av})^T}, p$)
4:     $\mathsf{ct}_{(\mathbf{Av})^T} \leftarrow$ MHEAAN.ColSum($\mathsf{ct}_{(\mathbf{Av})^T}$)
5:     return $\mathsf{ct}_{(\mathbf{Av})^T}$
6: end procedure
```

3.3.2 Matrix Multiplication

We adapt Fox matrix multiplication algorithm [FO87] to encrypted matrix multiplication. For $\mathsf{ct}_{\mathbf{A}}, \mathsf{ct}_{\mathbf{B}}$ be encryptions of matrices $\mathbf{A}, \mathbf{B} \in \mathbb{C}^{n \times n}$ with power-of-two n we define Algorithm 5.

Lemma 3.3.1 (Matrix Multiplication). *Let* $(\mathsf{ct}_{\mathbf{A}}, \beta_{\mathbf{A}} \cdot 2^p, 2^p, \ell)$ *and* $(\mathsf{ct}_{\mathbf{B}}, \beta_{\mathbf{B}} \cdot 2^p, 2^p, \ell)$ *be encryptions of matrices* $\mathbf{A}, \mathbf{B} \in \mathbb{C}^{n \times n}$ *respectively. The Algorithm 5 outputs* $(\mathsf{ct}_{\mathbf{C}}, \beta_{\mathbf{C}} \cdot n \cdot 2^p, n \cdot 2^p, \ell - 2p)$ *the valid encryption of* $\mathbf{C} = \mathbf{AB}$ *where* $\beta_{\mathbf{C}} = \beta_{\mathbf{A}} + \beta_{\mathbf{B}} + (\log n + 1) \cdot \beta^*$.

Remark 3.3.2. *The plain matrix multiplication algorithm has complexity* $O(n^3)$*. The Algorithm 5 requires totally* $O(n)$ *ciphertext multiplication (each of provides multiplication in parallel of* n^2 *values) and* $O(n \log n)$ *ciphertext rotations. This is almost optimal, compare to unencrypted case.*

Algorithm 5 Matrix Multiplication

```
procedure MHEAAN.MatMult($\mathsf{ct}_{\mathbf{A}}, \mathsf{ct}_{\mathbf{B}} \in \mathcal{R}'^2_{\ell}, p$)
    $\mathsf{ct}_{\mathbf{C}} \leftarrow 0$
    for $k = 0$ to $n-1$ do
        $\mathsf{ct}_{\mathbf{B}_k} \leftarrow \texttt{MHEAAN.Diag}_k(\mathsf{ct}_{\mathbf{B}}, p)$
        for $j = 1$ to $\log(n) - 1$ do
            $\mathsf{ct}_{\mathbf{B}_k} \leftarrow \texttt{MHEAAN.Add}(\mathsf{ct}_{\mathbf{B}_k}, \texttt{MHEAAN.Rotate}(\mathsf{ct}_{\mathbf{B}_k}, (0, 2^j))$
        end for
        $\mathsf{ct}_{\mathbf{A}_k} \leftarrow \texttt{MHEAAN.ModDown}(\texttt{MHEAAN.Rotate}(\mathsf{ct}_{\mathbf{A}}, (\frac{N_x}{2} - k, 0)), p)$
        $\mathsf{ct}_{\mathbf{C}_k} \leftarrow \texttt{MHEAAN.Mult}(\mathsf{ct}_{\mathbf{A}_k}, \mathsf{ct}_{\mathbf{B}_k})$
        $\mathsf{ct}_{\mathbf{C}} \leftarrow \texttt{MHEAAN.Add}(\mathsf{ct}_{\mathbf{C}}, \mathsf{ct}_{\mathbf{C}_k})$
    end for
    $\mathsf{ct}_{\mathbf{C}} \leftarrow \texttt{MHEAAN.ReScale}(\mathsf{ct}_{\mathbf{C}}, p)$
    return $\mathsf{ct}_{\mathbf{C}}$
end procedure
```

Matrix Multiplications with Permutations

We will mention about more efficient algorithm for matrix multiplication. If we consider the following permutations of matrices $\mathbf{B}'$ and $\mathbf{C}''$ of $\mathbf{B}$ and $\mathbf{C} = \mathbf{AB}$ respectively.

$$\mathbf{B}' = \begin{bmatrix} b_{0,0} & b_{1,n-1} & \cdots & b_{n-1,1} \\ b_{0,1} & b_{1,0} & \cdots & b_{n-1,2} \\ \vdots & \ddots & \vdots & \\ b_{0,n-1} & b_{1,n-2} & \cdots & b_{n-1,0} \end{bmatrix}, \mathbf{C}'' = \begin{bmatrix} c_{0,0} & c_{0,n-1} & \cdots & c_{0,1} \\ c_{1,1} & c_{1,0} & \cdots & c_{1,2} \\ \vdots & \ddots & \vdots & \\ c_{n-1,n-1} & c_{n-1,n-2} & \cdots & c_{n-1,0} \end{bmatrix}$$

Then for given encryptions of $\mathbf{A}$ and $\mathbf{B}'$, Algorithm 6 outputs encryption of $\mathbf{C}''$ - permutation of matrix $\mathbf{C}$. The Algorithm 6 requires totally $O(n)$ ciphertext multiplication (each of provides multiplication in parallel of n^2 values) and $O(n)$ ciphertext rotations. This is asymptotically optimal, compare to unencrypted case. However this algorithm is seems to be not practical for more complicated tasks as it does not preserve the matrix structure in slots.

Algorithm 6 Matrix Multiplication with Permutations

```
1: procedure MHEAAN.MatMultPermute($\mathsf{ct}_{\mathbf{A}}, \mathsf{ct}_{\mathbf{B}'} \in \mathcal{R}'^2_{\ell}, p$)
2:     $\mathsf{ct}_{\mathbf{C}''} \leftarrow 0$
3:     for $k = 0$ to $n-1$ do
4:         $\mathsf{ct}_{\mathbf{A}_k} \leftarrow$ MHEAAN.Rotate($\mathsf{ct}_{\mathbf{A}}, (k, 0)$)
5:         $\mathsf{ct}_{\mathbf{B}'_k} \leftarrow$ MHEAAN.Rotate($\mathsf{ct}_{\mathbf{B}'}, (k, k)$)
6:         $\mathsf{ct}_{\mathbf{C}''_k} \leftarrow$ MHEAAN.Mult($\mathsf{ct}_{\mathbf{A}_k}, \mathsf{ct}_{\mathbf{B}'_k}$)
7:         $\mathsf{ct}_{\mathbf{C}''} \leftarrow$ MHEAAN.Add($\mathsf{ct}_{\mathbf{C}}, \mathsf{ct}_{\mathbf{C}''_k}$)
8:     end for
9:     $\mathsf{ct}_{\mathbf{C}''} \leftarrow$ MHEAAN.ReScale($\mathsf{ct}_{\mathbf{C}''}, p$)
10:    return $\mathsf{ct}_{\mathbf{C}''}$
11: end procedure
```

3.3.3 Matrix Transposition

With `Diag` algorithm we can extract all the shifted diagonals of matrix $\mathbf{A}$. We can notice that transposed matrix $\mathbf{A}^T$ is actually consist of same shifted diagonals $\mathbf{A}_k$ of matrix $\mathbf{A}$, rotated by $(k, -k)$ slots.

Lemma 3.3.2 (Matrix Transposition). *Let* $(\mathsf{ct}_{\mathbf{A}}, \beta_{\mathbf{A}} \cdot 2^p, 2^p, \ell)$ *be an encryption of matrix* $\mathbf{A} \in \mathbb{C}^{n \times n}$. *The Algorithm 7 outputs* $(\mathsf{ct}_{\mathbf{A}^T}, \beta_{\mathbf{A}^T} \cdot 2^p, 2^p, \ell - p)$ *the valid encryption of* $\mathbf{A}^T$ *where* $\beta_{\mathbf{A}^T} = \beta_{\mathbf{A}} + \beta^*$. *So we have that the output message bound is close to* 0.

Algorithm 7 Matrix Transposition

```
1: procedure MHEAAN.MatTranspose($\mathtt{ct}_{\mathbf{A}} \in \mathcal{R}'^{2}_{\ell}, p$)
2:     $\mathtt{ct}_{\mathbf{A}^T} \leftarrow 0$
3:     for $k = 0$ to $n-1$ do
4:         $\mathtt{ct}_{\mathbf{A}_k} \leftarrow \mathtt{MHEAAN.Diag}_k(\mathtt{ct}_{\mathbf{A}}, p)$
5:         $\mathtt{ct}_{\mathbf{A}_k} \leftarrow \mathtt{MHEAAN.Rotate}(\mathtt{ct}_{\mathbf{A}_k}, (k, -k))$
6:         $\mathtt{ct}_{\mathbf{A}^T} \leftarrow \mathtt{MHEAAN.Add}(\mathtt{ct}_{\mathbf{A}^T}, \mathtt{ct}_{\mathbf{A}_k})$
7:     end for
8:     $\mathtt{ct}_{\mathbf{A}_k} \leftarrow \mathtt{MHEAAN.ReScale}(\mathtt{ct}_{\mathbf{A}_k}, p)$
9:     return $\mathtt{ct}_{\mathbf{A}_k}$
10: end procedure
```

3.3.4 Matrix Inverse

For matrix inverse we can adapt Schulz algorithm [Sch33] to encrypted approximate inverse circuit. However for `MHEAAN` we use a matrix version algorithm described in [cDSM15] and adopted in [CKKS17] as it more practical due to power-of-two degrees of matrix in the circuit. The algorithm is described below.

Assume that invertible square matrix $\mathbf{A}$ satisfies $\|\bar{\mathbf{A}}\| \leqslant \epsilon < 1$ for $\bar{\mathbf{A}} = \mathbf{I} - \frac{1}{2^t}\mathbf{A}$, for some $t \geqslant 0$ then we get

$$\frac{1}{2^t}\mathbf{A}(\mathbf{I} + \bar{\mathbf{A}})(\mathbf{I} + \bar{\mathbf{A}}^2)\dots(\mathbf{I} + \bar{\mathbf{A}}^{2^{r-1}}) = 1 - \bar{\mathbf{A}}^{2^r}$$

We can see that $\|\bar{\mathbf{A}}^{2^r}\| \leqslant \|\bar{\mathbf{A}}\|^{2^r} \leqslant \epsilon^{2^r}$, hence $\frac{1}{2^t}\prod_{j=0}^{r-1}(\mathbf{I}+\bar{\mathbf{A}}^{2^j}) = \mathbf{A}^{-1}(1-\bar{\mathbf{A}}^{2^r})$ is an approximate inverse of $\mathbf{A}$ for $\epsilon^{2^r} \ll 1$. We will slightly strengthen the condition on ϵ in the following lemma:

Lemma 3.3.3 (Matrix Inverse). *Let $(\mathtt{ct}_{\bar{\mathbf{A}}}, \beta \cdot \epsilon 2^p/n, \epsilon 2^p/n, \ell)$ be an encryption of matrix $\bar{\mathbf{A}} \in \mathbb{C}^{n\times n}$, and $\|\bar{\mathbf{A}}\| = \|\mathbf{I} - \frac{1}{2^t}\mathbf{A}\| \leqslant \epsilon < \frac{n-1}{n}$ for some t. The Algorithm 8 outputs $(\mathtt{ct}_{\mathbf{V}_r}, \beta_{\mathbf{V}_r} \cdot n^{1/n}2^{p-t}, n^{1/n}2^{p-t}, \ell - 2pr - t)$ the valid*

Algorithm 8 Matrix Inverse

```
1: procedure MHEAAN.MatInv($\mathsf{ct}_{\bar{\mathbf{A}}} \in \mathcal{R}'^2_{\ell}, r, p \in \mathbb{Z}$)
2:     $i = \texttt{MHEAAN.Encode}(\mathbf{I}, p)$
3:     $\mathsf{ct}_{\mathbf{A}_0} \leftarrow \mathsf{ct}_{\bar{\mathbf{A}}}$
4:     $\mathsf{ct}_{\mathbf{V}_0} \leftarrow \texttt{MHEAAN.ModDown}(i + \mathsf{ct}_{\bar{\mathbf{A}}}, p)$
5:     for $j = 0$ to $r - 1$ do
6:         $\mathsf{ct}_{\mathbf{A}_j} \leftarrow \texttt{MHEAAN.ReScale}(\texttt{MHEAAN.MatMult}(\mathsf{ct}_{\mathbf{A}_{j-1}}, \mathsf{ct}_{\mathbf{A}_{j-1}}), p)$
7:         $\mathsf{ct}_{\mathbf{V}_{j+1}} \leftarrow \texttt{MHEAAN.ReScale}(\texttt{MHEAAN.MatMult}(\mathsf{ct}_{\mathbf{V}_j}, i + \mathsf{ct}_{\mathbf{A}_j}), p)$
8:     end for
9:     $\mathsf{ct}_{\mathbf{V}_r} \leftarrow \texttt{MHEAAN.ReScale}(\mathsf{ct}_{\mathbf{V}_r}, t)$
10:    return $\mathsf{ct}_{\mathbf{V}_r}$
11: end procedure
```

encryption of $\mathbf{A}^{-1}$ *where* $\beta_{\mathbf{V}_r} = 2\beta + (r+1) \cdot (1 + \log n) \cdot \beta^*$*. So we have that the output message bound is close to* 2^{p-t} *and error growth linearly in* r*.*

Chapter 4

Applications

4.1 Sigmoid & Tanh Approximations

One limitation of the existing HE cryptosystems is that they only support polynomial arithmetic operations. However many machine learning algorithms require evaluation of the sigmoid or tanh functions, which become an obstacle for the implementation since they cannot be expressed as a polynomials.

Kim et al. [KSW$^+$] used the least squares approach to find a global polynomial approximation of the sigmoid function. We adapt this approximation method and consider the degree 3, 5, and 7 least squares polynomials of the sigmoid and tanh functions over the domain $[-8, 8]$.

Let a least squares polynomial of $\sigma(x)$ and $\tanh(x)$ will be denoted by $g_k(x)$ and $t_k(x)$ for $k = 3, 5, 7$. The approximate polynomials $g_k(x)$ and

$t_k(x)$ of degree 3, 5, and 7 are computed as follows:

$$\begin{cases} g_3(x) = & 0.5 - 1.20096 \cdot (x/8) + 0.81562 \cdot (x/8)^3, \\ g_5(x) = & 0.5 - 1.53048 \cdot (x/8) + 2.3533056 \cdot (x/8)^3 - 1.3511295 \cdot (x/8)^5, \\ g_7(x) = & 0.5 - 1.73496 \cdot (x/8) + 4.19407 \cdot (x/8)^3 - \\ & -5.43402 \cdot (x/8)^5 + 2.50739 \cdot (x/8)^7. \end{cases}$$

$$\begin{cases} t_3(x) = & 0.5 - 1.20096 \cdot (x/8) + 0.81562 \cdot (x/8)^3, \\ t_5(x) = & 0.5 - 1.53048 \cdot (x/8) + 2.3533056 \cdot (x/8)^3 - 1.3511295 \cdot (x/8)^5, \\ t_7(x) = & 0.5 - 1.73496 \cdot (x/8) + 4.19407 \cdot (x/8)^3 - \\ & -5.43402 \cdot (x/8)^5 + 2.50739 \cdot (x/8)^7. \end{cases}$$

A low-degree polynomial requires a smaller evaluation depth while a high-degree polynomial has a better precision. The maximum errors between $\sigma(-x)$ and the least squares $g_3(x)$, $g_5(x)$, and $g_7(x)$ are approximately 0.114, 0.061 and 0.032, respectively, and the maximum errors between $\tanh(x)$ and the least squares $t_3(x)$, $t_5(x)$, and $t_7(x)$ are approximately 0.114, 0.061 and 0.032, respectively

4.2 Homomorphic LR Training Phase

4.2.1 Database Encoding

For an efficient computation, it is crucial to find a good encoding method for the given database. The `MHEAAN` scheme supports the encryption of a plaintext matrix and the slot-wise operations over encryption. Our learning data is represented by a matrix $(z_{ij})_{1 \leqslant i \leqslant n, 0 \leqslant j \leqslant f}$. A recent work [?] used the column-wise approach, i.e., a vector of specific feature data $(z_{ij})_{1 \leqslant i \leqslant n}$ is encrypted in a single ciphertext. Consequently, this method required $(f+1)$ number of ciphertexts to encrypt the whole dataset. Another work [KSK+] used a more efficient encoding method to encrypt a *matrix* in a single ciphertext. A training dataset consists of n samples $\mathbf{z}_i \in \mathbb{R}^{f+1}$ for $1 \leqslant i \leqslant n$, which can be represented as a matrix Z as follows:

$$Z = \begin{bmatrix} z_{10} & z_{11} & \cdots & z_{1f} \\ z_{20} & z_{21} & \cdots & z_{1f} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n0} & z_{n1} & \cdots & z_{nf} \end{bmatrix}.$$

For simplicity, the authors assumed that n and $(f+1)$ are power-of-two integers satisfying $\log n + \log(f+1) \leqslant \log(N/2)$, and they packed the whole matrix in a single ciphertext in a row-by-row manner. It is necessary to perform shifting operations of row and column vectors for the evaluation of the GD algorithm, and the authors used circumvent algorithm to do row shifting.

In our approach we pack the whole matrix in a natural way, making it more easy to perform row and column rotations.

4.2.2 Homomorphic Evaluation of the GD

This section explains how to securely train the logistic regression model using the `MHEAAN` scheme. To be precise, we explicitly describe a full pipeline of the evaluation of the GD algorithm. We adapt the same assumptions as in the previous section so that the whole database can be encrypted in a single ciphertext. The generalization to arbitrary number of features and samples can be done in a straightforward way using divide-and-conquer algorithm.

First of all, a client encrypts the dataset and the initial (random) weight vector $\beta^{(0)}$ and sends them to the public cloud. The dataset matrix Z of size $n \times (f+1)$ is encrypted to a $\mathtt{ct}_Z$, and the transposed weight vector is encrypted in $\mathtt{ct}^{(0)}_{\beta^T}$. The plaintext matrices of the resulting ciphertexts are described as follows:

$$\mathtt{ct}_Z = \mathtt{Enc}\begin{bmatrix} z_{10} & \cdots & z_{1f} \\ \vdots & \ddots & \vdots \\ z_{n0} & \cdots & z_{nf} \end{bmatrix}, \mathtt{ct}^{(0)}_{\beta^T} = \mathtt{Enc}\begin{bmatrix} \beta^{(0)}_0 & \cdots & \beta^{(0)}_f \end{bmatrix}.$$

As mentioned before, both Z and $\beta^{(0)}$ are scaled by a factor of 2^p before encryption to maintain the precision of plaintexts. We skip to mention the scaling factor in the rest of this section since every step will return a ciphertext with the scaling factor of 2^p.

The public server takes two ciphertexts $\mathtt{ct}_Z$ and $\mathtt{ct}^{(t)}_{\beta^T}$ and evaluates the GD algorithm to find an optimal modeling vector. The goal of each iteration is to update the modeling vector $\beta^{(t)}$ using the gradient of loss function:

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \frac{\alpha_t}{n}\sum_{i=1}^{n} \sigma\left(-\mathbf{z}_i^T \beta^{(t)}\right) \cdot \mathbf{z}_i$$

where α_t denotes the learning rate at the t-th iteration. Each iteration consists of the following eight steps.

Step 1: For given two ciphertexts $\mathtt{ct}_Z$ and $\mathtt{ct}^{(t)}_{\beta^T}$, compute their vector by matrix multiplication `MHEAAN.VecMatMult`. The output ciphertext $\mathtt{ct}_{Z\beta^T}$ encrypts $\mathbf{z}_i^T \beta^{(t)}$ as column:

$$\mathtt{ct}_{Z\beta^T} = \mathtt{Enc}\begin{bmatrix} \mathbf{z}_1^T\beta^{(t)} \\ \mathbf{z}_2^T\beta^{(t)} \\ \vdots \\ \mathbf{z}_n^T\beta^{(t)} \end{bmatrix}.$$

Step 2: This step simply evaluates an approximating polynomial of the sigmoid function, i.e., $\mathtt{ct}_\sigma \leftarrow g(\mathtt{ct}_{Z\beta^T})$ for some $g \in \{g_3, g_5, g_7\}$. The output ciphertext encrypts the values of $g(\mathbf{z}_i^T\beta^{(t)})$ in its plaintext slots:

$$\mathtt{ct}_\sigma = \mathtt{Enc}\begin{bmatrix} g(\mathbf{z}_1^T\beta^{(t)}) \\ g(\mathbf{z}_2^T\beta^{(t)}) \\ \vdots \\ g(\mathbf{z}_n^T\beta^{(t)}) \end{bmatrix}.$$

Step 3: The public cloud multiplies the ciphertext $\mathtt{ct}_\sigma$ with the encrypted dataset $\mathtt{ct}_Z$ multiplication and rescales the resulting ciphertext by p bits:

$$\mathtt{ct}_{\sigma Z} \leftarrow \mathtt{ReScale}(\mathtt{Mult}(\mathtt{ct}_\sigma, \mathtt{ct}_Z); p).$$

The output ciphertext encrypts the n vectors $g(\mathbf{z}_i^T\beta^{(t)}) \cdot \mathbf{z}_i$ in each row:

$$\mathtt{ct}_{\sigma Z} = \mathtt{Enc}\begin{bmatrix} g(\mathbf{z}_1^T\beta^{(t)}) \cdot z_{10} & \cdots & g(\mathbf{z}_1^T\beta^{(t)}) \cdot z_{1f} \\ g(\mathbf{z}_2^T\beta^{(t)}) \cdot z_{20} & \cdots & g(\mathbf{z}_2^T\beta^{(t)}) \cdot z_{2f} \\ \vdots & \ddots & \vdots \\ g(\mathbf{z}_n^T\beta^{(t)}) \cdot z_{n0} & \cdots & g(\mathbf{z}_n^T\beta^{(t)}) \cdot z_{nf} \end{bmatrix}.$$

Step 4: This step aggregates the vectors $g(\mathbf{z}_i^T\beta^{(t)})$ to compute the gradient of the loss function. It is obtained by applying ColSum operation to $\mathtt{ct}_{\sigma Z}$:

$$\mathtt{ct}_{\Sigma} \leftarrow \mathtt{ColSum}(\mathtt{ct}_{\sigma Z})$$

The output ciphertext is

$$\mathtt{ct}_{\Sigma} = \mathtt{Enc}\left[\sum_i g(\mathbf{z}_i^T \beta^{(t)}) \cdot z_{i0} \cdots \sum_i g(\mathbf{z}_i^T \beta^{(t)}) \cdot z_{if}\right],$$

as desired.

Step 5: For the learning rate α_t, it uses the parameter p to compute the scaled learning rate $\Delta^{(t)} = \lfloor 2^p \cdot \alpha_t \rceil$. The public cloud updates $\beta^{(t)}$ using the ciphertext $\mathtt{ct}_{\Sigma}$ and the constant $\Delta^{(t)}$:

$$\mathtt{ct}_{\Delta} \leftarrow \mathtt{ReScale}(\Delta^{(t)} \cdot \mathtt{ct}_{\Sigma}; p),$$
$$\mathtt{ct}_{\beta^T}^{(t+1)} \leftarrow \mathtt{Add}(\mathtt{ct}_{\beta^T}^{(t)}, \mathtt{ct}_{\Delta}).$$

Finally it returns a ciphertext encrypting the updated modeling vector

$$\mathtt{ct}_{\beta^T}^{(t+1)} = \mathtt{Enc}\left[\beta_0^{(t+1)}\ \beta_1^{(t+1)} \cdots \beta_f^{(t+1)}\right].$$

where $\beta_j^{(t+1)} = \beta_j^{(t)} + \frac{\alpha_t}{n} \sum_i g(\mathbf{z}_i^T \beta^{(t)}) \cdot z_{ij}$.

We have to note here that original algorithm with HEAAN required much more steps due to impossibility to perform VecMatMult operation directly [KSK+].

4.2.3 Homomorphic Evaluation of NLGD

The performance of leveled HE schemes highly depends on the depth of a circuit to be evaluated. The bottleneck of homomorphic evaluation of the

GD algorithm is that we need to repeat the update of weight vector $\beta^{(t)}$ iteratively. Consequently, the total depth grows linearly on the number of iterations and it should be minimized for practical implementation.

For the homomorphic evaluation of Nesterov's accelerated gradient, a clients sends one more ciphertext $\mathtt{ct}_{v^T}^{(0)}$ encrypting the initial vector $\mathbf{v}^{(0)}$ to the public cloud. Then the server uses an encryption $\mathtt{ct}_Z$ of dataset Z to update two ciphertexts $\mathtt{ct}_{v^T}^{(t)}$ and $\mathtt{ct}_{\beta^T}^{(t)}$ at each iteration. One can securely compute $\beta^{(t+1)}$ in the same way as the previous section. Nesterov's accelerated gradient requires one more step to compute the second equation of (2.2.1) and obtain an encryption of $\mathbf{v}^{(t+1)}$ from $\mathtt{ct}_{\beta^T}^{(t)}$ and $\mathtt{ct}_{\beta^T}^{(t+1)}$.

Step 5: Let $\Delta_1^{(t)} = \lfloor 2^p \cdot \gamma_t \rceil$ and let $\Delta_2^{(t)} = 2^p - \Delta_1^{(t)}$. It obtains the ciphertext $\mathtt{ct}_{v^T}^{(t+1)}$ by computing

$$\begin{aligned}
\mathtt{ct}_{v^T}^{(t+1)} &\leftarrow \mathtt{Add}(\Delta_2^{(t)} \cdot \mathtt{ct}_{\beta^T}^{(t+1)}, \Delta_1^{(t)} \cdot \mathtt{ct}_{\beta^T}^{(t)}), \\
\mathtt{ct}_{v^T}^{(t+1)} &\leftarrow \mathtt{ReScale}(\mathtt{ct}_{v^T}^{(t+1)}; p).
\end{aligned}$$

Then the output ciphertext is

$$\mathtt{ct}_{v^T}^{(t+1)} = \mathtt{Enc}\left[v_0^{(t+1)}\ v_1^{(t+1)} \cdots v_f^{(t+1)} \right],$$

which encrypts $v_j^{(t+1)} = (1-\gamma_t) \cdot \beta_j^{(t+1)} + \gamma_t \cdot \beta_j^{(t)}$ in the plaintext slots.

4.3 Homomorphic DNN Classification

In this section we propose a homomorphic DNN classification algorithm classification algorithm that was explained in the Section 2.2.2. We will first describe how the one FC layer in DNN is implementated

For the linear transformation part we use Algorithms 9 and 10. For

Algorithm 9 Linear Transformation Column to Row

```
procedure MHEAAN.LTCR($\mathsf{ct}_{\mathbf{a}}, \mathsf{ct}_{\mathbf{W}^T}, \mathsf{ct}_{\mathbf{b}^T} \in \mathcal{R}'^2_\ell, p \in \mathbb{Z}$)
    $\mathsf{ct}_{(\mathbf{Wa})^T} \leftarrow$ MHEAAN.VecMatMult($\mathsf{ct}_{\mathbf{a}}, \mathsf{ct}_{\mathbf{W}^T}, p$)
    $\mathsf{ct}_{\mathbf{z}^T} \leftarrow$ MHEAAN.Add($\mathsf{ct}_{(\mathbf{Wa})^T}, \mathsf{ct}_{\mathbf{b}^T}$)
    return $\mathsf{ct}_{\mathbf{z}}$
end procedure
```

Algorithm 10 Linear Transformation Row to Column

```
procedure MHEAAN.LTRC($\mathsf{ct}_{\mathbf{a}^T}, \mathsf{ct}_{\mathbf{W}}, \mathsf{ct}_{\mathbf{b}} \in \mathcal{R}'^2_\ell, p \in \mathbb{Z}$)
    $\mathsf{ct}_{\mathbf{Wa}} \leftarrow$ MHEAAN.MatVecMult($\mathsf{ct}_{\mathbf{a}^T}, \mathsf{ct}_{\mathbf{W}}, p$)
    $\mathsf{ct}_{\mathbf{z}} \leftarrow$ MHEAAN.Add($\mathsf{ct}_{\mathbf{Wa}}, \mathsf{ct}_{\mathbf{b}}$)
    return $\mathsf{ct}_{\mathbf{z}}$
end procedure
```

simplicity we assume that weight matrices as well as input vectors can be encrypted in a single ciphertext. For general case we use divide-and-conquer straightforward algorithm. Consider the encryptions of weight matrix $\mathbf{W}_1 \in \mathbb{R}^{n_1 \times n_0}$, bias vector $\mathbf{b}_1 \in \mathbb{R}^{n_1}$, and input vector $\mathbf{a}_0 \in \mathbb{R}^{n_0}$.

$$\mathsf{ct}_{\mathbf{W}_1^T} = \mathtt{Enc}\begin{bmatrix} w_{11} & \cdots & w_{1n_1} \\ \vdots & \ddots & \vdots \\ w_{n_01} & \cdots & w_{n_0n_1} \end{bmatrix}, \mathsf{ct}_{\mathbf{a}_0} = \mathtt{Enc}\begin{bmatrix} a_1 \\ \vdots \\ a_{n_0} \end{bmatrix}, \mathsf{ct}_{\mathbf{b}^T} = \mathtt{Enc}\begin{bmatrix} \mathbf{b}_0 \cdots \mathbf{b}_{n_1} \end{bmatrix},$$

For linear transformation part we apply `LTCR` algorithm to $\mathsf{ct}_{\mathbf{a}_0}$, $\mathsf{ct}_{\mathbf{W}_1^T}$, $\mathsf{ct}_{\mathbf{b}_1^T}$ and obtain

$$\mathsf{ct}_{\mathbf{z}_1^T} = \mathtt{Enc}\begin{bmatrix} z_1 \cdots z_{n_1} \end{bmatrix} = \mathtt{Enc}\begin{bmatrix} (\mathbf{W}_1\mathbf{a}_0 + \mathbf{b}_1)_1 \cdots (\mathbf{W}_1\mathbf{a}_0 + \mathbf{b}_1)_{n_1} \end{bmatrix}$$

Then we evaluate $\mathsf{ct}_{\mathbf{a}_1^T}$ using polynomial approximation $g(x)$ of sigmoid function.

$$\mathtt{ct}_{\mathbf{a}_1^T} = \mathtt{Enc}\left[a_1 \cdots a_{n_1}\right] = \mathtt{Enc}\left[g(z_1) \cdots g(z_{n_1})\right]$$

After then we apply LTRC to $\mathtt{ct}_{a_1^T}$, $\mathtt{ct}_{\mathbf{W}_2}$, $\mathtt{ct}_{\mathbf{b}_2}$ to obtain $\mathtt{ct}_{z_2}$ and etc. Finally we output $\mathtt{ct}_{\mathbf{a}_\mathcal{L}}$.

4.4 Homomorphic RNN Classification

In this section we propose a homomorphic RNN classification algorithm that was explained in the Section **??**. As we can see the complexity of RNN circuit is much more complicated that the one of DNN. In our implementation we used different techniques of MHEAAN as matrix transposition, which cannot be implemented in HEAAN in a straightforwards way. As RNN circuit consist of FC and GRU layers we first show how one GRU layer can be implemented using MHEAAN techniques. At GRU step t we have encryptions of $\mathbf{x}_t$, $\mathbf{h}_{t-1}$, corresponding weight matrices $\mathbf{W}_z^T$, $\mathbf{U}_z^T$, $\mathbf{W}_r^T$, $\mathbf{U}_r^T$, $\mathbf{W}_h^T$, $\mathbf{U}_h^T$ and a bias vectors $\mathbf{b}_{\mathbf{W}_z^T}$, $\mathbf{b}_{\mathbf{U}_z^T}$, $\mathbf{b}_{\mathbf{W}_r^T}$, $\mathbf{b}_{\mathbf{U}_r^T}$, $\mathbf{b}_{\mathbf{W}_h^T}$, $\mathbf{b}_{\mathbf{U}_h^T}$. Remind the GRU circuit for unencrypted case

$$\text{(update gate) } \mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{b}_{W_z} + \mathbf{U}_z\mathbf{x}_t + \mathbf{b}_{\mathbf{U}_z}) \tag{4.4.1}$$

$$\text{(reset gate) } \mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{b}_{\mathbf{W}_r} + \mathbf{U}_r\mathbf{x}_t + \mathbf{b}_{\mathbf{U}_r}) \tag{4.4.2}$$

$$\text{(hidden cell) } \tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h\mathbf{x}_t + \mathbf{b}_{\mathbf{W}_h} + \mathbf{r}_t \odot (\mathbf{U}_h\mathbf{x}_t + \mathbf{b}_{\mathbf{U}_h})) \tag{4.4.3}$$

$$\text{(output) } \mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t \tag{4.4.4}$$

For encrypted case the update gate $\mathtt{ct}_{\mathbf{z}_t^T}$, the reset gate $\mathtt{ct}_{\mathbf{r}_t^T}$ the hidden cell $\tilde{\mathbf{h}}_t^T$ could be obtained using similar ideas to DNN with LTCR and approximate evaluations g and t of sigmoid and tanh function as described in Section ??, with one more step of hadamard multiplication of $\mathtt{ct}_{\mathbf{r}_t^T}$ and $\mathtt{ct}_{(\mathbf{U}_h\mathbf{x}_t+\mathbf{b}_{\mathbf{U}_h})^T}$ using `Mult` operation. For the output gate the main problem occurs in hadamard multiplication of $\mathbf{z}_t \odot \mathbf{h}_{t-1}$, as we have encryptions of $\mathbf{z}_t^T$ and $\mathbf{h}_{t-1}$. So we first transpose $\mathtt{ct}_{\mathbf{h}_{t-1}}$ to obtain $\mathtt{ct}_{\mathbf{h}_{t-1}^T}$, and evaluate the following circuit

$$\mathtt{ct}_{\mathbf{h}_t^T} = \mathtt{Add}(\mathtt{Mult}(\mathtt{ct}_{\mathbf{z}_t^T}, \mathtt{ct}_{\mathbf{h}_{t-1}^T}, \mathtt{Mult}((1-\mathtt{ct}_{\mathbf{z}_t^T}), \mathtt{ct}_{\tilde{\mathbf{h}}_t^T})))$$

Finally we transpose back $\mathtt{ct}_{\mathbf{h}_t^T}$ and obtain $\mathtt{ct}_{\mathbf{h}_t}$. The full flow of the algorithm is shown in Algorithm ??

Algorithm 11 Gated Recurrent Unit

```
1: procedure MHEAAN.GRU($\mathtt{ct}_{\mathbf{x}_i} \in \mathcal{R}'^2_\ell, T, p \in \mathbb{Z}$ i=1 ... T)
2:     $\mathtt{ct}_{\mathbf{h}_0} \leftarrow 0$
3:     for $t = 1 \ldots \mathcal{T}$ do
4:         $\mathtt{ct}_{\mathbf{z}_t} = g(\mathtt{Add}(\mathtt{LTCR}(\mathtt{ct}_{\mathbf{x}_t}, \mathtt{ct}_{\mathbf{W}_z}, \mathtt{ct}_{\mathbf{bW}_z}), \mathtt{LTCR}(\mathtt{ct}_{\mathbf{h}_{t-1}}, \mathtt{ct}_{\mathbf{U}_z}, \mathtt{ct}_{\mathbf{b}_{\mathbf{U}_z}})))$
5:         $\mathtt{ct}_{\mathbf{r}_t} = g(\mathtt{Add}(\mathtt{LTCR}(\mathtt{ct}_{\mathbf{x}_t}, \mathtt{ct}_{\mathbf{W}_r}, \mathtt{ct}_{\mathbf{b}_{\mathbf{W}_r}}), \mathtt{LTCR}(\mathtt{ct}_{\mathbf{h}_{t-1}}, \mathtt{ct}_{\mathbf{U}_r}, \mathtt{ct}_{\mathbf{b}_{\mathbf{U}_r}})))$
6:         $\mathtt{ct}_{\tilde{\mathbf{h}}_t} = \mathtt{Mult}(\mathtt{LTCR}(\mathtt{ct}_{\mathbf{h}_{t-1}}, \mathtt{ct}_{\mathbf{U}_h}, \mathtt{ct}_{\mathbf{b}_{\mathbf{U}_h}}), \mathtt{ct}_{\mathbf{r}_t}))$
7:         $\mathtt{ct}_{\tilde{\mathbf{h}}_t} = t(\mathtt{Add}(\mathtt{LTCR}(\mathtt{ct}_{\mathbf{x}_t}, \mathtt{ct}_{\mathbf{W}_h}, \mathtt{ct}_{\mathbf{b}_{\mathbf{W}_h}}), \mathtt{ct}_{\tilde{\mathbf{h}}_t})$
8:         $\mathtt{ct}_{\mathbf{h}_{t-1}^T} = \mathtt{MatTr}(\mathtt{ct}_{\mathbf{h}_{t-1}})$
9:         $\mathtt{ct}_{\mathbf{h}_t^T} = \mathtt{Add}(\mathtt{Mult}(\mathtt{ct}_{\mathbf{z}_t}, \mathtt{ct}_{\mathbf{h}_{t-1}^T}), \mathtt{Mult}(\mathtt{Sub}(1, \mathtt{ct}_{\mathbf{z}_t}), \mathtt{ct}_{\tilde{\mathbf{h}}_t}))$
10:        $\mathtt{ct}_{\mathbf{h}_t} = \mathtt{MatTr}(\mathtt{ct}_{\mathbf{h}_t^T})$
11:        if $l_{cur} < L$ and $t < T - 1$ then
12:            $\mathtt{Bootstrap}(\mathtt{ct}_{\mathbf{h}_t}, l_{cur}, L)$
13:        end if
14:    end for
15:    return $\mathtt{ct}_{h_T}$
16: end procedure
```

The output of the GRU algorithm is $\mathtt{ct}_{\mathbf{h}_{\mathcal{T}}^T}$ and then we proceed with FC layers as was described in Section ??

Chapter 5

Implementation Results

In this chapter, we provide implementation results with concrete parameter setting. Our implementation is based on the NTL C++ library running over GMP. Every experimentation was performed on a machine with an 2.9 GHz Intel Core i5 processor, 8 GB 1867 MHz DDR3 memory, with only 4 CPUs using a parameter set with 80-bit security level.

Parameters Setting The dimensions of a cyclotomic ring $\mathcal{R}'$ are chosen following the security estimator of Albrecht et al. [APS15] for the learning with errors problem.

Table 5.1: Parameter settings for `MHEAAN`

parameter	$N = N_0 \cdot N_1$	σ	h	L_{max}
Set_1	2^{13}	6.4	64	≈ 155
Set_2	2^{14}			≈ 310
Set_3	2^{15}			≈ 620
Set_4	2^{16}			≈ 1240

We use the discrete Gaussian distribution of standard deviation σ to sample error polynomials and set the Hamming weight h in a multivariate representation of a secret key $s(\mathbf{x})$.

We skip the results of the evaluation of component wise operations such as inverse, exponent, sigmoid functions, etc. Please refer to [CKKS17] for more details on evaluating these circuits.

Bootstrapping In Table 5.2, we present the parameter setting and performance results for full slots bootstrapping. Parameters r, p, L_{in} have the same meaning as r, $\log(p)$, $\log(q)$ in [CHK$^+$18] and similarly were chosen experimentally based on the bootstrapping error. For sufficiently large number r we maintain the precision of the output plaintext. L_{in} and L_{out} corresponds to the number of modulus bits before and after bootstrapping respectively. The running times are only for ciphertext operations and exclude encryption and decryption procedures.

Table 5.2: Implementation results for bootstrapping

parameter	N_0	N_1	r	p	L_{in}	L_{max}	L_{out}	precision	time	amor
$Boot_1$	256	256	7	35	40	1240	517	16 bits	2.5min	4.58ms
$Boot_2$			8	43	50	1240	312	20 bits	2.63min	4.83ms

Evaluation of Matrix Circuits In Table 5.3, we present the parameter setting and performance results for matrix multiplication, matrix 16-th power, and inverse. L_{in} and L_{out} corresponds to the number of modulus bits before and after operations respectively. The running times are only for ciphertext operations and exclude encryption and decryption procedures.

The homomorphic evaluation of the circuit $\mathbf{M}^{16}$ can be evaluated by squaring a matrix 4 times. Computing the matrix inverse homomorphically is done by evaluating a matrix polynomial up to degree 15 as was shown in Algorithm 8.

Table 5.3: Implementation results for $n \times n$ matrices $\mathbf{M}$, $\mathbf{M}_1$, $\mathbf{M}_2$

Function	n	N_0	N_1	p	L_{in}	L_{out}	time
$\mathbf{M}^T$	16	512	16	30	65	35	0.15s
	16	64	256				0.27s
	64	128	256				1.82s
$\mathbf{M}_1\mathbf{M}_2$	16	512	16		100	40	0.51s
	16	64	256				0.98s
	64	128	256				10.72s
$\mathbf{M}^{16}$	16	1024	16		300	60	6.82s
	16	64	256				17.23s
	64	128	256				87.65s
$\mathbf{M}^{-1}$	16	1024	16		300	60	10.61s
	16	64	256				12.87s
	64	128	256				2.1min

5.1 Evaluation of NLGD Training

Parameters settings We explain how to choose the parameter sets for the homomorphic evaluation of the NLGD algorithm. We start with the parameter L_{step} - number of bits required for one iteration. The modulus of a ciphertext is reduced after the `ReScale` operations and the evaluation of an approximate polynomial $g(x)$. The `ReScale` procedures after homomorphic multiplications reduce the ciphertext modulus by p bits. We choose degree 5 sigmoid approximation $g_5(x)$. The ciphertext modulus is reduced by $(3p + 3)$ bits for the evaluation of $g_5(x)$. For the final step we consume p bits. Therefore, we obtain the following bound on the parameter L_{step}:

$$L_{step} = 5p + 3$$

We also have to keep some L_0 bits to be able to decrypt a ciphertext. So if the number of iterations in training $\mathcal{I}$ satisfies the conditon

$$L > \mathcal{I} \cdot L_{step} + L_0$$

we can evaluate all the training without bootstrapping, otherwise we use bootstrapping as soon as as our current L_{cur} is less than $L_{step} + L_0$.

Implementation results In Table 5.4 we present parameter settings, performances, and accuracy results for genomic data privacy and security protection competition 2017, the goal of Track 3. It was to devise a weight vector to predict the disease using the genotype and phenotype data. This dataset consists of 1579 samples, each of which has 102 features and a cohort information (disease vs. healthy). Since we use the ring dimension $N_0 \cdot N_1 = 2^{16}$, we can only pack up to $N_0/2 \cdot N_1 = 2^7 \times 2^8 = 2^{15}$ dataset values in a single ciphertext but we have totally $1579 \times 103 > 2^{15}$ values to be packed. We can overcome this issue by using divide-and-conqure algorithm

The smoothing parameter γ_t is chosen in accordance with [Nes83]. The choice of proper GD learning rate parameter α_t normally depends on the problem at hand. Choosing too small α_t leads to a slow convergence, and choosing too large α_t could lead to a divergence, or a fluctuation near a local optima. It is often optimized by a trial and error method, which we are not available to perform. Under these conditions harmonic progression seems to be a good candidate and we choose a learning rate $\alpha_t = \frac{10}{t+1}$ in our implementation.

In order to estimate the validity of our method, we utilized 10-fold cross-validation (CV) technique: it randomly partitions the dataset into ten folds with approximately equal sizes, and uses every subset of 9 folds for training and the rest one for testing the model. The performance of our solution including the average running time (encryption and evaluation) and the storage (encrypted dataset) are shown in Table 5.4. This table also provides the average accuracy and the AUC (Area Under the Receiver Operating Characteristic Curve) which estimate the quality of a binary classifier.

Table 5.4: Implementation results for NLGD training

parameter	p	L_{in}	L_{out}	$\#s$	$\#f$	$\mathcal{I}$	Accuracy	AUC	time
iDASH	30	1071	40	1579	103	7	69.87%	0.729	9.6min

We also compared our method with one used in [KSW$^+$].

5.2 Evaluation of DNN Classification

Parameters settings We explain how to choose the parameter sets for the homomorphic evaluation of the DNN Classification algorithm. For each linear transformation part we consume p modulus bits. The ciphertext modulus is reduced by $(3p + 3)$ bits for the evaluation of $g_5(x)$. Therefore, we obtain the following lower bound on the parameter L_{FC}:

$$L_{FC} = 4p + 3$$

Similar to NLGD algorithm if the number of layers $\mathcal{L}$ satisfies the con-

Table 5.5: Implementation results for other datasets with 5-fold CV

Dataset	$\#s$	$\#f$	Method	$\mathcal{I}$	time	Accuracy	AUC
Edinburgh	1253	9	Ours	7	3.2min	91.04%	0.958
			[KSW+]	25	114min	86.03%	0.956
			[KSW+]	20	114min	86.19%	0.954
lbw	189	9	Ours	7	3.1min	69.19%	0.689
			[KSW+]	25	99min	69.30%	0.665
			[KSW+]	20	86min	69.29%	0.678
nhanes3	15649	15	Ours	7	6.9min	79.22%	0.717
			[KSW+]	25	235min	79.23%	0.732
			[KSW+]	20	208min	79.23%	0.737
pcs	379	9	Ours	7	3.2min	68.27%	0.740
			[KSW+]	25	103min	68.85%	0.742
			[KSW+]	20	97min	69.12%	0.750
uis	575	8	Ours	7	3.2min	74.44%	0.603
			[KSW+]	25	104min	74.43%	0.585
			[KSW+]	20	96min	75.43%	0.617

diton

$$L > \mathcal{L} \cdot L_{FC} + L_0$$

we can evaluate all the DNN classification without bootstrapping, otherwise we use bootstrapping as soon as as our current L_{cur} is less than $L_{FC} + L_0$.

Implementation results In Table 5.6 we present the parameter settings, performances, and accuracy results with one, two and four hidden layers. Our DNN classification algorithm applied to MNIST dataset [LCB10] with sigmoid activation functions. Accuracy is similar to the accuracy of predictions on unencrypted data, which is about 97.9%.

Table 5.6: Implementation results for DNN classification

parameter	p	L_{in}	L_{out}	$\mathcal{L}$	$n_0, n_1, \ldots, n_{\mathcal{L}}$	Accuracy	time
DNN_1	30	193	40	2	784,1024,10	92.9%	57s
DNN_2	30	316	40	3	784,1024,256,10	94.3%	79s
DNN_3	30	562	40	5	784,1024,1024,1024,256,10	97.9%	3.6min

5.3 Evaluation of RNN Classification

Parameters settings We explain how to choose the parameter sets for the homomorphic evaluation of the RNN Classification algorithm. For each GRU step we consume p modulus bits for linear transformations parts and $(3p + 3)$ bits for each of the g_5 and t_5 evaluations. For transposition and multiplication we consume p bits. Therefore, we obtain the following lower bound on the parameter L_{GRU}:

$$L_{GRU} = 11p + 6$$

We evaluate first several GRU steps without bootstrapping, and then we use bootstrapping as soon as our current L_{cur} is less than $L_{GRU} + L_0$.

Implementation results In Table 5.6 we present the parameter settings, performances, and accuracy results for homomorphic evaluations with gated RNNs with a real-life genomic dataset. We validate our methodology through a RNN-based model that solves microRNA(miRNA) target prediction problem [LBPY16]. The miRNA is an RNA molecule that is central in protein expression, and the model consists of RNN-based autoencoders with additional stacked RNNs; hence, the model of [LBPY16] is appropriate to validate our methodology. In this experiment, we encrypted miRNA and mRNA sequences, and subsequently trained the RNN-based model with these encrypted sequences. We used the site-level miRNA–mRNA pairing information dataset and the negative training dataset from [LBPY16]. The dataset obtained target sites from miRecords database and miRNA sequences from mirBase database. From the experimental results, we verified that the GRUs evaluated with `MHEAAN` were accurate and scalable to longer sequences.

In implementation we set $p = 35$ and $L_0 = 45$ and thus $L_{GRU} + L_0 < 517$ so we have enough capacity after bootstrapping to perform one GRU iteration.

Table 5.7: Implementation results for RNN classification

parameter	p	L_{in}	L_{out}	$\mathcal{T}$	$\#x$	$\#h$	n_1	Accuracy	time
GRU_1	30	1200	40	40	16	256	2	99.9%	254min
GRU_2	30	1200	40	40	32	64	10	99.9%	243min

Chapter 6

Conclusions

In this work, we present `MHEAAN` - a variant of the `HEAAN` homomorphic encryption scheme. `MHEAAN` takes advantage of `HEAAN` by supporting standard approximate HE operations. With a multi-dimensional packing `MHEAAN` enjoys more functionality like efficient operations on matrices and practical bootstrapping even for large number of slots. As applications of `MHEAAN` we propose a non-interactive logistic regression training, deep neural network and recurrent neural network classifications algorithms.

One of the future works could be applying `MHEAAN` to classification algorithms for general Neural Network architectures. Another interesting problem is to achieve learning phase of the Neural Networks with multiple layer structure. We believe that the idea of multi-dimensional variant could have a great potential for these as well as for other applications requiring computations on matrices and tensors.

Bibliography

[AHTPW16] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. Scalable and secure logistic regression via homomorphic encryption. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 142–144. ACM, 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 *Proc. of ITCS*, pages 309–325. ACM, 2012.

[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.

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

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

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS'11, pages 97–106. IEEE Computer Society, 2011.

[BV11b] 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.

[cDSM15] Gizem S. Çetin, Yarkın Doröz, Berk Sunar, and William J. Martin. An investigation of complex operations with word-size homomorphic encryption. Cryptology ePrint Archive, Report 2015/1195, 2015. `http://eprint.iacr.org/2015/1195`.

[CGGI18] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. *IACR Cryptology ePrint Archive*, 2018:421, 2018.

[CHK$^+$18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. Cryptology ePrint Archive, Report 2018/153, 2018. `https://eprint.iacr.org/2018/153`.

[CKK$^+$17] Jung Hee Cheon, Andrey Kim, Miran Kim, Keewoo Lee, and Yongsoo Song. Implementation for idash competition 2017, 2017. `https://github.com/kimandrik/IDASH2017`.

[CKKS16] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Implementation of HEAAN, 2016. `https://github.com/kimandrik/HEAAN`.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[CKY18] Jung Hee Cheon, Andrey Kim, and Donggeon Yhee. Multi-dimensional packing for heaan forapproximate matrix arithmetics. 2018.

[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.

[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.

[DGHV10] 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.

[DHS16] Yarkın Doröz, Yin Hu, and Berk Sunar. Homomorphic AES evaluation using the modified LTV scheme. *Designs, Codes and Cryptography*, 80(2):333–358, 2016.

[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.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology–CRYPTO 2012*, pages 643–662. Springer, 2012.

[FO87] G.C. Fox and S.W. Otto. Matrix algorithms on a hypercube i: Matrix multiplication. *Parallel Computing*, 4:17–31, 1987.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `http://crypto.stanford.edu/craig`.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology–CRYPTO 2012*, pages 850–867. Springer, 2012.

[GL09] Vernon Gayle and Paul S. Lambert. Logistic regression models in sociological research. 2009.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013*, pages 75–92. Springer, 2013.

[Har01] Frank E Harrell. Ordinal logistic regression. In *Regression modeling strategies*, pages 331–343. Springer, 2001.

[HS14] Shai Halevi and Victor Shoup. Algorithms in helib. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 554–571, 2014.

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

[HS18] Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in helib. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, pages 93–120, 2018.

[HTG17] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *CoRR*, abs/1711.05189, 2017.

[JKN+19] Jaehee Jang, Andrey Kim, Byunggook Na, Lee Byunghan, Yoon Sungroh, and Cheon Junghee. Privacy-preserving inference for gated rnns withmatrix homomorphic encryptions, 2019.

[KSK+] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic regression model training based on the approximate homomorphic encryption. *https://eprint.iacr.org/2018/254*.

[KSW+] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. Privacy-preserving logistic regression based on homomorphic encryption. preprint.

[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 44th Symposium on Theory of Computing Conference, STOC 2012*, pages 1219–1234. ACM, 2012.

[LBPY16] Byunghan Lee, Junghwan Baek, Seunghyun Park, and Sungroh Yoon. deeptarget: end-to-end learning framework for microrna target prediction using deep recurrent neural networks. In *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 434–442. ACM, 2016.

[LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[LK12] Cathryn M Lewis and Jo Knight. Introduction to genetic association studies. *Cold Spring Harbor Protocols*, 2012(3):pdb–top068163, 2012.

[LL90] Edmund G Lowrie and Nancy L Lew. Death risk in hemodialysis patients: the predictive value of commonly measured variables and an evaluation of death rate differences between facilities. *American Journal of Kidney Diseases*, 15(5):458–482, 1990.

[LP13] Tancrède Lepoint and Pascal Paillier. On the minimal number of bootstrappings in homomorphic circuits. In *WAHC 2013*, Lecture Notes in Computer Science. Springer, 2013.

[MRDY] Pradeep Kumar Mishra, Deevashwer Rathee, Dung Hoang Duong, and Masaya Yasuda. Fast secure matrix multiplications over ring-based homomorphic encryption.

[Nes83] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[PUTPPG15] Alberto Pedrouzo-Ulloa, Juan Ramón Troncoso-Pastoriza, and Fernando Pérez-González. Multivariate lattices for encrypted image processing. In *IEEE ICASSP*. 2015.

[PUTPPG16] Alberto Pedrouzo-Ulloa, Juan Ramón Troncoso-Pastoriza, and Fernando Pérez-González. On ring learning with errors over the tensor product of number fields. 2016. `https://arxiv.org/abs/1607.05244`.

[RAD78] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Sch33] G. Schulz. Iterative berechnung der reziproken matrix. *Zeitschrift für angewandte Mathematik und Mechanik*, 13:57–59, 1933.

[WS13] Kikuchi H. Wu S., Teruya T. Kawamoto J. Sakuma J. Privacy-preservation for stochastic gradient descent application to secure logistic regression. *The 27th Annual Conference of the Japanese Society for Artificial Intelligence*, (1-4), 2013.

[XWBB16] Wei Xie, Yang Wang, Steven M Boker, and Donald E Brown. Privlogit: Efficient privacy-preserving logistic regression by tailoring numerical optimizers. *arXiv preprint arXiv:1611.01170*, 2016.

Appendix A

Proofs

We follow the heuristic approach in [GHS12]. Assume that a polynomial $a(\mathbf{x}) \in \mathcal{R}'$ sampled from one of above distributions, so its nonzero entries are independently and identically distributed. Let $\boldsymbol{\xi} = (\xi_{M_0}, \ldots, \xi_{M_s})$ The value $a(\boldsymbol{\xi})$ can be obtained by consecutively computing N/N_i inner products of vectors of coefficients of a corresponding to a power x_i^j for $j = 0, \ldots, N_i - 1$ by a fixed vector $(1, \xi_{M_i}, \ldots, \xi_{M_i}^{N_i})$ of Euclidean norm $\sqrt{N_i}$. Then $a(\boldsymbol{\xi})$ has variance $V = \sigma^2 \prod_{i=0}^{s} N_i = \sigma^2 N$, where σ^2 is the variance of each coefficient of a. Hence $a(\boldsymbol{\xi})$ has the variances $V_U = 2^{2\ell}N/12$, $V_G = \sigma^2 N$ and $V_Z = \rho N$, when a is sampled from $\mathcal{R}_\ell$, $\mathcal{DG}(\sigma^2)$, $\mathcal{ZO}(\rho)$ respectively. In particular, $a(\boldsymbol{\xi})$ has the variance $V_H = h$ when $a(\mathbf{x})$ is chosen from $\mathcal{HWT}(h)$. Moreover, we can assume that $a(\boldsymbol{\xi})$ is distributed similarly to a Gaussian random variable over complex plane since it is a sum of $\phi_{M_0 \cdots M_s}/2$ independent and identically distributed random complex variables. Every evaluations at roots of unity $(\boldsymbol{\xi})$ share the same variance. Hence, we will use 6σ as a high-probability bound on the canonical embedding norm of $a(\mathbf{x})$ when each coefficient has a variance σ^2. For a multiplication of two independent random variables close to Gaussian distributions with variances σ_1^2 and σ_2^2, we will use $16\sigma_1\sigma_2$ as a high-probability bound.

Proof of Proposition 2.4.1

Proof. One of such maps $\mathcal{R}' \to \mathcal{R}$ is given by

$$x_j \mapsto x^{M/M_j} \bmod \Phi_M(x) \text{ for all } j = 0, 1, \cdots, s$$

and it extends to

$$\mathcal{S}' = \mathbb{R} \bigotimes_{\mathbb{Z}} \mathcal{R}' \to \mathcal{S} = \mathbb{R} \bigotimes_{\mathbb{Z}} \mathcal{R}$$

At first we check that this map is well-defined. This means that, for all j, x_j and $x_j + \Phi_{M_j}(x_j)$ have same image in $\mathcal{S}$, or simply $\Phi_{M_j}(x^{M/M_j})$ is divisible by $\Phi_M(x)$. Since

$$\Phi_K(x) = \prod_{(k,K)=1, 1 \leqslant k \leqslant K} (x - \zeta_K^k)$$

for any positive integer K and a primitive K-th root of unity $\zeta_K = e^{2\pi i/K}$, we have the following divisibility

$$\Phi_M(x) = \prod_{(k,M)=1, 1 \leqslant k \leqslant M} (x - \zeta_M^k) \,\Big|\, \prod_{(k,M)=1, 1 \leqslant k \leqslant M} (x^{M/M_j} - \zeta_M^{kM/M_j}) = \left(\Phi_{M_j}(x^{M/M_j})\right)^{M_j}.$$

Note that $x - a$ is always a factor of $(x^* - a^*) = (x - a)(x^{*-1} + x^{*-2}a + \cdots + a^{*-1})$. The divisibility formula concludes that $\Phi_M(x)$ and $\Phi_{M_j}(x^{M/M_j})$ shares a nontrivial common factor, and the irreducibility of $\Phi_M(x)$ implies that the common factor is $\Phi_M(x)$ itself.

Secondly we check the map is surjective. In particular, x lies in the image of the map. Since $M/M_0, M/M_1, \cdots, M/M_s$ are coprime, integers

$r_0, r_1, \cdots, r_s$ can be chosen so that $r_0 M/M_0 + r_1 M/M_1 + \cdots + r_s M/M_s = 1$. In other words, $x_0^{r_0} x_1^{r_1} \cdots x_s^{r_s}$ goes to x. Thus the map, or the restricted one on $\mathcal{R}$, is surjective.

Since both sides have same dimension, here we complete the proof. □

□

Proof of Lemma 2.4.1

Proof. From the isomorphisms above, we can consider a variant of canonical embedding map to a complex tensors:

$$\tau'_{\mathbf{N}_h}(a) = (a(\xi_{M_0}^{{g_0}^{j_0}}, \dots \xi_{M_s}^{{g_s}^{j_s}})) \in \mathbb{C}^{\mathbf{N}_h}$$

where $a \in \mathcal{S}'$, ξ_{M_i} is M_i-th root of unity, $g_0 = 5$, $0 \leqslant j_0 < N_0/2$, g_i are primitive elements in $\mathbb{Z}_{M_i}^*$, $0 \leqslant j_i < N_i$ for $i = 1, \dots, s$. The map $\tau'_{\mathbf{N}_h}$ can be written as a composition of maps

$$\tau'_{\mathbf{N}_h} = \tau'^{(0)}_{N_0/2} \circ \tau'^{(1)}_{N_1} \circ \cdots \circ \tau'^{(s)}_{N_s} \tag{A.0.1}$$

where $\tau'^{(i)}$ is given by a tensor of following linear transforms

$$\Sigma'_i = \begin{bmatrix} \xi^0_{M_i,0} & \xi^1_{M_i,0} & \cdots & \xi^{N_i-1}_{M_i,0} \\ \xi^0_{M_i,1} & \xi^1_{M_i,1} & \cdots & \xi^{N_i-1}_{M_i,1} \\ \vdots & \vdots & \ddots & \vdots \\ \xi^0_{M_i,N_i-1} & \xi^1_{M_i,N_i-1} & \cdots & \xi^{N_i-1}_{M_i,N_i-1} \end{bmatrix} \tag{A.0.2}$$

and I_j the identity matrix of size N_j, where $\xi_{M_i,j} = \exp(\frac{2\pi i \cdot {g_i}^j}{M_i})$.

By using the formula of the linear transforms, we can compare norms;

$\|a\|_2^{\mathsf{can}} \leqslant (\prod_i \|\Sigma_i'\|) \|a\|_2$, $\|a\|_2 \leqslant (\prod_i \|\Sigma_i'\|^{-1}) \|a\|_2^{\mathsf{can}}$

where $\|L\|$ for a linear operator L on a complex-valued space is given by the supremum of $\|Lx\|/\|x\|$ along all x. In above, it's square is the sum of maginitude squares of all components in the matrix, or just $\mathrm{Tr}(L^*L)$.

$\Sigma_i'^{-1}$ has components

$$l_{ab} = (-1)^{N_i-a} \frac{e_{N_i-a}(\overline{\xi}_{M_i,b})}{\prod_{c\neq b}(\xi_{M_i,b} - \xi_{M_i,c})} \tag{A.0.3}$$

For the p^k-th cyclotomic polynomial

$$\Phi_{p^k}(x) = \Phi_p(x^{p^{k-1}}) = x^{p^{k-1}(p-1)} + x^{p^{k-1}(p-2)} + \cdots + x^{p^{k-1}} + 1$$

, the roots $\xi_1, \cdots, \xi_{p^{k-1}(p-1)}$, and an index $b = 1, 2, \cdots, p^{k-1}(p-1)$, we have

$$\frac{d}{dx}\Big((x^{p^{k-1}} - 1)\Phi_{p^{k-1}}(x)\Big) = p^{k-1}x^{p^{k-1}-1}\Phi_{p^{k-1}}(x) + (x^{p^{k-1}} - 1)\frac{d}{dx}\Phi_{p^{k-1}}(x)$$

$$\Phi'_{p^{k-1}}(\xi_b) = \frac{p^k \xi_b^{p^k-1}}{\xi_b^{p^{k-1}} - 1}$$

where $\overline{\xi}_b$ is a vector consisting of all roots but ξ_b of Φ_p and $e_j(\overline{x})$ is an elementary symmetric polynomial of degree j in $p-2$ variables. Note that the denominator is of form 'p-th root of unity -1', not depending on k.

For $N = \phi(p^k) = p^k - p^{k-1}$, $(-1)^{N-a}e_{N-a}(\overline{\xi}_b)$ is the degree a coefficient of

$$\prod_{c\neq b}(x - \xi_c) = \frac{\Phi_{p^k}(x)}{x - \xi_b}$$

, which is in fact $\xi_b^{N-a-[N-a]}(1+\xi_b^{p^{k-1}}+\cdots+\xi_b^{[N-a]})$ with $[N-a]$ is the largest multiple of p^{k-1} less or eqaul to $N-a$.

In other hands,

$$\Phi'_{p^{k-1}}(\xi_k)=\prod_{l\neq k}(\xi_k-\xi_l)$$

which is the denominator of the formula A.0.3.

Therefore we have

$$\|\Sigma_i'^{-1}\|=\sum_{a,b}|l_{ab}|^2=\sum_{a,b}\left|\frac{1-\xi_{M_i,b}^{N_i-a}}{p_i^k\xi_{M_i,b}^{p_i^{k-1}}}\right|^2=\frac{N_i}{p_i^{2k}}\sum_{a \mod N_i}|1-\zeta^{N_i-a}|^2$$

where ζ is any primitive p_i-th (NOT p_i^k-th) root of unity . The right-hand side is in fact

$$(\text{if } k_i>1)\ \frac{N_i^2}{p_i^{2k+1}}\sum_{i=1}^{p_i-1}(2-2\cos 2\pi i/p_i)$$

$$(\text{if } k_i=1)\ \frac{N_i}{p_i^2}\sum_{i=1}^{p_i-1}(2-2\cos 2\pi i/p_i)$$

and since

$$\begin{aligned}\frac{1}{p}\sum_{i=1}^{p-1}\cos(2\pi i/p)&=\frac{1}{p}\left(\sum_{i=1}^{(p-1)/2}\cos(2\pi i/p)+\sum_{i=(p+1)/2}^{p-1}\cos(2\pi i/p)\right)\\&\geqslant\int_{2\pi/p}^{2\pi(p+1)/2p}\cos(x)\,dx+\int_{2\pi(p-1)/2p}^{2\pi(p-1)/p}\cos(x)\,dx\\&=\int_0^{2\pi}\cos(x)\,dx-2\int_0^{2\pi/p}\cos(x)\,dx+\int_{2\pi(p-1)/2p}^{2\pi(p+1)/2p}\cos(x)\,dx\\&\geqslant-2\times 2\pi/p-2\pi/p=-6\pi/p\end{aligned}$$

for any integer p, we conclude that

$$\|\Sigma_i'^{-1}\|^2 \leqslant \frac{p_i - 1}{p_i} \times (2 + 12\pi/p_i).$$

$\|a(\mathbf{x})\|_2$ is the ℓ_2-norm of a vector whose components consist of the coefficients of $a(\mathbf{x})$. By applying canonical embedding only on x_s, we get a new vector whose components consist of the coefficients of a polynomial $a(x_0, \cdots, x_{s-1}, \xi_s)$ in s variables $x_0, \cdots, x_{s-1}$ and their conjugations. The ℓ_2 norm of the new vector is given by $\Sigma_s^{-1} \cdot$ (coefficient vector of $a(\mathbf{x})$), thus is bounded by $\|\Sigma_s^{-1}\|\|a\|_2$. By induction on s, we have the total bound of $\|a\|_\infty^{\mathsf{can}}{}_2$ as $\prod_{i=0}^{s} \|\Sigma_i'^{-1}\|$. $p_0 = 2$ in our case and it has a special bound $\|\Sigma_0'^{-1}\| = 1$ so that our bound is in fact $\prod_{i=1}^{s} \|\Sigma_i'^{-1}\|$ as desired. □ □

Proof of Lemma 3.1.1.

Proof. We choose $v \leftarrow \mathcal{ZO}(\rho)$, $e_0, e_1 \leftarrow \mathcal{DG}(\sigma)$, then set $\mathsf{ct} \leftarrow v \cdot \mathsf{pk} + (e_0, e_1 + m)$. The bound δ_{clean} of encryption noise is computed by the following inequality:

$$\begin{aligned}\|\langle \mathsf{ct}, \mathsf{sk}\rangle - m \pmod{2^L}\|_\infty^{\mathsf{can}} &= \|v \cdot e + e_1 + e_0 \cdot s\|_\infty^{\mathsf{can}} \\ &\leqslant \|v \cdot e\|_\infty^{\mathsf{can}} + \|e_1\|_\infty^{\mathsf{can}} + \|e_0 \cdot s\|_\infty^{\mathsf{can}} \\ &\leqslant 8\sqrt{2} \cdot \sigma N + 6\sigma\sqrt{N} + 16\sigma\sqrt{hN}.\end{aligned}$$

□ □

Proof of Lemma 3.1.2.

Proof. It is satisfied that $\langle \mathsf{ct}, \mathsf{sk}\rangle = m + e \pmod{2^\ell}$ for some polynomial $e \in \mathcal{S}$ such that $\|e\|_\infty^{\mathsf{can}} < \delta$. The output ciphertext $\mathsf{ct}' \leftarrow \lfloor 2^{-p} \cdot \mathsf{ct}\rceil$ satisfies

$\langle \mathsf{ct}', \mathsf{sk} \rangle = 2^{-p} \cdot (m+e) + e_{\mathsf{scale}} \pmod{2^{\ell-p}}$ for the rounding error vector $\tau = (\tau_0, \tau_1) = \mathsf{ct}' - 2^{-p} \cdot \mathsf{ct}$ and the error polynomial $e_{\mathsf{scale}} = \langle \tau, \mathsf{sk} \rangle = \tau_0 \cdot s + \tau_1$.

We may assume that each coefficient of τ_0 and τ_1 in the rounding error vector is computationally indistinguishable from the random variable in the interval $2^{-p} \cdot \mathbb{Z}_{2^p}$ with variance $\approx 1/12$. Hence, the magnitude of scale error polynomial is bounded by

$$\|e_{\mathsf{scale}}\|_\infty^{\mathsf{can}} \leqslant \|\tau_0 \cdot s\|_\infty^{\mathsf{can}} + \|\tau_1\|_\infty^{\mathsf{can}} \leqslant 6\sqrt{N/12} + 16\sqrt{hN/12}$$

as desired. □ □

Proof of Lemma 3.1.3.

Proof. Let $\mathsf{ct}_i = (a_i, b_i)$ for $i = 1, 2$. Then $\langle \mathsf{ct}_i, \mathsf{sk} \rangle = m_i + e_i \pmod{2^\ell}$ for some polynomials $e_i \in \mathcal{S}$ such that $\|e_i\|_\infty^{\mathsf{can}} \leqslant \delta_i$. Let $(d_0, d_1, d_2) = (a_1a_2, a_1b_2 + a_2b_1, b_1b_2)$. This vector can be viewed as an encryption of $m_1 \cdot m_2$ with an error $m_1 \cdot e_2 + m_2 \cdot e_1 + e_1 \cdot e_2$ with respect to the secret vector $(s^2, s, 1)$. It follows from Lemma 3.1.2 that the ciphertext $\mathsf{ct}_{\mathsf{mult}} \leftarrow (d_1, d_2) + \lfloor 2^{-L} \cdot (d_0 \cdot \mathsf{evk} \pmod{2^{\ell+L}}) \rceil$ contains an additional error $e'' = 2^{-L} \cdot d_0 e'$ and a rounding error bounded by δ_{scale}. We may assume that d_0 behaves as a uniform random variable on $\mathcal{R}_\ell$, so $2^L \|e''\|_\infty^{\mathsf{can}}$ is bounded by $16\sqrt{Nq_\ell^2/12}\sqrt{N\sigma^2} = 8N\sigma q_\ell/\sqrt{3} = \delta_{\mathsf{ks}} \cdot 2^\ell$. Therefore, $\mathsf{ct}_{\mathsf{mult}}$ is an encryption of $m_1 \cdot m_2$ with an error and the error is bounded by

$$\|m_1e_2 + m_2e_1 + e_1e_2 + e''\|_\infty^{\mathsf{can}} + \delta_{\mathsf{scale}} \leqslant \mu_1\delta_2 + \mu_2\delta_1 + \delta_1\delta_2 + 2^{-L} \cdot 2^\ell \cdot \delta_{\mathsf{ks}} + \delta_{\mathsf{scale}}$$

as desired. □ □

Proof of Lemma 3.1.4.

Proof. Let prove the lemma for conjugation, proofs of others are the same. The vector $(a', b') = (\kappa_{-\mathbf{1}}(a), \kappa_{-\mathbf{1}}(b)) \pmod{2^\ell}$ can be viewed as an encryption of $\bar{\mathbf{Z}}$ with and error $\kappa_{-\mathbf{1}}(e)$ with respect to the secret vector $(\kappa_{-\mathbf{1}}(s), 1)$. Using proof of Lemma 3.1.3 we can get that $\mathsf{ct}_{\mathsf{cj}}$ is an encryption of $\bar{\mathcal{Z}}$ with an error bounded by

$$\|\kappa_{-1,1}(e) + e''\|_\infty^{\mathsf{can}} + \delta_{\mathsf{scale}} \leqslant \delta + 2^{-L} \cdot 2^\ell \cdot \delta_{\mathsf{ks}} + \delta_{\mathsf{scale}}$$

as desired. □ □

Proof of Lemma 3.3.1.

Proof. From Lemma 3.1.4 and the following remark about the relative error we can see that bound of message increase only after summations in line 10 of Algorithm 5, so the bound M of the output is equal to $n \cdot 2^p$. Note also that these summations do not increase the bound of the relative error. The relative error increases by β^* after rotation and increases by β^* after multiplication. So the relative error of each summand in line 10 is bounded by $\beta_{\mathbf{A}} + \beta_{\mathbf{B}} + (1 + \log n)\beta^*$. □ □

Proof of Lemma 3.3.2.

Proof. The relative error increases by β^* after rotation. So the relative error of each summand $\mathsf{ct}_{\mathbf{A}_k}$ is bounded by $\beta_{\mathbf{A}} + \beta^*$. The relative error we can see that bound of message and bound of relative error does not increase during summations of $\mathsf{ct}_{\mathbf{A}_k}$. □ □

Proof of Lemma 3.3.3.

Proof. From Lemma 3.3.1 the message of $\mathsf{ct}_{\mathbf{A}_j}$ is bounded by $\epsilon^{2^j} 2^p/n$ which implies that the message of $\mathsf{ct}_{\mathbf{V}_r}$ is bounded by

$$2^{p-t} \prod_{j=0}^{r-1} (1 + \epsilon^{2^j}/n) < \frac{2^{p-t}}{(1-\epsilon)^{1/n}} < n^{1/n} 2^{p-t}$$

The relative error β_j of $\mathsf{ct}_{\mathbf{A}_j}$ is bounded by $\beta_j \leqslant 2^j(\beta + (1 + \log n)\beta^*)$, which implies that the relative error β'_j of $\mathsf{ct}_{\mathbf{A}_j} + i$ is bounded by

$$\beta'_j \leqslant \beta_j / \left(1 + \frac{n}{\epsilon^{2^j}}\right)$$

Using induction on j, we can show that a relative error β''_j of $\mathsf{ct}_{\mathbf{V}_j}$ is bounded by

$$\beta''_j \leqslant \left(\sum_{k=0}^{j-1} \frac{2^k \epsilon^{2^k}}{n + \epsilon^{2^k}}\right) \cdot (\beta + (1 + \log n) \cdot \beta^*) + (j-1) \cdot (1 + \log n) \cdot (\beta^*) \leqslant$$
$$\frac{1}{n} \sum_{k=0}^{j-1} (2^k \epsilon^{2^k}) \cdot (\beta + (1 + \log n) \cdot \beta^*) + (j-1) \cdot (1 + \log n) \cdot \beta^* \leqslant$$
$$\frac{2}{n(1-\epsilon)} \cdot (\beta + (1 + \log n) \cdot \beta^*) + (j-1) \cdot (1 + \log n) \cdot \beta^* \leqslant$$
$$2\beta + (j+1) \cdot (1 + \log n) \cdot \beta^*$$

□ □

국문초록

혜안(Homomrphic Encryption for Arithmetics of Approximate Numbers, HEAAN)은 근사 계산을 지원하는 동형 암호 스킴이다. 혜안의 벡터 패킹 기술은 데이터 분석 및 기계 학습 분야 등 근사적인 계산이 필요한 암호화 응용 프로그램에서 효율성을 입증하였다.

다변수 혜안(Multivariate HEAAN, MHEAAN)은 평문의 텐서 구조에 대한 HEAAN의 일반화이다. 본 설계는 연산 과정에서 줄어드는 유효 숫자의 길이가 연산 서킷의 두께로 제한된다는 HEAAN의 장점을 그대로 가지고, 평문 상태에서의 근사 연산과 비교하였을 때에도 유효 숫자 낭비가 1비트를 넘지 않는다. 평문 벡터의 회전 등 고차원 벡터의 다양한 구조들이 응용에 많이 쓰임에 따라, MHEAAN은 행렬 및 텐서와 관련된 응용 프로그램에서 기존 HEAAN에 비하여 보다 효율적인 결과를 낳는다.

MHEAAN의 구체적인 2 차원 구조는 행렬 연산에 대한 MHEAAN 기법의 효율성을 보여 주며, 로지스틱 회귀분석, 심 신경망 구조 및 회귀 신경망 구조와 같은 암호화 된 데이터 및 암호화 된 모델에 대한 여러 기계 학습 알고리즘에 적용될 수 있다. 또한 효율적인 재부팅 구현을 통하여, 이는 임의의 로지스틱 회귀 분석 등의 다양한 응용 분야에 쉽게 활용될 수 있다.

주요어휘: 동형암호, 정보보호,
학번: 2014-31408

감사의 글

대학원에 입학한 것이 엊그제 같은데 벌써 5년이라는 시간이 흘러 이렇게 논문을 쓰고 있다니 참 감회가 새롭습니다. 한국어 실력도 많이 부족하고 한국에 아는 사람 하나 없이 대학원에 입학해서 그런지 5년이라는 시간은 저에게 참 힘든, 그러나 즐거운 시간이었습니다. 제가 대학원 생활에 잘 적응하고 실력도 쌓을 수 있도록 도와주신 모든 분들께 감사의 인사를 전하려고 합니다.

우선 논문을 지도해 주신, 생활적으로 지도해주신 천정희 교수님께 진심으로 감사드립니다. 다음으로 논문 심사를 해주신 김명환 교수님, 서재홍 교수님, 현동훈 교수님, 신지선 교수님께 감사드립니다. 그리고 그 누구보다 무한한 사랑으로 부족한 저를 믿고 멀리서 지켜봐 주시고 항상 응원해 주신 부모님께 감사의 말을 드립니다.

연구실에서 많은 시간을 함께 했던 홍현숙, 류한솔, 김미란, 정희원, 이창민, 송용수, 이동건, 김진수, 한규형, 이지은, 정진혁, 이주희, 손용하, 김재윤, 김두형, 한민기, 김동우, 홍승완, 조원희, 이기우 많은 도움이 주셔서 감사합니다.