RESEARCH-ARTICLE

Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference

HAO CHEN, Microsoft Research, Redmond, WA, United States

WEI DAI, Microsoft Research, Redmond, WA, United States

MIRAN KIM, University of Texas Health Science Center at Houston, Houston, TX, United States

YONGSOO SONG, Microsoft Research, Redmond, WA, United States

Open Access Support provided by:

Microsoft Research

University of Texas Health Science Center at Houston

Published: 06 November 2019

Citation in BibTeX format

CCS '19: 2019 ACM SIGSAC Conference on Computer and Communications Security
November 11 - 15, 2019
London, United Kingdom

Conference Sponsors:
SIGSAC

CCS '19: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (November 2019)
https://doi.org/10.1145/3319535.3363207
ISBN: 9781450367479

Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference

Hao Chen
haoche@microsoft.com
Microsoft Research, Redmond

Wei Dai
wei.dai@microsoft.com
Microsoft Research, Redmond

Miran Kim
miran.kim@uth.tmc.edu
UT Health Science Center at Houston

Yongsoo Song
yongsoo.song@microsoft.com
Microsoft Research, Redmond

ABSTRACT

Homomorphic Encryption (HE) is a cryptosystem which supports computation on encrypted data. López-Alt et al. (STOC 2012) proposed a generalized notion of HE, called Multi-Key Homomorphic Encryption (MKHE), which is capable of performing arithmetic operations on ciphertexts encrypted under different keys.

In this paper, we present multi-key variants of two HE schemes with packed ciphertexts. We present new relinearization algorithms which are simpler and faster than previous method by Chen et al. (TCC 2017). We then generalize the bootstrapping techniques for HE to obtain multi-key fully homomorphic encryption schemes. We provide a proof-of-concept implementation of both MKHE schemes using Microsoft SEAL. For example, when the dimension of base ring is 8192, homomorphic multiplication between multi-key BFV (resp. CKKS) ciphertexts associated with four parties followed by a relinearization takes about 116 (resp. 67) milliseconds.

Our MKHE schemes have a wide range of applications in secure computation between multiple data providers. As a benchmark, we homomorphically classify an image using a pre-trained neural network model, where input data and model are encrypted under different keys. Our implementation takes about 1.8 seconds to evaluate one convolutional layer followed by two fully connected layers on an encrypted image from the MNIST dataset.

CCS CONCEPTS

• **Security and privacy** → **Cryptography**.

KEYWORDS

multi-key homomorphic encryption; packed ciphertext; ring learning with errors; neural networks

ACM Reference Format:
Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2019. Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference. In *2019 ACM SIGSAC Conference on Computer & Communications Security (CCS’19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3363207

1 INTRODUCTION

As large amount of data are being generated and used for driving novel scientific discoveries, the effective and responsible utilization of large data remains to be a big challenge. This issue might be alleviated by outsourcing to public cloud service providers with intensive computing resources. However, there still remains a problem in privacy and security of outsourcing data and analysis. In the past few years, significant progresses have been made on cryptographic techniques for *secure computation.* Among the techniques for secure computation, Multi-Party Computation (MPC) and Homomorphic Encryption (HE) have received increasing attention due to technical breakthroughs.

The history of MPC dates back three decades ago [5, 50], and since then it has been intensively studied in the theory community. In this approach, two or more parties participate in an interactive protocol to compute a function on their private inputs, where only the output of the function is revealed to the parties. Recent years witnessed a large body of works on improving the practical efficiency of MPC, and state-of-the-art protocols have achieved orders of magnitude improvements on performance (see e.g. [20, 36, 49]). However, these protocols are still inherently inefficient in terms of communication complexity: the number of bits that the parties need to exchange during the protocol is proportional to the *product* between the complexity of the function and the number of parties. Therefore, the high communication complexity remains the main bottleneck of MPC protocols.

Moreover, the aforementioned MPC protocols may not be desirable for cloud-based applications, as all the parties involved need to perform local computation proportional to the complexity of the function. However, in practical use-cases, we cannot expect the data providers to either perform large amount of work or stay online during the entire protocol execution. Another model was proposed where the data owners secret-share their data with a small number of independent servers, who perform an MPC to generate the computation result [23, 44]. These protocols have good performance and they moved the burden from the data providers to the servers, but their privacy guarantees rely on the assumption that the servers do not collude.

CCS ’19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3363207

HE refers to a cryptosystem that allows computing on encrypted data without decrypting them, thus enabling securely outsourcing computation in an untrusted cloud. There have been significant technical advances on HE after Gentry's first construction [24]. For example, one can encrypt multiple plaintext values into a single *packed* ciphertext, and use the single instruction multiple data (SIMD) techniques to perform operations on these values in parallel [26, 48]. Hence, HE schemes with packing techniques [6, 7, 16, 22] have good amortized complexity per plaintext value, and they have been applied to privacy-preserving big data analysis [11, 37, 39]. However, traditional HE schemes only allow computation on ciphertexts decryptable under the same secret key. Therefore, HE does not naturally support secure computation applications involving multiple data providers, each providing its own secret key.

López-Alt et al. [43] proposed a *Multi-Key Homomorphic Encryption* (MKHE) scheme, which is a cryptographic primitive supporting arithmetic operations on ciphertexts which are not necessarily decryptable to the same secret key. In addition to solving the aforementioned issues of HE, MKHE can be also used to design round-efficient MPC protocols with minimal communication cost [45]. In addition, an MPC protocol from MKHE satisfies the *on-the-fly* MPC [43] property, where the circuit to be evaluated can be dynamically decided after the data providers upload their encrypted data.

Despite its versatility, MKHE has been seldom used in practice. Early studies [19, 45, 46] used a multi-key variant of the GSW scheme [28]. These constructions have large ciphertexts and their performance does not scale well with the number of parties. Previous works [8, 10] proposed MKHE schemes with short ciphertexts, with the caveat that one ciphertext encrypts only a single bit. The only existing MKHE scheme with packed ciphertexts [13, 41] is a multi-key variant of the BGV scheme [7]. Note that all the above studies were purely abstract with no implementation given, and it remains an open problem whether an MKHE scheme with support for SIMD operations can be practical.

1.1 Our Contributions

We design multi-key variants of the BFV [6, 22] and CKKS [16] schemes. We propose a new method for generating a relinearization key which is simpler and faster compared to previous technique in [13]. Furthermore, we adapt the state-of-the-art bootstrapping algorithms for these schemes [9, 12, 14] to the multi-key scenario to build Multi-Key Fully Homomorphic Encryptions with packed ciphertexts. Finally, we give a proof-of-concept implementation of our multi-key schemes using Microsoft SEAL [47] and present experimental results. To the best of our knowledge, this is the first practical implementation of MKHE schemes that support packed ciphertexts.

We also present the first viable application of MKHE that securely evaluates a pre-trained convolutional neural network (CNN) model. We build an efficient protocol where a cloud server provides on-line prediction service to a data owner using a classifier from a model provider, while protecting the privacy of both data and model using MKHE. Our scheme with support for the multi-key operations makes it possible to achieve this at a low end-to-end latency, and near-optimal cost for the data and model providers, as shown in Fig. 1. The server can store numerous ciphertexts encrypted under different keys, but the computational cost of a certain task depends only on the number of parties related to the circuit. We note that our solution has the advantage over single-key HE since the ML model providers do not need to send the unencrypted model to the server.

1.2 Overview of Our Construction

Let $R = \mathbb{Z}[X]/(X^n + 1)$ be the cyclotomic ring with a power-of-two dimension n, and $s_i \in R$ be the secret of the i-th party. The starting point of the construction of a ring-based MKHE scheme is the requirement that the resulting scheme should be able to handle homomorphic computations on ciphertexts under independently generated secret keys. A ciphertext of our MKHE scheme associated to k different parties is of the form $\overline{\mathrm{ct}} = (c_0, c_1, \ldots, c_k) \in R_q^{k+1}$ for a modulus q, which is decryptable by the concatenated secret $\overline{\mathrm{sk}} = (1, s_1, \ldots, s_k)$. In other words, its phase $\mu = \langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle \pmod{q}$ is a randomized encoding of a plaintext message m corresponding to the base scheme.

Homomorphic multiplication of BFV or CKKS consists of two steps: tensor product and relinearization. The tensor product of two input ciphertexts satisfies $\langle \overline{\mathrm{ct}}_1 \otimes \overline{\mathrm{ct}}_2, \overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}} \rangle = \langle \overline{\mathrm{ct}}_1, \overline{\mathrm{sk}} \rangle \cdot \langle \overline{\mathrm{ct}}_2, \overline{\mathrm{sk}} \rangle$, so it is a valid encryption under the tensor squared secret $\overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}}$. In the relinearization step, we aim to transform the extended ciphertext $\overline{\mathrm{ct}} = \overline{\mathrm{ct}}_1 \otimes \overline{\mathrm{ct}}_2 \in R_q^{(k+1)^2}$ into a canonical ciphertext encrypting the same message under $\overline{\mathrm{sk}}$. This step can be understood as a key-switching process which requires a special encryption of $\overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}}$. We note that $\overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}}$ contains entries $s_i s_j$ which depend on secrets of two distinct parties. Hence a relinearization key corresponding to non-linear entries cannot be generated by a single party, different from the traditional HE schemes.

We propose an RLWE-based cryptosystem to achieve this functionality. It looks similar to the ring variant of GSW [21, 28] but our scheme supports some operations between ciphertexts under different keys. Let $\mathbf{g} \in \mathbb{Z}^d$ be an integral vector, called the gadget vector. This scheme assumes the Common Reference String (CRS) model so all parties share a random polynomial vector $\mathbf{a} \in R_q^d$. Each party i generates a special encryption of secret s_i by itself, which is a matrix $\mathbf{D}_i = [\mathbf{d}_{i,0} | \mathbf{d}_{i,1} | \mathbf{d}_{i,2}] \in R_q^{d \times 3}$ satisfying $\mathbf{d}_{i,0} + s_i \cdot \mathbf{d}_{i,1} \approx r_i \cdot \mathbf{g} \pmod{q}$ and $\mathbf{d}_{i,2} \approx r_i \cdot \mathbf{a} + s_i \cdot \mathbf{g} \pmod{q}$ where r_i is a small polynomial sampled from the key distribution. It is published as the evaluation key of the i-th party.

We present two relinearization methods with different advantages. For each pair $1 \le i, j \le k$, the first method combines the i-th evaluation key $\mathbf{D}_i$ with the j-th public key $\mathbf{b}_j \approx -s_j \cdot \mathbf{a} \pmod{q}$ to generate $\mathbf{K}_{i,j} \in R_q^{d \times 3}$ such that $\mathbf{K}_{i,j} \cdot (1, s_i, s_j) \approx s_i s_j \cdot \mathbf{g} \pmod{q}$. That is, $\mathbf{K}_{i,j}$ can be used to relinearize one entry $c_{i,j}$ of an extended ciphertext into a triple (c_0', c_i', c_j') such that $c_0' + c_i' s_i' + c_j' s_j \approx c_{i,j} s_i s_j \pmod{q}$. This method can be viewed as a variant of the previous GSW ciphertext extension proposed in [45]. In particular, each row of $\mathbf{K}_{i,j}$ consists of three polynomials in R_q (compared to $O(k)$ dimension of previous works [13, 41]), so that the bit size of a shared relinearization key $\{\mathbf{K}_{i,j}\}_{1 \le i,j \le k}$ is $O(dk^2 \cdot n \log q)$ and the complexity of key generation is $O(d^2 k^2)$ polynomial operations modulo q (see Section 3 for details). The relinearization algorithm

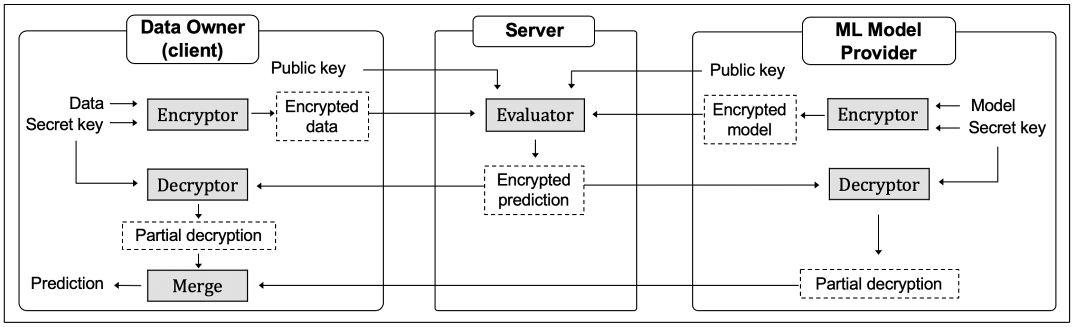

Figure 1: High-level Overview of the oblivious neural network inference.

repeats $O(k^2)$ key-switching operations from $s_i s_j$ to $(1, s_i, s_j)$, so its complexity is $O(dk^2)$ operations in R_q. We note that $\mathbf{K}_{i,j}$ can be pre-computed before multi-key operations, and a generated key can be reused for any computation related to the parties i and j.

Our second approach directly linearizes each of the entries of an extended ciphertext by multiplying the j-th public key $\mathbf{b}_j$ and i-th evaluation key $\mathbf{D}_i$ in a recursive way. The first solution should generate and store a shared relinearization key $\{\mathbf{K}_{i,j}\}_{1 \le i \le k}$, so its space and time complexity grow quadratically on k. However, the second algorithm allows us to keep only the individual evaluation keys which is linear on k. Furthermore, it significantly reduces the variance of additional noise from relinearization, so that we can use a smaller parameter while keeping the same functionality. Finally, we adapt the modulus raising technique [27, 38] to the second approach to reduce the noise growth even more.

As an orthogonal issue, the bootstrapping of packed MKHE schemes has not been studied in the literature. We generalize the existing bootstrapping methods for HE schemes [9, 12, 14, 25, 32] to the multi-key setting. The main issue of generalization is that the pipeline of bootstrapping includes some advanced functionalities such as slot permutation. We resolve this issue and provide all necessary operations by applying the multi-key-switching technique in [10] to Galois automorphism.

Finally, we apply the state-of-art optimization techniques for implementing HE schemes [4, 15, 30] to our MKHE schemes for performance improvement. For example, we implement full Residue Number System (RNS) variants of MKHE schemes and use an RNS-friendly decomposition method [4, 15, 30] for relinearization, thereby avoiding expensive high-precision arithmetic.

1.3 Related Works

López-Alt et al. [43] firstly proposed an MKHE scheme based on NTRU. After that, Clear and McGoldrick [19] suggested a multi-key variant of GSW together with ciphertext extension technique to design an MKHE scheme and it was simplified by Mukherjee and Wichs [45]. Peikert and Shiehian [46] developed two multi-hop MKHE schemes based on the same multi-key GSW scheme. However, these schemes could encrypt only a single bit in a huge extended GSW ciphertext.

Brakerski and Perlman [8] suggested an MKHE scheme with short (LWE-based) ciphertexts, however, its asymptotic/concrete efficiency has not been presented clearly. Chen, Chillotti and Song [10] proposed an improved scheme by applying the framework of TFHE [17] with the first implementation of MKHE primitive. However, this scheme does not support the packing technique, thereby resulting in a large expansion rate similar to TFHE.

Scheme	Space		Time	
	Type	Complexity	Type	Complexity
LZY+19 [41]	EvalKey	$\tilde{O}(k^3n)$	EvalKey Gen	$\tilde{O}(k^3n)$
	Ciphertext	$\tilde{O}(kn)$	Mult	$\tilde{O}(k^3n)$
CCS19 [10]	EvalKey	$\tilde{O}(k^2n^2)$	EvalKey Gen	$\tilde{O}(k^2n^2)$
	Ciphertext	$\tilde{O}(kn)$	NAND	$\tilde{O}(k^2n^2)$
This work	EvalKey	$\tilde{O}(kn)$	EvalKey Gen	–
	Ciphertext	$\tilde{O}(kn)$	Mult	$\tilde{O}(k^2n)$

Table 1: Memory (bit-size) and computational costs (number of scalar operations) of MKHE schemes. k denotes the number of parties and n is the dimension of the (R)LWE assumption. EvalKey denotes the evaluation (or bootstrapping) keys.

The most relevant studies are by Chen et al. [13] and Li et al. [41]. They designed multi-key variants of BGV [7] by generating a relinearization key based on the multi-key GSW scheme. However, it consists of $O(k^2)$ key-switching keys from $s_i s_j$ to the ordinary key each of which has $O(k)$ components. In addition, they did not provide any implementation result or analysis about concrete performance. Our work is an extension of these researches in the sense that our relinearization method and other optimization techniques can be applied to BGV as well. We also stress that the performance of previous batch MKHE schemes can be improved by observing the sparsity of evaluation keys, but this point was not pointed out in the manuscripts. In Table 1, we provide the performance of the recent MKHE schemes; in the context of our work, the second method is taken into account for comparison.

2 BACKGROUND

2.1 Notation

All logarithms are in base two unless otherwise indicated. We denote vectors in bold, e.g. $\mathbf{a}$, and matrices in upper-case bold, e.g. $\mathbf{A}$. We denote by $\langle \mathbf{u}, \mathbf{v} \rangle$ the usual dot product of two vectors $\mathbf{u}, \mathbf{v}$. For a real number r, $\lfloor r \rceil$ denotes the nearest integer to r, rounding upwards in case of a tie. We use $x \leftarrow D$ to denote the sampling x according to distribution D. For a finite set S, $U(S)$ denotes the uniform distribution on S. We let λ denote the security parameter throughout the paper: all known valid attacks against the cryptographic scheme under scope should take $\Omega(2^{\lambda})$ bit operations.

2.2 Multi-Key Homomorphic Encryption

A multi-key homomorphic encryption is a cryptosystem which allows us to evaluate an arithmetic circuit on ciphertexts, possibly encrypted under different keys.

Let $\mathcal{M}$ be the message space with arithmetic structure. An MKHE scheme $\mathtt{MKHE}$ consists of five PPT algorithms ($\mathtt{Setup}$, $\mathtt{KeyGen}$, $\mathtt{Enc}$, $\mathtt{Dec}$, $\mathtt{Eval}$). We assume that each participating party has a reference (index) to its public and secret keys. A multi-key ciphertext implicitly contains an *ordered* set $T = \{id_1, \ldots, id_k\}$ of associated references. For example, a fresh ciphertext $\mathsf{ct} \leftarrow \mathtt{MKHE.Enc}(\mu; \mathsf{pk}_{id})$ corresponds to a single-element set $T = \{id\}$ but the size of reference set gets larger as the computation between ciphertexts from different parties progresses.

- **Setup:** $pp \leftarrow \mathtt{MKHE.Setup}(1^{\lambda})$. Takes the security parameter as an input and returns the public parameterization. We assume that all the other algorithms implicitly take pp as an input .

- **Key Generation:** $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{MKHE.KeyGen}(pp)$. Outputs a pair of secret and public keys.

- **Encryption:** $\mathsf{ct} \leftarrow \mathtt{MKHE.Enc}(\mu; \mathsf{pk})$. Encrypts a plaintext $\mu \in \mathcal{M}$ and outputs a ciphertext $\mathsf{ct} \in \{0,1\}^*$.

- **Decryption:** $\mu \leftarrow \mathtt{MKHE.Dec}(\overline{\mathsf{ct}}; \{\mathsf{sk}_{id}\}_{id \in T})$. Given a ciphertext $\overline{\mathsf{ct}}$ with the corresponding sequence of secret keys, outputs a plaintext μ.

- **Homomorphic evaluation:**

$$\overline{\mathsf{ct}} \leftarrow \mathtt{MKHE.Eval}(C, (\overline{\mathsf{ct}}_1, \ldots, \overline{\mathsf{ct}}_{\ell}), \{\mathsf{pk}_{id}\}_{id \in T}).$$

Given a circuit C, a tuple of multi-key ciphertexts $(\overline{\mathsf{ct}}_1, \ldots, \overline{\mathsf{ct}}_{\ell})$ and the corresponding set of public keys $\{\mathsf{pk}_{id}\}_{id \in T}$, outputs a ciphertext $\overline{\mathsf{ct}}$. Its reference set is the union $T = T_1 \cup \cdots \cup T_{\ell}$ of reference sets T_j of the input ciphertexts $\overline{\mathsf{ct}}_j$ for $1 \le j \le \ell$.

Semantic Security. For any two messages $\mu_0, \mu_1 \in \mathcal{M}$, the distributions $\{\mathtt{MKHE.Enc}(\mu_i; \mathsf{pk})\}$ for $i = 0, 1$ should be computationally indistinguishable where $pp \leftarrow \mathtt{MKHE.Setup}(1^{\lambda})$ and $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{MKHE.KeyGen}(pp)$.

Correctness and Compactness. An MKHE scheme is compact if the size of a ciphertext relevant to k parties is bounded by $\mathrm{poly}(\lambda, k)$ for a fixed polynomial $\mathrm{poly}(\cdot, \cdot)$.

For $1 \le j \le \ell$, let $\overline{\mathsf{ct}}_j$ be a ciphertext (with reference set T_j) such that $\mathtt{MKHE.Dec}(\overline{\mathsf{ct}}_j, \{\mathsf{sk}_{id}\}_{id \in T_j}) = \mu_j$. Let $C : \mathcal{M}^{\ell} \to \mathcal{M}$ be a circuit and $\overline{\mathsf{ct}} \leftarrow \mathtt{MKHE.Eval}(C, (\overline{\mathsf{ct}}_1, \ldots, \overline{\mathsf{ct}}_{\ell}), \{\mathsf{pk}_{id}\}_{id \in T})$ for $T = T_1 \cup \cdots \cup T_{\ell}$. Then,

$$\mathtt{MKHE.Dec}(\overline{\mathsf{ct}}, \{\mathsf{sk}_{id}\}_{id \in T}) = C(\mu_1, \ldots, \mu_{\ell}) \qquad (1)$$

with an overwhelming probability. The equality of (1) can be substituted by approximate equality similar to the CKKS scheme for approximate arithmetic [16].

2.3 Ring Learning with Errors

Throughout the paper, we assume that n is a power-of-two integer and $R = \mathbb{Z}[X]/(X^n + 1)$. We write $R_q = R/(q \cdot R)$ for the residue ring of R modulo an integer q. The Ring Learning with Errors (RLWE) assumption with parameter (n, q, χ, ψ) is that given any polynomial number of samples of the form $(a_i, b_i = s \cdot a_i + e_i) \in R_q^2$, where a_i is uniformly random in R_q, s is chosen from the key distribution χ over R_q, and e_i is drawn from the error distribution ψ over R, the b_i's are computationally indistinguishable from uniformly random elements of R_q.

2.4 Gadget Decomposition

Let $\mathbf{g} = (g_i) \in \mathbb{Z}^d$ be a gadget vector and q an integer. The gadget decomposition, denoted by $\mathbf{g}^{-1}$, is a function from R_q to R^d which transforms an element $a \in R_q$ into a vector $\mathbf{u} = (u_0, \ldots, u_{d-1}) \in R^d$ of *small* polynomials such that $a = \sum_{i=0}^{d-1} g_i \cdot u_i \pmod{q}$.

The gadget decomposition technique is widely used in the construction of HE schemes. For example, homomorphic evaluation of a nonlinear circuit is based on the key-switching technique and most of HE schemes exploit various gadget decomposition method to control the noise growth. There have been suggested in the literature various decomposition methods such as bit decomposition [6, 7], base decomposition [17, 21] and RNS-based decomposition [4, 30]. Our implementation exploits an RNS-friendly decomposition for the efficiency.

3 RELINEARIZING MULTI-KEY CIPHERTEXTS

This section provides a high-level description of our MKHE schemes and explain how to perform the relinearization procedures which are core operations in homomorphic arithmetic.

3.1 Overview of HEs with Packed Ciphertexts

In recent years, there have been remarkable advances in the performance of HE schemes. For example, the ciphertext packing technique allows us to encrypt multiple data in a single ciphertext and perform parallel homomorphic operations in a SIMD manner. Currently the batch HE schemes such as BGV [7], BFV [6, 22] and CKKS [16] are the best-performing schemes in terms of amortized size and timing per plaintext slot. They adapt some DFT-like algorithms to transform a vector of plaintext values into an element of a cyclotomic ring.

Let $\mathsf{sk} = (1, s)$ for the secret $s \in R$. A canonical RLWE-based ciphertext is of the form $\mathsf{ct} = (c_0, c_1) \in R_q^2$ such that the inner product $\mu = \langle \mathsf{ct}, \mathsf{sk} \rangle \pmod{q}$, called the *phase*, is a randomized encoding of a plaintext m. For example, the phase of a BFV ciphertext has the form of $\mu = (q/t) \cdot m + e$ for the plaintext modulus t while the phase $\mu = m + e$ of CKKS is an approximate value of the plaintext.

For homomorphic computation, we basically perform arithmetic operations between the phases of given ciphertexts. In particular, homomorphic multiplication of RLWE ciphertexts consists of two steps: tensor product and relinearization. For input ciphertexts ct_1 and ct_2, we first compute their tensor product and return the extended ciphertext $\mathsf{ct} = \mathsf{ct}_1 \otimes \mathsf{ct}_2$ that satisfies $\langle \mathsf{ct}, \mathsf{sk} \otimes \mathsf{sk} \rangle = \langle \mathsf{ct}_1, \mathsf{sk} \rangle \cdot \langle \mathsf{ct}_2, \mathsf{sk} \rangle$. Since $\mathsf{sk} \otimes \mathsf{sk}$ contains the nonlinear entry s^2, it requires to perform the relinearization procedure which transforms the extended ciphertext to a canonical ciphertext encrypting the same message. Roughly speaking, we publish a relinerization key which is some kind of ciphertext encrypting s^2 under sk and run the key-switching algorithm for this conversion.

In the multi-key case, a ciphertext related to k different parties is of the form $\overline{\text{ct}} = (c_0, c_1, \ldots, c_k) \in R_q^{k+1}$ which is decryptable by the concatenated secret $\overline{\text{sk}} = (1, s_1, \ldots, s_k)$, i.e., its phase is computed by $\mu = \langle \overline{\text{ct}}, \overline{\text{sk}} \rangle = c_0 + \sum_{i=1}^{k} c_i \cdot s_i$. If we follow the same pipeline for homomorphic operation as in the single-key setting, the tensor product step returns an extended ciphertext corresponding $\overline{\text{sk}} \otimes \overline{\text{sk}}$. Hence, we need to generate a relinearization key which consists of multiple ciphertexts encrypting the entries $s_i \cdot s_j$ of $\overline{\text{sk}} \otimes \overline{\text{sk}}$. Different from the classical HE schemes, it requires some additional computations since the term $s_i \cdot s_j$ depends on two secret keys which are independently generated by different parties. In the following, we will explain how to efficiently generate a relinearization key for multi-key homomorphic multiplication.

3.2 Basic Scheme

In this section, we present a ring-based scheme which will be used to generate some public material for relinearization.

- $\underline{\texttt{Setup}(1^\lambda)}$: For a given security parameter λ, set the RLWE dimension n, ciphertext modulus q, key distribution χ and error distribution ψ over R. Generate a random vector $\mathbf{a} \leftarrow U(R_q^d)$. Return the public parameter $pp = (n, q, \chi, \psi, \mathbf{a})$.
- $\underline{\texttt{KeyGen}(pp)}$: Sample the secret key $s \leftarrow \chi$. Sample an error vector $\mathbf{e} \leftarrow \psi^d$ and set the public key as $\mathbf{b} = -s \cdot \mathbf{a} + \mathbf{e} \pmod q$ in R_q^d.
- $\underline{\texttt{UniEnc}(\mu; s)}$: For an input plaintext $\mu \in R$, generate a ciphertext $\mathbf{D} = [\mathbf{d}_0 | \mathbf{d}_1 | \mathbf{d}_2] \in R_q^{d \times 3}$ as follows:
  1. Sample $r \leftarrow \chi$.
  2. Sample $\mathbf{d}_1 \leftarrow U(R_q^d)$ and $\mathbf{e}_1 \leftarrow \psi^d$, and set $\mathbf{d}_0 = -s \cdot \mathbf{d}_1 + \mathbf{e}_1 + r \cdot \mathbf{g} \pmod q$.
  3. Sample $\mathbf{e}_2 \leftarrow \psi^d$ and set $\mathbf{d}_2 = r \cdot \mathbf{a} + \mathbf{e}_2 + \mu \cdot \mathbf{g} \pmod q$.

The public parameter pp contains a randomly generated vector $\mathbf{a} \in R_q^d$, so we are assuming the common reference string model. All parties should take the same public parameter as an input of the key-generation algorithm to support multi-key homomorphic arithmetic. We note that the same assumption was made in all the previous work on MKHE.

The uni-encryption algorithm is a symmetric encryption which can encrypt a single ring element. An uni-encrypted ciphertext $\mathbf{D} = [\mathbf{d}_0 | \mathbf{d}_1 | \mathbf{d}_2] \leftarrow \texttt{UniEnc}(\mu; s)$ consists of three vectors in R_q^d so is $(3/4)$ times as large as an ordinary RGSW ciphertext in $R_q^{2d \times 2}$. For an uni-encrypted ciphertext $\mathbf{D}$, the first two columns $[\mathbf{d}_0 | \mathbf{d}_1]$ can be viewed as an encryption of r under the secret s while $[\mathbf{d}_2 | -\mathbf{a}]$ forms an encryption of μ under secret r.

Security. The uni-encryption scheme is IND-CPA secure under the RLWE assumption of parameter (n, q, χ, ψ). We prove it by showing that the distribution

$$\{(\mathbf{a}, \mathbf{b}, \mathbf{D}) : pp = (n, \chi, \psi, \mathbf{a}) \leftarrow \texttt{Setup}(1^\lambda), \\ (s, \mathbf{b}) \leftarrow \texttt{KeyGen}(pp), \mathbf{D} \leftarrow \texttt{UniEnc}(\mu; s)\}$$

is computationally indistinguishable from the uniform distribution over $R_q^d \times R_q^d \times R_q^{d \times 3}$ for an arbitrary $\mu \in R$. We refer the reader to Appendix B for the proof of this result.

3.3 Relinearization

We revisit the relinearization procedure on extended ciphertexts and present two solutions with different advantages. We recall that the tensor product $\overline{\text{ct}} = \overline{\text{ct}}_1 \otimes \overline{\text{ct}}_2$ of two multi-key ciphertexts $\overline{\text{ct}}_i \in R_q^{k+1}$ encrypted under the concatenated secret $\overline{\text{sk}} = (1, s_1, \ldots, s_k)$ can be viewed as a ciphertext corresponding to the tensor squared secret $\overline{\text{sk}} \otimes \overline{\text{sk}}$. Note that $\overline{\text{sk}} \otimes \overline{\text{sk}}$ contains some nonlinear entries $s_i \cdot s_j$ related to two different parties. Therefore, the computing server should be able to transform the extended ciphertext $\overline{\text{ct}} \in R_q^{(k+1)\times(k+1)}$ into a canonical ciphertext by linearization of the nonlinear entries $s_i \cdot s_j$.

Our relinearization methods require the same public material (evaluation key) that is generated by individual parties as follows:

- $\underline{\texttt{EvkGen}(s)}$: Given the secret $s \in R$, return $\mathbf{D} \leftarrow \texttt{UniEnc}(s; s)$.

To be precise, each party i generates its own secret, public, and evaluation keys by running the algorithms $(s_i, \mathbf{b}_i) \leftarrow \texttt{KeyGen}(pp)$ and $\mathbf{D}_i \leftarrow \texttt{EvkGen}(s_i)$, then publishes the pair $(\mathbf{b}_i, \mathbf{D}_i)$. For the rest of this section, we present two relinearization algorithms and explain their pros and cons.

We make an additional *circular security* assumption since the evaluation key is an uni-encryption of secret s encrypted by itself. However, we stress that our assumption is no stronger than the same assumption in HE schemes [16, 17, 22, 27] requiring either bootstrapping or relinearization of ciphertexts.

3.3.1 *First Method.*

This solution includes a pre-processing step which generates a shared relinearization key corresponding to the set of involved parties. A shared relinearization key consists of encryptions of $s_i \cdot s_j$ for all pairs $1 \le i, j \le k$. Then, we can linearize an extended ciphertext by applying a standard key-switching technique.

This approach is similar to a method proposed in previous work [13, 41] which also generates a shared evaluation key. However, each element of our shared relinearization key is computed from the public information of at most two parties so consists of three vectors, while previous method based on the multi-key GSW scheme has $O(k)$ dimensional entries.

- $\underline{\texttt{Convert}(\mathbf{D}_i, \mathbf{b}_j)}$: It takes as the input a pair of an uni-encryption $\mathbf{D}_i = [\mathbf{d}_{i,0} | \mathbf{d}_{i,1} | \mathbf{d}_{i,2}] \in R_q^{d \times 3}$ and a public key $\mathbf{b}_j \in R_q^d$ generated by (possibly different) parties i and j. Let $\mathbf{k}_{i,j,0}$ and $\mathbf{k}_{i,j,1}$ be the vectors in R_q^d such that $\mathbf{k}_{i,j,0}[\ell] = \langle \mathbf{g}^{-1}(\mathbf{b}_j[\ell]), \mathbf{d}_{i,0} \rangle$ and $\mathbf{k}_{i,j,1}[\ell] = \langle \mathbf{g}^{-1}(\mathbf{b}_j[\ell]), \mathbf{d}_{i,1} \rangle$ for $1 \le \ell \le d$, i.e., $[\mathbf{k}_{i,j,0} | \mathbf{k}_{i,j,1}] = \mathbf{M}_j \cdot [\mathbf{d}_{i,0} | \mathbf{d}_{i,1}]$ where $\mathbf{M}_j \in R^{d \times d}$ is the matrix whose ℓ-th row is $\mathbf{g}^{-1}(\mathbf{b}_j[\ell]) \in R^d$. Let $\mathbf{k}_{i,j,2} = \mathbf{d}_{i,2}$ and return the ciphertext $\mathbf{K}_{i,j} = [\mathbf{k}_{i,j,0} | \mathbf{k}_{i,j,1} | \mathbf{k}_{i,j,2}] \in R_q^{d \times 3}$.

$$\left[\begin{array}{c|c} \\ \mathbf{k}_{i,j,0} & \mathbf{k}_{i,j,1} \\ \\ \end{array}\right] = \left[\begin{array}{c} \mathbf{g}^{-1}(\mathbf{b}_j[1]) \\ \hline \vdots \\ \hline \mathbf{g}^{-1}(\mathbf{b}_j[d]) \end{array}\right] \cdot \left[\begin{array}{c|c} \\ \mathbf{d}_{i,0} & \mathbf{d}_{i,1} \\ \\ \end{array}\right], \quad \left[\begin{array}{c} \\ \mathbf{k}_{i,j,2} \\ \\ \end{array}\right] = \left[\begin{array}{c} \\ \mathbf{d}_{i,2} \\ \\ \end{array}\right].$$

Algorithm 1 Relinearization method 1

```
Input: $\overline{\mathsf{ct}} = (c_{i,j})_{0 \le i,j \le k}$, $\overline{\mathsf{rlk}} = \{\mathbf{K}_{i,j}\}_{1 \le i,j \le k}$.
Output: $\overline{\mathsf{ct}}' = (c'_i)_{0 \le i \le k} \in R_q^{k+1}$.
1: $c'_0 \leftarrow c_{0,0}$
2: for $1 \le i \le k$ do
3:     $c'_i \leftarrow c_{0,i} + c_{i,0} \pmod q$
4: end for
5: for $1 \le i,j \le k$ do
6:     $(c'_0, c'_i, c'_j) \leftarrow (c'_0, c'_i, c'_j) + \mathbf{g}^{-1}(c_{i,j}) \cdot \mathbf{K}_{i,j} \pmod q$
7: end for
```

- $\underline{\mathtt{Relin}(\overline{\mathsf{ct}}; \{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k})}$: Given an extended ciphertext $\overline{\mathsf{ct}} = (c_{i,j})_{0 \le i,j \le k}$ and k pairs of evaluation/public keys $\{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k}$, generate a ciphertext $\overline{\mathsf{ct}}' \in R_q^{k+1}$ as follows:

  (1) Compute $\mathbf{K}_{i,j} \leftarrow \mathtt{Convert}(\mathbf{D}_i, \mathbf{b}_j)$ for all $1 \le i,j \le k$ and set the relinearization key as $\overline{\mathsf{rlk}} = \{\mathbf{K}_{i,j}\}_{1 \le i,j \le k}$.

  (2) Run Alg. 1 to relinearize $\overline{\mathsf{ct}}$.

We note that the first step (generation of $\overline{\mathsf{rlk}}$) can be pre-computed on public information $\{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k}$ without taking a ciphertext as the input.

Correctness. We first claim that, if $\mathbf{D}_i$ is an uni-encryption of $\mu_i \in R$ encrypted by the i-th party and $\mathbf{b}_j$ is the public key of the j-th party, then the output $\mathbf{K}_{i,j} \leftarrow \mathtt{Convert}(\mathbf{D}_i, \mathbf{b}_j)$ of the conversion algorithm is an encryption of $\mu_i s_j$ with respect to the secret $(1, s_i, s_j)$, i.e., $\mathbf{k}_{i,j,0} + s_i \cdot \mathbf{k}_{i,j,1} + s_j \cdot \mathbf{k}_{i,j,2} \approx \mu_i s_j \cdot \mathbf{g} \pmod q$. It is derived from the following formulas:

$$\mathbf{k}_{i,j,0} + s_i \cdot \mathbf{k}_{i,j,1} = \mathbf{M}_j \cdot (\mathbf{d}_0 + s_i \cdot \mathbf{d}_1) \approx \mathbf{M}_j \cdot r_i \mathbf{g} = r_i \mathbf{b}_j \pmod q,$$
$$s_j \cdot \mathbf{k}_{i,j,2} = s_j \cdot \mathbf{d}_2 \approx r_i s_j \cdot \mathbf{a} + \mu s_j \cdot \mathbf{g} \approx -r \cdot \mathbf{b}_j + \mu_i s_j \cdot \mathbf{g} \pmod q.$$

Note that $\mathbf{M}_j$, s_j and r_i should be small to hold the approximate equalities. We estimate the size of noise in Appendix C.

We now show the correctness of our algorithm. Since the evaluation key $\mathbf{D}_i$ of the i-th party is an uni-encryption of $\mu_i = s_i$, we obtain that $\mathbf{K}_{i,j} \cdot (1, s_i, s_j) \approx s_i s_j \cdot \mathbf{g} \pmod q$. From the definition of $\overline{\mathsf{ct}}'$, we get

$$\langle \overline{\mathsf{ct}}', \overline{\mathsf{sk}} \rangle = c'_0 + \sum_{i=1}^{k} c'_i \cdot s_i$$
$$= c_{0,0} + \sum_{i=1}^{k} (c_{0,i} + c_{i,0}) s_i + \sum_{i,j=1}^{k} \mathbf{g}^{-1}(c_{i,j}) \cdot \mathbf{K}_{i,j} \cdot (1, s_i, s_j) \pmod q$$
$$\approx c_{0,0} + \sum_{i=1}^{k} (c_{0,i} + c_{i,0}) s_i + \sum_{i,j=1}^{k} c_{i,j} \cdot s_i s_j = \langle \overline{\mathsf{ct}}, \overline{\mathsf{sk}} \otimes \overline{\mathsf{sk}} \rangle \pmod q.$$

3.3.2 *Second Method.*

Our second solution does not generate a shared relinearization key different from the previous one. Instead, it directly linearizes each entry $c_{i,j}$ of an extended ciphertext $\overline{\mathsf{ct}} = (c_{i,j})_{0 \le i,j \le k}$ by multiplying it to $\mathbf{b}_j$ and $\mathbf{D}_i$ in a recursive way.

- $\underline{\mathtt{Relin}(\overline{\mathsf{ct}}; \{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k})}$: For given extended ciphertext $\overline{\mathsf{ct}} = (c_{i,j})_{0 \le i,j \le k}$ and k pairs of evaluation/public keys $\{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k}$, generate a ciphertext $\overline{\mathsf{ct}}' \in R_q^{k+1}$ as described in Alg. 2.

Algorithm 2 Relinearization method 2

```
Input: $\overline{\mathsf{ct}} = (c_{i,j})_{0 \le i,j \le k}$, $\left\{(\mathbf{D}_i = [\mathbf{d}_{i,0} | \mathbf{d}_{i,1} | \mathbf{d}_{i,2}], \mathbf{b}_i)\right\}_{1 \le i \le k}$.
Output: $\overline{\mathsf{ct}}' = (c'_i)_{0 \le i \le k} \in R_q^{k+1}$.
1: $c'_0 \leftarrow c_{0,0}$
2: for $1 \le i \le k$ do
3:     $c'_i \leftarrow c_{0,i} + c_{i,0} \pmod q$
4: end for
5: for $1 \le i,j \le k$ do
6:     $c'_{i,j} \leftarrow \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{b}_j \rangle \pmod q$
7:     $(c'_0, c'_i) \leftarrow (c'_0, c'_i) + \mathbf{g}^{-1}(c'_{i,j}) \cdot [\mathbf{d}_{i,0} | \mathbf{d}_{i,1}] \pmod q$
8:     $c'_j \leftarrow c'_j + \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{d}_{i,2} \rangle \pmod q$
9: end for
```

We will analyze and compare two relinearization methods in the following section. In short, the second method has advantages in storage and noise growth while the first method could be faster if a shared evaluation key is used repeatedly to relinearize multiple ciphertexts corresponding to the same set of parties. We first show the correctness of the second method.

Correctness. At each iteration of the second for-loop in Alg. 2, we compute $c'_{i,j} = \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{b}_j \rangle$, then add $\mathbf{g}^{-1}(c'_{i,j}) \cdot [\mathbf{d}_{i,0} | \mathbf{d}_{i,1}]$ and $\langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{d}_{i,2} \rangle$ to (c'_0, c'_i) and c'_j, respectively. We note that

$$\mathbf{g}^{-1}(c'_{i,j}) \cdot [\mathbf{d}_{i,0} | \mathbf{d}_{i,1}] \cdot (1, s_i) \approx r_i \cdot c'_{i,j} \pmod q, \quad \text{and}$$
$$\langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{d}_{i,2} \rangle \cdot s_j \approx \langle \mathbf{g}^{-1}(c_{i,j}), -r_i \cdot \mathbf{b}_j + s_i s_j \cdot \mathbf{g} \rangle \pmod q$$
$$= -r_i \cdot c'_{i,j} + c_{i,j} \cdot s_i s_j \pmod q.$$

From the definition of $\overline{\mathsf{ct}}'$, we get

$$\langle \overline{\mathsf{ct}}', \overline{\mathsf{sk}} \rangle = c'_0 + \sum_{i=1}^{k} c'_i \cdot s_i$$
$$= c_{0,0} + \sum_{i=1}^{k} (c_{0,i} + c_{i,0}) s_i + \sum_{i,j=1}^{k} \mathbf{g}^{-1}(c'_{i,j}) \cdot [\mathbf{d}_{i,0} | \mathbf{d}_{i,1}] \cdot (1, s_i)$$
$$+ \sum_{i,j=1}^{k} \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{d}_{i,2} \rangle \cdot s_j \pmod q$$
$$\approx c_{0,0} + \sum_{i=1}^{k} (c_{0,i} + c_{i,0}) s_i + \sum_{i,j=1}^{k} c_{i,j} \cdot s_i s_j = \langle \overline{\mathsf{ct}}, \overline{\mathsf{sk}} \otimes \overline{\mathsf{sk}} \rangle \pmod q.$$

3.3.3 *Performance of Relinearization Algorithms.*

Suppose that there are k different parties involved in a multi-key computation. For relinearizing an extended ciphertext $\overline{\mathsf{ct}} = (c_{i,j})_{0 \le i,j \le k} \in R_q^{(k+1)^2}$, both of our relinearization methods repeat some computations on each $c_{i,j}$ to switch its corresponding secret $s_i \cdot s_j$ into $(1, s_i, s_j)$. So we will focus on a single step (i,j) of each solution to compare their performance.

In our first method, a computing party generates a shared relinearization key $\mathbf{K}_{i,j}$ and uses it to linearize an input extended ciphertext. The generation of $\mathbf{K}_{i,j}$ includes a multiplication between $d \times d$ and $d \times 2$ matrices so its complexity is $2d^2$ polynomial multiplications. However, the computation of $\mathbf{g}^{-1}(c_{i,j}) \cdot \mathbf{K}_{i,j}$ in Step 6 of Alg. 1 requires only $3d$ polynomial multiplications. Meanwhile, the second method does not have any pre-processing but a single iteration of Alg. 2 requires $4d$ polynomial multiplications. As a result, the first method can be up to (4/3) times faster when one performs multiple homomorphic arithmetic on the same set (or its subset) of parties using a pre-computed shared relinearization key, however, the required storage grows quadratically on k compared to the linear memory of the second method.

The second method also has an advantage in noise management, which we will discuss below together with modulus raising technique.

3.3.4 Special Modulus Technique. Noise growth is the main factor determining the parameter size and thereby overall performance of a cryptosystem. In general, we can use a large decomposition degree d to reduce the size of a decomposed vector $\mathbf{g}^{-1}(\cdot)$ as well as key-switching error, but this naive method causes performance degradation. In addition, the benefit of this trade-off between noise growth and computational complexity gets smaller and smaller as d increases. Therefore, this method is not the best option when we should have a small noise.

The special modulus (a.k.a. modulus raising) technique proposed in [27, 38] is one attractive solution to address this noise problem with a smaller overhead. Roughly speaking, it raises the ciphertext modulus from q to pq for an integer p called special modulus, and then computes the key-switching procedure over R_{pq} followed by modulus reduction back to q. The main advantage of this method is that a key-switching error is decreased by a factor of about p due to the modulus reduction. We apply this technique to our relinearization and encryption algorithms. In particular, a special modulus variant of relinearization requires two sequential modulus switching operations (see Appendix A for details).

We recall that for an extended ciphertext $\overline{\text{ct}} \in R_q^{(k+1)^2}$, the goal of relinearization is to generate a ciphertext $\overline{\text{ct}}' \in R_q^{k+1}$ such that $\langle \overline{\text{ct}}', \overline{\text{sk}} \rangle = \langle \overline{\text{ct}}, \overline{\text{sk}} \otimes \overline{\text{sk}} \rangle + e_{lin}$ for some error e_{lin}, which should be minimized for efficiency. We refer the reader to Appendix C which provides a noise analysis based on the variance of polynomial coefficients, but we present a concise summary in this section.

Let u be a uniform random variable over R_q. We consider its decomposition $\mathbf{g}^{-1}(u)$ and denote by V_g the average of variances of its coefficients. We respectively estimate the variance of a relinearization error from our first and second methods:

$$V_1 \approx k^2 n^2 \sigma^2 \cdot d^2 V_g^2, \quad V_2 \approx k^2 n^2 \sigma^2 \cdot d V_g.$$

In addition, the special modulus variant of the second method achieves a smaller noise whose variance is

$$V_2' = p^{-2} \cdot V_2 + \frac{1}{24}(k^2 + k)n.$$

Compared to the first method, our second solution has significant advantages in practice because we may use an efficient decomposition method with a small d while obtaining the same level of noise growth. Furthermore, its modulus raising variant obtains an even smaller error variance which is not nearly affected by the size of decomposition since V_2' is dominated by the second term (rounding error) when we introduce a special modulus p which can cancel out the term V_2.

4 TWO MKHE SCHEMES WITH PACKED CIPHERTEXTS

In this section, we present multi-key variants of the BFV [6, 22] and CKKS [16] schemes. They share the following setup and key generation phases but have different algorithms for message encoding and homomorphic operations.

- $\underline{\texttt{MKHE.Setup}(1^{\lambda})}$: Run $\texttt{Setup}(1^{\lambda})$ and return the parameter pp.

- $\underline{\texttt{MKHE.KeyGen}(pp)}$: Each party i generates secret, public and evaluation keys by $(s_i, \mathbf{b}_i) \leftarrow \texttt{KeyGen}(pp)$ and $\mathbf{D}_i \leftarrow \texttt{EvkGen}(s_i)$, respectively.

Encryption, decryption and homomorphic arithmetic of our MKHE schemes are described in the next subsections. We have a common pre-processing when performing a homomorphic operation between ciphertexts. For given ciphertexts $\overline{\text{ct}}_i \in R_q^{k_i+1}$, we denote $k \geq \max\{k_1, k_2\}$ the number of parties involved in either $\overline{\text{ct}}_1$ or $\overline{\text{ct}}_2$. We rearrange the entries of $\overline{\text{ct}}_i$ and pad zeros in the empty entries to generate some ciphertexts $\overline{\text{ct}}_i^*$ sharing the same secret $\overline{\text{sk}} = (1, s_1, \ldots, s_k)$. To be precise, a ciphertext $\overline{\text{ct}}_i = (c_0, c_1, \ldots, c_{k_i})$ corresponding to the tuple of parties $(id_1, \ldots, id_{k_i}) \in \{1, 2, \ldots, k\}^{k_i}$ is converted into the ciphertext $\overline{\text{ct}}_i^* = (c_0^*, c_1^*, \ldots, c_k^*) \in R_q^{k+1}$ which is defined as $c_0^* = c_0$ and $c_i^* = \begin{cases} c_j & \text{if } i = id_j \text{ for some } 1 \leq j \leq k_i; \\ 0 & \text{otherwise,} \end{cases}$
for $1 \leq i \leq k$. We remark that

$$\langle \overline{\text{ct}}_i, (1, s_{id_1}, \ldots, s_{id_{k_i}}) \rangle = \langle \overline{\text{ct}}^*, (1, s_1, \ldots, s_k) \rangle.$$

For simplicity, we will assume that this pre-processing is always done before homomorphic arithmetic so that two input ciphertexts are related to the same set of k parties.

Security and Correctness. We recall that BFV and CKKS are IND-CPA secure under the RLWE assumption of parameter (n, q, χ, ψ). Our MKHE schemes have exactly the same single-key encryption algorithms so that their security relies on the hardness of the same RLWE problem (see Appendix B for details). We will briefly show the correctness of our schemes in this section but the rigorous proofs with noise analysis will be provided in Appendix C.

4.1 Multi-Key BFV

The BFV scheme [6, 22] is a scale-invariant HE which supports exact computation on a discrete space with a finite characteristic. We denote by t the plaintext modulus and $\Delta = \lfloor q/t \rceil$ be the scaling factor of the BFV scheme. The native plaintext space is the set of cyclotomic polynomials R_t, but a plaintext is decoded to a tuple of finite field elements via a ring isomorphism from R_t depending on the relation of t and n [48].

- $\underline{\texttt{MK-BFV.Enc}(m; \mathbf{b}, \mathbf{a})}$: This is the standard BFV encryption which takes a polynomial $m \in R_t$ as the input. Let $a = \mathbf{a}[0]$ and $b = \mathbf{b}[0]$. Sample $v \leftarrow \chi$ and $e_0, e_1 \leftarrow \psi$. Return the ciphertext $\text{ct} = (c_0, c_1) \in R_q^2$ where $c_0 = v \cdot b + \Delta \cdot m + e_0 \pmod{q}$ and $c_1 = v \cdot a + e_1 \pmod{q}$.

- $\underline{\texttt{MK-BFV.Dec}(\overline{\text{ct}}; s_1, \ldots, s_k)}$: Let $\overline{\text{ct}} = (c_0, c_1, \ldots, c_k) \in R_q^{k+1}$ be a ciphertext associated to k parties and $s_1, \ldots, s_k$ be their secret keys. Set $\overline{\text{sk}} = (1, s_1, \ldots, s_k)$ and compute $\left\lfloor (t/q) \cdot \langle \overline{\text{ct}}, \overline{\text{sk}} \rangle \right\rceil \pmod{t}$.

- $\underline{\texttt{MK-BFV.Add}(\overline{\mathrm{ct}}_1, \overline{\mathrm{ct}}_2)}$: Given two ciphertexts $\overline{\mathrm{ct}}_i \in R_q^{k+1}$, return the ciphertext $\overline{\mathrm{ct}}' = \overline{\mathrm{ct}}_1 + \overline{\mathrm{ct}}_2 \pmod q$.
- $\underline{\texttt{MK-BFV.Mult}(\overline{\mathrm{ct}}_1, \overline{\mathrm{ct}}_2; \{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k})}$: Given two ciphertexts $\overline{\mathrm{ct}}_i \in R_q^{k+1}$, compute $\overline{\mathrm{ct}} = \lfloor (t/q) \cdot (\overline{\mathrm{ct}}_1 \otimes \overline{\mathrm{ct}}_2) \rceil \pmod q \in R_q^{(k+1)^2}$ and return the ciphertext $\overline{\mathrm{ct}}' \leftarrow \texttt{Relin}(\overline{\mathrm{ct}}; \{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k})$.

The correctness of our scheme is obtained from the properties of the basic BFV and relinearization algorithm. A multi-key BFV encryption of $m \in R_t$ is a vector $\overline{\mathrm{ct}} = (c_0, c_1, \ldots, c_k) \in R_q^{k+1}$ such that $\langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle \approx \Delta \cdot m \pmod q$ for the secret $\overline{\mathrm{sk}} = (1, s_1, \ldots, s_k)$. So the decryption algorithm can recover m correctly. If $\overline{\mathrm{ct}}_1$ and $\overline{\mathrm{ct}}_2$ are encryptions of m_1 and m_2 with respect to the secret $\overline{\mathrm{sk}} = (1, s_1, \ldots, s_k)$, then their (scaled) tensor product $\overline{\mathrm{ct}} = \lfloor (t/q) \cdot (\overline{\mathrm{ct}}_1 \otimes \overline{\mathrm{ct}}_2) \rceil \pmod q$ satisfies $\langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}} \rangle \approx \Delta \cdot m_1 m_2 \pmod q$ similar to the ordinary BFV scheme. The output $\overline{\mathrm{ct}}' \leftarrow \texttt{Relin}(\overline{\mathrm{ct}}; \overline{\mathrm{rlk}})$ holds $\langle \overline{\mathrm{ct}}', \overline{\mathrm{sk}} \rangle \approx \langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}} \rangle \approx \Delta \cdot m_1 m_2 \pmod q$.

4.2 Multi-Key CKKS

The CKKS scheme [16] is a leveled HE scheme with support for approximate fixed-point arithmetic. We assume $q = \prod_{i=0}^{L} p_i$ for some integers p_i to have a chain of ciphertext moduli $q_0 < q_1 < \cdots < q_L$ for $q_\ell = \prod_{i=0}^{\ell} p_i$. The native plaintext is a small polynomial $m \in R$, but one can pack at most $(n/2)$ complex numbers in a single polynomial via DFT. In addition to the basic arithmetic operations, it supports the rescaling algorithm to control the magnitude of encrypted message. For homomorphic operations between ciphertexts at different levels, it requires to transform a high-level ciphertext to have the same level as the other.

- $\underline{\texttt{MK-CKKS.Enc}(m; \mathbf{b}, \mathbf{a})}$: Let $m \in R$ be an input plaintext and let $a = \mathbf{a}[0]$ and $b = \mathbf{b}[0]$. Sample $v \leftarrow \chi$ and $e_0, e_1 \leftarrow \psi$. Return the ciphertext $\mathrm{ct} = (c_0, c_1) \in R_q^2$ where $c_0 = v \cdot b + m + e_0 \pmod q$ and $c_1 = v \cdot a + e_1 \pmod q$.
- $\underline{\texttt{MK-CKKS.Dec}(\overline{\mathrm{ct}}; s_1, \ldots, s_k)}$: Let $\overline{\mathrm{ct}} = (c_0, c_1, \ldots, c_k) \in R_{q_\ell}^{k+1}$ be a ciphertext at level ℓ associated to k parties and $s_1, \ldots, s_k$ be their secret keys. Set $\overline{\mathrm{sk}} = (1, s_1, \ldots, s_k)$ and return $\langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle \pmod{q_\ell}$.
- $\underline{\texttt{MK-CKKS.Add}(\overline{\mathrm{ct}}_1, \overline{\mathrm{ct}}_2)}$: Given two ciphertexts $\overline{\mathrm{ct}}_i \in R_{q_\ell}^{k+1}$ at level ℓ, return the ciphertext $\overline{\mathrm{ct}}' = \overline{\mathrm{ct}}_1 + \overline{\mathrm{ct}}_2 \pmod{q_\ell}$.
- $\underline{\texttt{MK-CKKS.Mult}(\overline{\mathrm{ct}}_1, \overline{\mathrm{ct}}_2; \{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k})}$: Given two ciphertexts $\overline{\mathrm{ct}}_i \in R_{q_\ell}^{k+1}$ at level ℓ, compute $\overline{\mathrm{ct}} = \overline{\mathrm{ct}}_1 \otimes \overline{\mathrm{ct}}_2 \pmod{q_\ell} \in R_{q_\ell}^{(k+1)^2}$ and return the ciphertext $\overline{\mathrm{ct}}' \leftarrow \texttt{Relin}(\overline{\mathrm{ct}}; \{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k}) \in R_{q_\ell}^{k+1}$. The relinearization algorithm is defined over modulus $q = q_L$, but we compute the same algorithm modulo q_ℓ for level-ℓ ciphertexts.
- $\underline{\texttt{MK-CKKS.Rescale}(\overline{\mathrm{ct}})}$: Given a ciphertext $\overline{\mathrm{ct}} = (c_0, c_1, \ldots, c_k) \in R_{q_\ell}^{k+1}$ at level ℓ, compute $c_i' = \left\lfloor p_\ell^{-1} \cdot c_i \right\rceil$ for $0 \le i \le k$ and return the ciphertext $\overline{\mathrm{ct}}' = (c_0', c_1', \ldots, c_k') \in R_{q_{\ell-1}}^{k+1}$.

A level-ℓ multi-key encryption of a plaintext m with respect to the secret $\overline{\mathrm{sk}} = (1, s_1, \ldots, s_k)$ is a vector $\overline{\mathrm{ct}} = (c_0, c_1, \ldots, c_k) \in R_{q_\ell}^{k+1}$ satisfying $\langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle \approx m \pmod{q_\ell}$. For basic homomorphic operation, we take as input level-ℓ encryptions of m_1 and m_2. Then, homomorphic addition (resp. multiplication) returns a ciphertext $\overline{\mathrm{ct}}'$ such that $[\langle \overline{\mathrm{ct}}', \overline{\mathrm{sk}} \rangle]_{q_\ell}$ is approximately equal to $m_1 + m_2$ (resp. $m_1 m_2$). Finally, we show that for a level-ℓ encryption $\overline{\mathrm{ct}}$ of m, the rescaling algorithm returns a ciphertext $\overline{\mathrm{ct}}'$ at level $(\ell-1)$ encrypting $p_\ell^{-1} \cdot m$ from the equation $[\langle \overline{\mathrm{ct}}', \overline{\mathrm{sk}} \rangle]_{q_{\ell-1}} \approx p_\ell^{-1} \cdot [\langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle]_{q_\ell}$.

4.3 Distributed Decryption

In the classical definition of MKHE primitive, all the secrets of the parties involved are required to decrypt a multi-key ciphertext. In practice, however, it is not reasonable to assume that there is a party holding multiple secret keys. Instead, we can think about a protocol between several key owners to jointly decrypt a ciphertext. The decryption algorithms of our schemes are (approximate) linear combinations of secrets with known coefficients, and there have been proposed some secure methods for this task. We introduce one simple solution based on the noise flooding technique, but any secure solution achieving the same functionality can be used.

The distributed decryption consists of two algorithms: partial decryption and merge. In the first phase, each party i receives the i-th entry of a ciphertext and decrypts it with a noise. We set the noise distribution ϕ which has a larger variance than the standard error distribution ψ of basic scheme. Then, we merge partially decrypted results with c_0 to recover the message.

- $\underline{\texttt{MKHE.PartDec}(c_i, s_i)}$: Given a polynomial c_i and a secret s_i, sample an error $e_i \leftarrow \phi$ and return $\mu_i = c_i \cdot s_i + e_i \pmod q$.
- $\underline{\texttt{MK-BFV.Merge}(c_0, \{\mu_i\}_{1 \le i \le k})}$: Compute $\mu = c_0 + \sum_{i=1}^{k} \mu_i \pmod q$ and return $m = \lfloor (t/q) \cdot \mu \rceil$.
- $\underline{\texttt{MK-CKKS.Merge}(c_0, \{\mu_i\}_{1 \le i \le k})}$: Compute and return $\mu = c_0 + \sum_{i=1}^{k} \mu_i \pmod q$.

For a multi-key ciphertext $\overline{\mathrm{ct}} = (c_0, \ldots, c_k)$, both multi-key BFV and CKKS schemes compute $\mu = c_0 + \sum_{i=1}^{k} \mu_i = \langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle + \sum_{i=1}^{k} e_i \approx \langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle \pmod q$ in the merge phase. Then, BFV extracts the plaintext by cancelling the scaling factor (q/t).

5 BOOTSTRAPPING FOR TWO MKHE SCHEMES

There have been several studies on the bootstrapping procedures of the standard (single-key) ring-based HE schemes [9, 12, 14, 25, 32]. Previous work had different goals and solutions depending on the underlying schemes but they are basically following the Gentry's technique [24] – homomorphic evaluation of the decryption circuit. In particular, the BFV and CKKS schemes have a very similar pipeline for bootstrapping which consists of four steps: (1) Modulus Raise, (2) Coeff to Slot, (3) Extraction and (4) Slot to Coeff. The second and last steps are specific linear transformations, which require rotation operations on encrypted vectors.

For the rest of this section, we first explain how to perform the rotation operation on multi-key ciphertexts based on the evaluation of Galois automorphisms. Then, we revisit the bootstrapping procedures for BFV and CKKS to generalize the existing solutions to our MKHE schemes.

5.1 Homomorphic Evaluation of Galois Automorphisms

The Galois group $\mathcal{G}\mathrm{al}(\mathbb{Q}[X]/(X^n+1))$ of a cyclotomic field consists of the transformation $X \mapsto X^j$ for $j \in \mathbb{Z}_{2n}^*$. We recall that BFV (resp. CKKS) uses the DFT on R_t (resp. R) to pack multiple plaintext values into a single polynomial. As noted in [26], these automorphisms provide special functionalities on packed ciphertext such as rotation of plaintext slots.

The evaluation of an automorphism can be done based on the key-switching technique. In some more details, let $\tau_j : a(X) \mapsto a(X^j)$ be an element of the Galois group. Given an encryption $\overline{\mathrm{ct}} = (c_0, c_1, \ldots, c_k) \in R_q^{k+1}$ of m, we denote by $\tau_j(\overline{\mathrm{ct}}) = (\tau_j(c_0), \ldots, \tau_j(c_k))$ the ciphertext obtained by taking τ_j to the entries of $\overline{\mathrm{ct}}$. Then $\tau_j(\overline{\mathrm{ct}})$ is a valid encryption of $\tau_j(m)$ corresponding the secret key $\tau_j(\overline{\mathrm{sk}})$. We then perform the key-switching procedure from $\tau_j(\overline{\mathrm{sk}})$ back to $\overline{\mathrm{sk}}$, so as to generate a new ciphertext encrypting the same message under the original secret key $\overline{\mathrm{sk}}$.

In the following, we present two algorithms for the evaluation of the Galois element. The first algorithm generates an evaluation key for the Galois automorphism τ_j. The second algorithm gathers the evaluation keys of multiple parties and evaluates τ_j on a multi-key ciphertext using the multi-key-switching technique proposed in [10].

- `MKHE.GkGen`$(j; s)$: Generate a random vector $\mathbf{h}_1 \leftarrow U(R_q^d)$ and an error vector $\mathbf{e}' \leftarrow \psi^d$. For an RLWE secret $s \in R$, compute $\mathbf{h}_0 = -s \cdot \mathbf{h}_1 + \mathbf{e}' + \tau_j(s) \cdot \mathbf{g} \pmod q$. Return the Galois evaluation key as $\mathrm{gk} = [\mathbf{h}_0 | \mathbf{h}_1] \in R_q^{d\times 2}$.

- `MKHE.EvalGal`$(\overline{\mathrm{ct}}; \{\mathrm{gk}_i\}_{1\le i\le k})$: Let $\mathrm{gk}_i = [\mathbf{h}_{i,0} | \mathbf{h}_{i,1}]$ be the Galois evaluation key of the i-th party for $1 \le i \le k$. Given a ciphertext $\overline{\mathrm{ct}} = (c_0, \ldots, c_k) \in R_q^{k+1}$, compute and return the ciphertext $\overline{\mathrm{ct}}' = (c_0', \ldots, c_k')$ by

$$c_0' = \tau_j(c_0) + \sum_{i=1}^{k} \langle \mathbf{g}^{-1}(\tau_j(c_i)), \mathbf{h}_{i,0} \rangle \pmod q, \quad \text{and}$$
$$c_i' = \langle \mathbf{g}^{-1}(\tau_j(c_i)), \mathbf{h}_{i,1} \rangle \pmod q \quad \text{for} \quad 1 \le i \le k.$$

In the context of CKKS, all the computations are carried out over modulus $q = q_\ell$ for level-ℓ ciphertext. We now show the correctness of our algorithms.

Correctness. From the definition, the output ciphertext $\overline{\mathrm{ct}}' = (c_0', \ldots, c_k') \leftarrow$ `MKHE.EvalGal`$(\overline{\mathrm{ct}}; \{\mathrm{gk}_i\}_{1\le i\le k})$ holds

$$\begin{aligned}
\langle \overline{\mathrm{ct}}', \overline{\mathrm{sk}} \rangle &= c_0' + \sum_{i=1}^{k} c_i' \cdot s_i \\
&= \tau_j(c_0) + \sum_{i=1}^{k} \langle \mathbf{g}^{-1}(\tau_j(c_i)), \mathbf{h}_{i,0} \rangle + \langle \mathbf{g}^{-1}(\tau_j(c_i)), \mathbf{h}_{i,1} \rangle \cdot s_i \pmod q \\
&\approx \tau_j(c_0) + \sum_{i=1}^{k} \langle \mathbf{g}^{-1}(\tau_j(c_i)), \tau_j(s_i) \cdot \mathbf{g} \rangle \pmod q \\
&= \left\langle \tau_j(\overline{\mathrm{ct}}), \tau_j(\overline{\mathrm{sk}}) \right\rangle = \tau_j\left(\langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle \right) \pmod q,
\end{aligned}$$

as desired. In other words, if the input ciphertext has the phase $\mu(X) = \langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle \pmod q$, then the phase of the output ciphertext is approximately equal to $\tau_j(\mu(X)) = \mu(X^j)$.

Besides the rotation of plaintext slots, we can evaluate the Frobenius endomorphism $X \mapsto X^t$ on BFV ciphertexts using the same technique. In the case of CKKS, the map $X \mapsto X^{-1}$ corresponds to the complex conjugation over plaintext slots.

Any linear transformation can be represented as a linear combination of shifted plaintext vectors. We note that previous HE optimization techniques [9, 25, 32] for linear transformations can be directly applied to our MKHE schemes.

5.2 Bootstrapping for Multi-Key BFV

Chen and Han [12] described a bootstrapping procedure for the single-key BFV scheme, which follows the paradigm of [32], done for BGV scheme. The bootstrapping procedure in [12] takes as input a ciphertext with an arbitrary noise and outputs another ciphertext with a low noise encrypting the same plaintext. Below we present a multi-key variant of [12].

(1) The previous work [12] published an encryption of the secret key by itself to raise the modulus. However, we observe that this step can be done by multiplying a constant without extra information. Suppose that the input ciphertext $\overline{\mathrm{ct}}$ encrypts a message m with a plaintext modulus t, i.e., $\langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle = \frac{q}{t} m + e \pmod q$ for some error e. Then we perform a modulus-switching down to a divisor q' of q, resulting in $\langle \overline{\mathrm{ct}}', \overline{\mathrm{sk}} \rangle = \frac{q'}{t} m + e' \pmod{q'}$, then multiply the ciphertext with q/q' and get a ciphertext $\overline{\mathrm{ct}}''$ whose phase is $\langle \overline{\mathrm{ct}}'', \overline{\mathrm{sk}} \rangle = \frac{q}{q'} \cdot \left(\frac{q'}{t} m + e' \right) \pmod q$. This is a trivial (noise free) encryption of $\mu = \frac{q'}{t} m + e'$ with plaintext modulus q' and ciphertext modulus q.

(2) It computes a homomorphic linear transform which produces multiple ciphertexts holding the coefficients $\mu_i \in \mathbb{Z}_t$ of μ in their plaintext slots. We note that this step can be done using the additions, scalar multiplications and multi-key rotations.

(3) We homomorphically evaluate a polynomial, called lower digits removal [12], on the multi-key ciphertexts obtained in previous step. It removes the noise e' and leaves the coefficients of m in plaintext slots.

(4) The final step is another linear transformation which inverts the second step and outputs an encryption of m.

As a consequence, the output ciphertext has the phase $\frac{q}{t} m + e'' \pmod q$ for an error which is smaller than the initial noise e.

5.3 Bootstrapping for Multi-Key CKKS

Cheon et al. [14] presented a bootstrapping procedure for the single-key CKKS scheme and its performance was improved in the follow-up research [9]. The bootstrapping procedure of CKKS aims to refresh a low-level ciphertext and return an encryption of the (almost) same messages in a larger ciphertext modulus. We describe its multi-key version as follows.

(1) The first step takes a lowest-level ciphertext $\overline{\mathrm{ct}}$ as an input. Let $\mu = \langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle \pmod{q_0}$. Then $\langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \rangle = q_0 \cdot I + \mu$ for a

small $I \in R$, so $\overline{\mathrm{ct}}$ can be considered as an encryption of $t = q_0 \cdot I + \mu$ in the largest ciphertext modulus q_L.

(2) We apply a homomorphic linear transformation to compute one or two ciphertexts encrypting the coefficients of $t(X)$ in their plaintext slots. This step requires multi-key rotation and conjugation described in Section 5.1.
(3) We evaluate a polynomial which approximates the reduction modular q_0 function. It removes the I part of t and leaves coefficients of μ in the slots.
(4) Finally, we apply the inverse linear transformation of the second step to pack all the coefficients of μ back into a ciphertext.

The output ciphertext $\overline{\mathrm{ct}}'$ encrypts the same plaintext μ in a higher level than the input ciphertext $\overline{\mathrm{ct}}$, i.e., $\langle \overline{\mathrm{ct}}', \overline{\mathrm{sk}} \rangle \approx \mu \pmod{q_\ell}$ for some $0 < \ell < L$.

6 IMPLEMENTATION

We provide a proof-of-concept implementation to show the performance of our MKHE schemes. Our source code is developed in C++ with Microsoft SEAL version 3.2.0 [47] which includes BFV and CKKS implementations. We summarize our optimization techniques, recommended parameter sets, and some experimental results in this section. Finally, we apply the multi-key CKKS scheme to evaluate an encrypted neural network model on encrypted data and report the experimental result to classify handwritten images on the MNIST dataset [40]. All experiments are performed on a ThinkPad P1 laptop: Intel Xeon E-2176M @ 4.00 GHz single-threaded with 32 GB memory, compiled with GNU C++ 7.3.0 (-O2). In our implementation, we set the secret distribution χ as the uniform distribution over the set of polynomials in R whose coefficients are in $\{0, \pm 1\}$. Each coefficient of an error $e \leftarrow \psi$ is drawn according to the discrete Gaussian distribution centered at zero with standard deviation $\sigma = 3.2$.

6.1 Optimization Techniques

6.1.1 Basic Optimizations. In the relinearization process, we first compute the tensor product of two ciphertexts which corresponds to the tensor squared secret $\overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}}$. It has duplicated entries at (i, j) and (j, i), so we can reduce its dimension from $(k+1)^2$ down to $\frac{1}{2}k(k+1)$. Both the size of the relinearization key and complexity of the algorithm are almost halved. Furthermore, each of the diagonal entries s_i^2 of $\overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}}$ depends on a single party, so we can include a key-switching key for s_i^2 in the generation of an evaluation key. It increases the size of evaluation keys but reduces the complexity and noise of relinearization.

6.1.2 RNS and NTT. Our schemes are designed on the ring structure R_q, so we need to optimize the basic polynomial arithmetic. There is a well-known technique to use an RNS by taking a ciphertext modulus $q = \prod_{i=0}^{L} p_i$ which is a product of coprime integers. Based on the ring isomorphism $R_q \to \prod_{i=0}^{L} R_{p_i}$, $a \mapsto (a \pmod{p_i})_{0 \le i \le L}$, we achieve asymptotic/practical improvements in polynomial arithmetic over R_q. In particular, it has been studied how to design full-RNS variants of BFV and CKKS [4, 15, 30], which do not require any RNS conversions. In addition, each of base prime can be chosen properly so that there exists a $(2n)$-th root of unity modulo p_i. It allows us to exploit an efficient Number Theoretic Transformation (NTT) modulo p_i. Our implementation adapts these techniques to improve the speed of polynomial arithmetic.

Parameter					Public Key		Evaluation Key	
ID	n	$\lceil \log q \rceil$	$\lceil \log p_i \rceil$	# $p_i' s$	Size	Gen.	Size	Gen.
I	2^{13}	218	49–60	4	0.75 MB	3 ms	2.25 MB	8 ms
II	2^{14}	438	53–60	8	7 MB	24 ms	21 MB	59 ms
III	2^{15}	881	54–60	16	60 MB	195 ms	180 MB	470 ms

Table 2: Proposed parameter sets. $\log q$ **and** $\log p_i$ **denote the bit lengths of the largest RLWE modulus and individual RNS primes, respectively. #** $p_i' s$ **denotes the number of RNS primes. Public keys' and evaluations keys' generation times and sizes are those of each party. ms** $= 10^{-3}$ **sec.**

6.1.3 Gadget Decomposition. As mentioned before, the gadget decomposition has a major effect on the performance of homomorphic arithmetic. Bajard et al. [4] observed that the formula $a = \sum_i g_i \cdot [a]_{p_i} \pmod{q}$ where $g_i = \left[\left(\prod_{j \neq i} p_j\right)^{-1}\right]_{p_i} \cdot \left(\prod_{j \neq i} p_j\right)$ can be used to build an RNS-friendly decomposition $a \mapsto ([a]_{p_i})_i$ with the gadget vector $\mathbf{g} = (g_i)_i$. We adapt this decomposition method and take an advantage of an RNS-based implementation by storing ciphertexts in the RNS form.

In [4, 30], the authors further combined this method with the classical digit decomposition method to provide a more fine-grained control of the trade-off between complexity and noise growth. However, we realize that this hybrid method increases the decomposition degree (and thereby space and computational complexity) several times, and the special modulus technique described in Section 3.3.4 provides a much better trade-off. Therefore, the digit decomposition is *not* used in our implementation.

6.2 Micro-benchmarks for MKHE Schemes

Table 2 illustrates the selected parameter sets used in experiments. They are default parameter sets in Microsoft SEAL which provide at least 128-bit of security level according to LWE-estimator [3] and HE security standard [2]. Generation time and size of secret keys, and execution time of encryption are the same as those in single-key BFV and CKKS. Decryption and ciphertext addition take $\frac{1}{2}(k+1)$ times longer than the ordinary HE schemes. We remark that generation time and size of public and evaluation keys $\{(\mathbf{b}_i, \mathbf{D}_i)\}_{1 \le i \le k}$ do not depend on the number of parties or the scheme because the generation can be executed in a synchronous way.

In our experiments, a homomorphic multiplication is always followed by a relinearization procedure. BFV requires more NTTs than CKKS overall to perform these operations, and is therefore slower, which is confirmed by the timing results.

The execution times of multiplications in both MKHE schemes are asymptotically quadratic in the number of parties as discussed in Section 3.3. In practice, they are better than quadratic, as reported in Table 3. This is because both multiplication and relinearization

ID	#Parties	Mult + Relin		EvalGal	
		BFV	CKKS	BFV	CKKS
I	1	20 ms	8 ms	3 ms	4 ms
	2	44 ms	22 ms	7 ms	8 ms
	4	116 ms	67 ms	14 ms	16 ms
	8	365 ms	229 ms	28 ms	31 ms
II	1	110 ms	59 ms	22 ms	24 ms
	2	257 ms	165 ms	47 ms	49 ms
	4	717 ms	521 ms	88 ms	95 ms
	8	2, 350 ms	1, 845 ms	176 ms	193 ms
III	1	675 ms	465 ms	170 ms	172 ms
	2	1, 715 ms	1, 364 ms	333 ms	359 ms
	4	5, 025 ms	4, 287 ms	646 ms	711 ms
	8	17, 450 ms	15, 159 ms	1, 332 ms	1, 413 ms

Table 3: Execution time that depends on the number of parties. Multiplication is always followed by a relinearization in MKHE. ms = 10^{-3} sec.

include a notable portion of computation that is linear in the number of parties. The execution times of homomorphic evaluation of Galois automorphisms are almost linear on the number of the parties as described in Section 5.1.

For the single-party scenario in Table 3, we measured performance of our modified Microsoft SEAL [47] with a special modulus. It is infeasible to fairly compare the ordinary BFV and CKKS with their multi-key variants because the performance of a scheme can be analyzed from various perspectives: space/time complexity, noise growth, functionality, etc. It is provided merely as a reference point, for a more portable estimation of MKHE on different processors.

6.3 Application to Oblivious Neural Network Inference

Jiang et al. [34] proposed a novel framework to test encrypted neural networks on encrypted data in the single-key scenario. We consider the same service paradigm but in a multi-key setting: the data and trained model are encrypted under different keys.

6.3.1 Homomorphic Evaluation of CNN.. We present an efficient strategy to evaluate CNN prediction model on the MNIST dataset. Each image is a 28×28 pixel array and will be labeled with 10 possible digits after an arbitrary number of hidden layers. We assume that a neural network is trained with the plaintext dataset in the clear. Table 4 describes our neural network topology which uses one convolution layer and two fully-connected (FC) layers with square activation function. The final step is to apply the softmax activation function for a purpose of probabilistic classification, so it is enough to obtain an index of maximum values of outputs in a prediction phase. Our objective is to predict a single image in an efficient way, thereby achieving a low latency. In Appendix D, we describe the detailed algorithms for encryption and evaluation.

Layer	Description
Convolution	Input image 28×28, window size 4×4, stride $(2, 2)$, number of output channels 5
1st square	Squaring each of the 845 inputs
FC-1	Fully connecting with 845 inputs and 64 outputs
2nd square	Squaring each of the 64 inputs
FC-2	Fully connecting with 64 inputs and 10 outputs

Table 4: Description of our CNN to the MNIST dataset.

The Convolutional Layer. As noted in [35], strided convolution can be decomposed into a sum of simple convolutions (i.e., the stride parameter = 1). From our choice of the parameters, each of such simple convolutions takes as inputs 14×14 images and 2×2 filters. This representation allows more SIMD parallelism, since we can pack all the inputs into a single ciphertext and perform four simple convolutions in parallel. Once this is done, we can accumulate the results across plaintext slots using rotate-and-sum operations in [31]. Moreover, we can pack multiple channels in a single ciphertext as in [35, Section VI.D], yielding in a fully-packed ciphertext of the convolution result.

The First Square Layer. This step applies the square activation function to all the encrypted output of the convolutional layer in an SIMD manner.

The FC-1 Layer. In general, an FC layer with n_i inputs and n_o outputs can be computed as a matrix-vector multiplication. Let $\mathbf{W}$ and $\mathbf{v}$ be the $n_o \times n_i$ weight matrix and n_i-length vector, respectively. We assume that n_i and n_o are smaller than the number of plaintext slots, and n_o is much lower than n_i in the context of FC layers. Halevi and Shoup [31] presented the diagonal encoding method which puts a square matrix in diagonal order, multiplies each of them with a rotation of the input vector, and then accumulates all the output vectors to obtain the result. Juvekar et al. [35] extended the method to multiply a vector by a rectangular matrix. If the input vector is encrypted in a single ciphertext, the total complexity is n_o homomorphic multiplications, $(n_o - 1)$ rotations of the input ciphertext of $\mathbf{v}$, and $\log(n_i/n_o)$ rotations for rotate-and-sum algorithm.

We extend their ideas to split the original matrix $\mathbf{W}$ into smaller sized blocks and perform computation on the sub-matrices as shown in Fig. 2. Suppose that the vector $\mathbf{v}$ is split into ℓ many sub-strings with the same length. For simplicity, we consider the first ℓ rows of $\mathbf{W}$. We first apply the diagonal method to arrange the $1 \times (n_i/\ell)$ sized sub-matrices of $\mathbf{W}$ in a way that intermediate numbers are aligned in the same position across multiple slots after homomorphic multiplications. To be precise, the encryptions of diagonal components are multiplied with ℓ rotations of the encrypted vector and all these encryptions are added together similar to the diagonal method. Then, the output ciphertext represents (n_i/ℓ)-sized ℓ chunks, each containing partial sums of ℓ entries of n_i inputs. Finally, we can accumulate these using a rotate-and-sum algorithm with $\log(n_i/\ell)$ rotations. As a consequence, the output ciphertext encrypts the first ℓ many entries of $\mathbf{Wv}$. We repeat this procedure for each ℓ many rows of $\mathbf{W}$, resulting in (n_o/ℓ) ciphertexts.

When n_i is significantly smaller than the number of plaintext slots n_s, the performance can be improved by packing multiple copies of the input vector into a single ciphertext and performing (n_s/n_i) aforementioned operations in parallel. The computational cost is $(n_o \cdot n_i)/n_s$ homomorphic multiplications, $(\ell - 1)$ rotations of

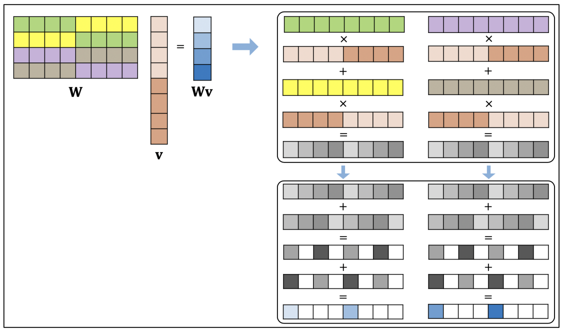

Figure 2: Our matrix-vector multiplication algorithm ($\ell = 2$).

the input ciphertext of $\mathbf{v}$, and $(n_o \cdot n_i)/(n_s \cdot \ell) \cdot \log(n_i/\ell)$ rotations. We provide additional details in Appendix D.2. As a result, our method provides a trade-off between rotations on the same input ciphertext (which can benefit from the hoisting optimization of [33]) and rotations on distinct ciphertexts (which cannot benefit from hoisting).

As described in Fig. 2, all slots except the ones corresponding to the result components may reveal information about partial sums. We therefore multiply the output ciphertexts by a constant zero-one plaintext vector to remove the information.

The Second Square Layer. This step applies the square activation function to all the output nodes of the first FC layer.

The FC-2 Layer. This step performs a multiplication with small sized weight matrix $\mathbf{U}$ and vector $\mathbf{v}$. As discussed in [31], it can be considered as the linear combination of $\mathbf{U}$'s columns using coefficients from $\mathbf{v}$. Suppose that the column vectors are encrypted in a single ciphertext in such a way that they are aligned with the encrypted vector. We first repeatedly rotate the encryption of the vector to generate a single ciphertext with n_o copies of each entry. Then, we apply pure SIMD multiplication to multiply each column vector by the corresponding scalar of the vector in parallel. Finally, we aggregate all the resulting columns over the slots to generate the final output.

6.3.2 *Performance Evaluation.*

We evaluated our framework to classify encrypted handwritten images of the MNIST dataset. We used the library keras [18] with Tensorflow [1] to train the CNN model from 60,000 images of the dataset. We employ the special modulus variant of the multi-key CKKS scheme to achieve efficiency of approximate computation. Each layer of the network has a depth of one homomorphic multiplication (except the first FC layer requiring one more depth for multiplicative masking), so it requires 6 levels for the evaluation of CNN. We chose the parameter Set-II from Table 2 so as to cope with such levels of computations.

The data owner chooses one among 10,000 test images from MNIST dataset, normalizes it by dividing by the maximum value 255, and encrypts it into a single ciphertext using the public key, which takes 1.75 MB of space. Meanwhile, the model provider generates a relatively large number of ciphertexts for the trained model: four for the multiple channels, eight for the weight matrix of the FC-1 layer, and one of each for the other weight or bias. Therefore, the total size of the output ciphertexts is 18.5 MB and it takes roughly 7 times longer to encrypt the trained model than an image, but it is an one-time process before data outsourcing and so it is a negligible overhead. After the evaluation, the cloud server outputs a single multi-key ciphertext encrypting the prediction result with respect to the extended secret key of the data and model owner. Table 5 presents running time of the CNN evaluation. It takes about 1.8 seconds to classify an image using a pre-trained model.

	Stage	Runtime
Data owner	Image encryption	31 ms
Model provider	Model encryption	236 ms
Cloud server	Convolutional layer	705 ms
	1^{st} square layer	143 ms
	FC-1 layer	739 ms
	2^{nd} square layer	75 ms
	FC-2 layer	135 ms
	Total evaluation	**1,797 ms**

Table 5: Performance breakdown for evaluating an encrypted neural network on encrypted MNIST data, where the two encryptions are under different secret keys.

Framework	Methodology	Runtime	
		Latency	Amortized
CryptoNets	HE	570 s	0.07 s
MiniONN	HE, MPC	1.28 s	-
Gazelle	HE, MPC	0.03 s	-
E2DM	HE	28.59 s	0.45 s
Ours	MKHE	1.80 s	-

Table 6: MNIST benchmarks of privacy-preserving neural network frameworks.

Our parameter guarantees at least 32-bit precision after the decimal point. That is, the infinity norm distance between encrypted evaluation and plain computation is bounded by 2^{-32}. Therefore, we had enough space to use the noise flooding technique for decryption. In terms of the accuracy, it achieves about 98.4% on the test set which is the same as the one obtained from the evaluation in the clear.

6.3.3 *Comparison with Previous Works.*

In Table 6, we compare our benchmark result with the state-of-the-art frameworks for oblivious neural network inference: CryptoNets [29], MiniONN [42], Gazelle [35], and E2DM [34]. The first column indicates the framework and the second column denotes the cryptographic primitives used for preserving privacy. The last columns give running time for image classification as well as amortized time per instance.

Among the aforementioned solutions for private neural network prediction, E2DM relies on a third-party authority holding a secret key of HE, since the data and model are under the same secret key. CryptoNets has good amortized complexity, but it has a high latency for a single prediction. MiniONN and Gazelle have good latency, but they require both parties to be online during the protocol execution, and at least one party performs local work proportional to the complexity of the function being evaluated. Also, the number of round scales with the number of layers for MiniONN and Gazelle in the neural network. On the other hand, our solution has a constant number of rounds.

Moreover, our solution allows the parties to outsource homomorphic evaluation to an untrusted server (e.g. a VM in the cloud with large computing power), so both parties only need to pay encryption/decryption cost, and the communication cost only scales with the input/model sizes, but not the complexity of the network itself. This feature is made possible since our scheme supports multi-key operations. Note that the server is only assumed to be semi-honest: we do *not* require non-collusion assumptions since even if the server colludes with one party, they cannot learn the other party's private inputs due to the IND-CPA security of MKHE. Therefore, we believe that our work presents an interesting point in the design space of oblivious machine learning inference. Last but not least, MiniONN and Gazelle have communication cost scaling linearly with the number of activation nodes in the neural network due to the use of garbled circuits. In our case, the size of the ciphertexts and public/evaluation keys only depends on the number of layers and the number of input nodes of the neural network. Hence our solution is asymptocially more communication size-efficient.

7 CONCLUSION

In this paper, we presented practical multi-key variants of the BFV and CKKS schemes and their bootstrapping methods. We provided the first experimental results of MKHE with packed ciphertexts by implementing our schemes. The main technical contribution is to propose new relinearization algorithms achieving better performance compared to prior works [13, 41]. Finally, we showed that our scheme can be applied to secure on-line prediction services by evaluating an encrypted classifier on an encrypted data under two different keys. We implemented our protocol on CNNs trained on the MNIST dataset and showed that it can achieve a low end-to-end latency by leveraging the optimized homomorphic convolutions and homomorphic matrix-vector multiplications.

ACKNOWLEDGMENTS

M. Kim was supported in part by the Cancer Prevention Research Institute of Texas (CPRIT) under award number RR180012, UT STARs award, and the National Institute of Health (NIH) under award numbers U01TR002062, R01GM118574, R01GM124111, and R41HG010978.

REFERENCES

[1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2015. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. https://www.tensorflow.org.

[2] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2018. *Homomorphic Encryption Security Standard*. Technical Report. HomomorphicEncryption.org, Toronto, Canada.

[3] Martin R Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology* 9, 3 (2015), 169–203.

[4] Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. 2016. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *International Conference on Selected Areas in Cryptography*. Springer, 423–442.

[5] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 1–10.

[6] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *CRYPTO 2012*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Lecture Notes in Computer Science, Vol. 7417. Springer, 868–886.

[7] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *Proc. of ITCS*. ACM, 309–325.

[8] Zvika Brakerski and Renen Perlman. 2016. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In *Annual Cryptology Conference*. Springer, 190–213.

[9] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Improved bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 34–54.

[10] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Multi-Key Homomophic Encryption from TFHE. Cryptology ePrint Archive, Report 2019/116. https://eprint.iacr.org/2019/116.

[11] Hao Chen, Ran Gilad-Bachrach, Kyoohyung Han, Zhicong Huang, Amir Jalali, Kim Laine, and Kristin Lauter. 2018. Logistic regression over encrypted data from fully homomorphic encryption. *BMC medical genomics* 11, 4 (2018), 81.

[12] Hao Chen and Kyoohyung Han. 2018. Homomorphic Lower Digits Removal and Improved FHE Bootstrapping. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 315–337.

[13] Long Chen, Zhenfeng Zhang, and Xueqing Wang. 2017. Batched Multi-hop Multi-key FHE from Ring-LWE with Compact Ciphertext Extension. In *Theory of Cryptography Conference*. Springer, 597–627.

[14] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for Approximate Homomorphic Encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 360–384.

[15] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. A full RNS variant of approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*. Springer.

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

[17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. 2016. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology – ASIACRYPT 2016*. Springer, 3–33.

[18] François Chollet et al. 2015. Keras. https://github.com/keras-team/keras.

[19] Michael Clear and Ciaran McGoldrick. 2015. Multi-identity and multi-key leveled FHE from learning with errors. In *Annual Cryptology Conference*. Springer, 630–656.

[20] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. 2013. Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits. In *European Symposium on Research in Computer Security*. Springer, 1–18.

[21] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology–EUROCRYPT 2015*. Springer, 617–640.

[22] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144. https://eprint.iacr.org/2012/144.

[23] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. 2017. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies* 2017, 4 (2017), 345–364.

[24] Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing (STOC '09)*. ACM, 169–178. https://doi.org/10.1145/1536414.1536440

[25] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Better bootstrapping in fully homomorphic encryption. In *Public Key Cryptography–PKC 2012*. Springer, 1–16.

[26] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Fully Homomorphic Encryption with Polylog Overhead. In *EUROCRYPT 2012*, David Pointcheval and Thomas Johansson (Eds.). Lecture Notes in Computer Science, Vol. 7237. Springer, 465–482.

[27] Craig Gentry, Shai Halevi, and Nigel P. Smart. 2012. Homomorphic Evaluation of the AES Circuit. In *Advances in Cryptology - CRYPTO 2012 (Lecture Notes in Computer Science)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.), Vol. 7417. Springer, 850–867.

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

[29] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*. 201–210.

[30] Shai Halevi, Yuriy Polyakov, and Victor Shoup. 2019. An improved RNS variant of the BFV homomorphic encryption scheme. In *Cryptographers' Track at the RSA Conference*. Springer, 83–105.

[31] Shai Halevi and Victor Shoup. 2014. Algorithms in HElib. In *Advances in Cryptology–CRYPTO 2014*. Springer, 554–571.

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

[33] Shai Halevi and Victor Shoup. 2018. Faster homomorphic linear transformations in HElib. In *Annual International Cryptology Conference*. Springer, 93–120.

[34] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. 2018. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1209–1222.

[35] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD.

[36] Marcel Keller, Valerio Pastro, and Dragos Rotaru. 2018. Overdrive: making SPDZ great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 158–189.

[37] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. 2018. Logistic Regression Model Training based on the Approximate Homomorphic Encryption. *BMC medical genomics* 11, 4 (2018), 83.

[38] Miran Kim, Yongsoo Song, Baiyu Li, and Daniele Micciancio. 2019. Semi-parallel logistic regression for GWAS on encrypted data. Cryptology ePrint Archive, Report 2019/294. https://eprint.iacr.org/2019/294.

[39] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. 2018. Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation. *JMIR medical informatics* 6, 2 (2018).

[40] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann.lecun.com/exdb/mnist/* (1998).

[41] Ningbo Li, Tanping Zhou, Xiaoyuan Yang, Yiliang Han, Wenchao Liu, and Guangsheng Tu. 2019. Efficient Multi-Key FHE With Short Extended Ciphertexts and Directed Decryption Protocol. *IEEE Access* 7 (2019), 56724–56732.

[42] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 619–631.

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

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

[45] Pratyay Mukherjee and Daniel Wichs. 2016. Two round multiparty computation via multi-key FHE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 735–763.

[46] Chris Peikert and Sina Shiehian. 2016. Multi-key FHE from LWE, revisited. In *Theory of Cryptography Conference*. Springer, 217–238.

[47] SEAL 2019. Microsoft SEAL (release 3.2). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA.

[48] Nigel P Smart and Frederik Vercauteren. 2014. Fully homomorphic SIMD operations. *Designs, codes and cryptography* 71, 1 (2014), 57–81.

[49] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Global-scale secure multiparty computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 39–56.

[50] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*. IEEE, 162–167.

A SPECIAL MODULUS VARIANTS OF OUR MKHE SCHEMES

• $\underline{\texttt{MKHE.Setup}(1^\lambda)}$: Given a security parameter λ, set the RLWE dimension n, ciphertext modulus q, special modulus p, key distribution χ and error distribution ψ over R. Generate a random vector $\mathbf{a} \leftarrow U(R_{pq}^d)$. Return the public parameter $pp = (n, p, q, \chi, \psi, \mathbf{a})$.

• $\underline{\texttt{UniEnc}(\mu; s)}$: For an input plaintext $\mu \in R$, generate a ciphertext $\mathbf{D} = [\mathbf{d}_0|\mathbf{d}_1|\mathbf{d}_2] \in R_{pq}^{d\times 3}$ as follows:

(1) Sample $r \leftarrow \chi$.

(2) Sample $\mathbf{d}_1 \leftarrow U(R_{pq}^d)$ and $\mathbf{e}_1 \leftarrow \psi^d$, and set $\mathbf{d}_0 = -s \cdot \mathbf{d}_1 + \mathbf{e}_1 + pr \cdot \mathbf{g} \pmod{pq}$.

(3) Sample $\mathbf{e}_2 \leftarrow \psi^d$ and set $\mathbf{d}_2 = r \cdot \mathbf{a} + \mathbf{e}_2 + p\mu \cdot \mathbf{g} \pmod{pq}$ in R_{pq}^d.

Algorithm 3 Relinearization method with modulus raising

```
Input: $\overline{\text{ct}} = (c_{i,j})_{0 \le i,j \le k}$, $\left\{(\mathbf{D}_i = [\mathbf{d}_{i,0}|\mathbf{d}_{i,1}|\mathbf{d}_{i,2}], \mathbf{b}_i)\right\}_{1 \le i \le k}$.
Output: $\overline{\text{ct}}' = (c'_i)_{0 \le i \le k} \in R_q^{k+1}$.
$(c''_i)_{0 \le i \le k} \leftarrow \mathbf{0}$    ▷ Create a temporary vector modulo $pq$
for $1 \le i, j \le k$ do
    $c''_{i,j} \leftarrow \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{b}_j \rangle \pmod{pq}$
    $c'_{i,j} \leftarrow \lfloor p^{-1} \cdot c''_{i,j} \rceil$    ▷ It is a polynomial modulo $q$
    $(c''_0, c''_i) \leftarrow (c''_0, c''_i) + \mathbf{g}^{-1}(c'_{i,j}) \cdot [\mathbf{d}_{i,0}|\mathbf{d}_{i,1}] \pmod{pq}$
    $c''_j \leftarrow c''_j + \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{d}_{i,2} \rangle \pmod{pq}$
end for
$c'_0 \leftarrow c_{0,0} + \lfloor p^{-1} \cdot c''_0 \rceil \pmod{q}$
for $1 \le i \le k$ do
    $c'_i \leftarrow c_{0,i} + c_{i,0} + \lfloor p^{-1} \cdot c''_i \rceil \pmod{q}$
end for
```

• $\underline{\texttt{MKHE.KeyGen}(pp)}$: Each party i samples the secret key $s_i \leftarrow \chi$, an error vector $\mathbf{e}_i \leftarrow \psi^d$ and sets the public key as $\mathbf{b}_i = -s_i \cdot \mathbf{a} + \mathbf{e}_i \pmod{pq}$. Set the evaluation key $\mathbf{D}_i \leftarrow \texttt{UniEnc}(s_i; s_i)$.

• $\underline{\texttt{Relin}(\overline{\text{ct}}; \{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k})}$: Given an extended ciphertext $\overline{\text{ct}} = (c_{i,j})_{0 \le i,j \le k} \in R_q^{(k+1)^2}$ and k pairs of evaluation and public keys $\{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k} \in (R_{pq}^{d\times 3} \times R_{pq}^d)^k$, generate a ciphertext $\overline{\text{ct}}' \in R_q^{k+1}$ as described in Alg. 3.

• $\underline{\texttt{MK-CKKS.Enc}(m; \mathbf{b}_i, \mathbf{a})}$: Let $m \in R$ be an input plaintext and let $a = \mathbf{a}[0]$ and $b_i = \mathbf{b}_i[0]$ be the first entries of the common reference string and public key of the i-th party. Sample $v \leftarrow \chi$ and $e_0, e_1 \leftarrow \psi$. Return the ciphertext $\text{ct} = (m, 0) + \lfloor p^{-1} \cdot (c_0, c_1) \rceil \in R_q^2$ where $(c_0, c_1) = v \cdot (b, a) + (e_0, e_1) \pmod{pq}$.

• $\underline{\texttt{MK-CKKS.Dec}, \texttt{Add}, \texttt{Mult}, \texttt{Rescale}}$: These algorithms are the same as the ones described in Section 4.2.

• $\underline{\texttt{MKHE.GkGen}(j; s)}$: Generate a random vector $\mathbf{h}_1 \leftarrow U(R_{pq}^d)$ and an error vector $\mathbf{e}' \leftarrow \psi^d$. Compute $\mathbf{h}_0 = -s \cdot \mathbf{h}_1 + \mathbf{e}' + \tau_j(s) \cdot \mathbf{g} \pmod{pq}$. Return the Galois evaluation key as $\text{gk} = [\mathbf{h}_0|\mathbf{h}_1] \in R_{pq}^{d\times 2}$.

• $\underline{\texttt{MKHE.EvalGal}(\overline{\text{ct}}; \{\text{gk}_i\}_{1 \le i \le k})}$: Let $\text{gk}_i = [\mathbf{h}_{i,0}|\mathbf{h}_{i,1}]$ be the Galois evaluation key of the i-th party for $1 \le i \le k$. Given a ciphertext $\overline{\text{ct}} = (c_0, \ldots, c_k) \in R_q^{k+1}$, compute

$$c''_0 = \sum_{i=1}^{k} \langle \mathbf{g}^{-1}(\tau_j(c_i)), \mathbf{h}_{i,0} \rangle \pmod{pq} \quad \text{and}$$

$$c''_i = \langle \mathbf{g}^{-1}(\tau_j(c_i)), \mathbf{h}_{i,1} \rangle \pmod{pq} \quad \text{for} \quad 1 \le i \le k,$$

then return the ciphertext $\overline{\text{ct}}' = (c_0, \ldots, c'_k) \in R_q^{k+1}$ where $c'_0 = \tau_j(c_0) + \lfloor p^{-1} \cdot c''_0 \rceil \pmod{q}$ and $c'_i = \lfloor p^{-1} \cdot c''_i \rceil \pmod{q}$ for $1 \le i \le k$.

B SECURITY PROOFS

B.1 Basic Scheme (Uni-encryption)

Let (n, q, χ, ψ) be the RLWE parameter generated in the setup phase. Given a plaintext $\mu \in R$, we define the distribution $\mathcal{D}_0 = \{(\mathbf{a}, \mathbf{b}, \mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2)\}$ over $R_q^{d\times 5}$ as follows:

(1) $\mathbf{a} \leftarrow U(R_q^d)$, $s \leftarrow \chi$, $e \leftarrow \psi$, and $\mathbf{b} = -s \cdot \mathbf{a} + \mathbf{e} \pmod{q}$,

(2) $\mathbf{d}_1 \leftarrow U(R_q^d)$, $r \leftarrow \chi$, $e_1 \leftarrow \psi$, and $\mathbf{d}_0 = -s \cdot \mathbf{d}_1 + \mathbf{e}_1 + r \cdot \mathbf{g}$ (mod q),

(3) $\mathbf{e}_2 \leftarrow \psi^d$ and $\mathbf{d}_2 = r \cdot \mathbf{a} + \mathbf{e}_2 + \mu \cdot \mathbf{g} \pmod{q}$,

which represents the tuple of CRS, public key, and uni-encryption of μ.

First, we consider the distribution $\mathcal{D}_1$ over $R_q^{d\times 5}$ which is obtained from $\mathcal{D}_0$ by modifying its definition (1) and (2) into

(1)' $\mathbf{a} \leftarrow U(R_q^d)$ and $\mathbf{b} \leftarrow U(R_q^d)$,

(2)' $\mathbf{d}_1 \leftarrow U(R_q^d)$ and $\mathbf{d}_0 \leftarrow U(R_q^d)$.

Since (3) is independent from the choice of $s \leftarrow \chi$, the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are computationally indistinguishable if the RLWE problem (of secret s) with parameter (n, q, χ, ψ) is hard.

In the next step, we change the definition (3) of $\mathcal{D}_1$ into

(3)' $\mathbf{d}_2 \leftarrow U(R_q^d)$,

and use the same RLWE assumption (of secret r) to show that it is computationally indistinguishable from $\mathcal{D}_1$.

Finally, $U(R_q^{d\times 5})$ is independent from the given plaintext μ, so we conclude that the uni-encryption scheme is IND-CPA secure.

B.2 MKHE Schemes

As mentioned earlier, our MKHE schemes have single-key encryption algorithms. A ciphertext is generated by adding an encoded plaintext to a random encryption of zero. Hence, it suffices to show that a random encryption of zero is computationally indistinguishable from a uniform random variable.

We consider the random variable (a, b, c_0, c_1) over R_q^4 defined by $a \leftarrow U(R_q)$, $b = -s \cdot a + e \pmod{q}$ for $s \leftarrow \chi$ and $e \leftarrow \psi$, and $(c_0, c_1) = v \cdot (b, a) + (e_0, e_1) \pmod{q}$ for $v \leftarrow \chi$ and $e_0, e_1 \leftarrow \psi$. It represents the tuple of a CRS, a public key, and an encryption of zero. First, we change the definition of b as $b \leftarrow U(R_q)$. It is computationally indistinguishable by the RLWE assumption with parameter (n, q, χ, ψ) (of secret s). Then, (b, c_0) and (a, c_1) can be viewed as two RLWE samples of secret v which are computationally indistinguishable from the uniform random variable $U(R_q^2)$ under the same RLWE assumption.

B.3 Special Modulus Variants

We set the ciphertext modulus as pq to design the special modulus variants of uni-encryption and relinearization algorithms. Therefore, we can show that special modulus variants of our MKHEs are IND-CPA secure under the RLWE assumption of parameter (n, pq, χ, ψ) by simply replacing q's in the previous security proofs by pq.

C NOISE ANALYSIS

We provide an average-case noise estimation on the variances of polynomial coefficients. In the following, we make a heuristic assumption that the coefficients of each polynomial behave like independent zero-mean random variables of the same variance. We denote by $\mathsf{Var}(a) = \mathsf{Var}(a_i)$ the variance of its coefficients for given random variable $a = \sum_i a_i \cdot X^i \in R$ over R. Hence, the product $c = a \cdot b$ of two polynomials will have the variance of $\mathsf{Var}(c) = n \cdot \mathsf{Var}(a) \cdot \mathsf{Var}(b)$. More generally, for a vector $\mathbf{a} \in R^d$ of random variables, we define $\mathsf{Var}(\mathbf{a}) = \frac{1}{d}\sum_{i=1}^{d} \mathsf{Var}(\mathbf{a}[i])$.

Specifically, we let $V_g = \mathsf{Var}(\mathbf{g}^{-1}(a))$ of a uniform random variable a over R_q to estimate the size of gadget decomposition. Recall that our implementation exploits the RNS-friendly decomposition $R_q \to \prod_i R_{p_i}, a \mapsto ([a]_{p_i})_i$ for distinct word-size primes of the same bit-size so that $d = \lceil \log q / \log p_i \rceil$ and $V_g \approx \frac{1}{12d}\sum_{i=1}^{d} p_i^2$. Finally, we assume that every ciphertext $\overline{\mathsf{ct}} \in R_q^{k+1}$ behaves as if it is a uniform random variable over R_q^{k+1}.

C.1 Relinearization

We first specify some distributions for detailed analysis. We set the key distribution χ and the distribution ψ as the uniform distribution over the set of binary polynomials and the Gaussian distribution of variance σ^2, respectively.

Method 1. We first analyze the conversion algorithm $\mathbf{K}_{i,j} \leftarrow \mathtt{Convert}(\mathbf{D}_i, \mathbf{b}_j)$. Let $\mathbf{D}_i = [\mathbf{d}_{i,0} | \mathbf{d}_{i,1} | \mathbf{d}_{i,2}]$ be an uni-encryption of $\mu_i \in R$ encrypted by the secret s_i and $(s_j, \mathbf{b}_j)$ a pair of the secret and public keys of the j-th party, i.e., $\mathbf{b}_j = -s_j \cdot \mathbf{a} + \mathbf{e}_j \pmod{q}$, $\mathbf{d}_{i,0} = -s_i \cdot \mathbf{d}_{i,1} + \mathbf{e}_{i,1} + r_i \cdot \mathbf{g} \pmod{q}$, and $\mathbf{d}_{i,2} = r_i \cdot \mathbf{a} + \mathbf{e}_{i,2} + \mu_i \cdot \mathbf{g}$ (mod q) for fresh errors $\mathbf{e}_j$, $\mathbf{e}_{i,1}$ and $\mathbf{e}_{i,2}$. We observe that

$$\mathbf{k}_{i,j,0} + s_i \cdot \mathbf{k}_{i,j,1} = \mathbf{M}_j \cdot (\mathbf{d}_{i,0} + s_i \cdot \mathbf{d}_{i,1}) = \mathbf{M}_j \mathbf{e}_{i,1} + r_i \mathbf{b}_j \pmod{q},$$
$$s_j \cdot \mathbf{k}_{i,j,2} = r_i s_j \cdot \mathbf{a} + s_j \cdot \mathbf{e}_{i,2} + \mu_i s_j \cdot \mathbf{g} \pmod{q},$$

and consequently $\mathbf{k}_{i,j,0} + s_i \cdot \mathbf{k}_{i,j,1} + s_j \cdot \mathbf{k}_{i,j,2} = (\mathbf{M}_j \mathbf{e}_{i,1} + r_i \mathbf{e}_j + s_j \mathbf{e}_{i,2}) + \mu_i s_j \mathbf{g} \pmod{q}$. Therefore, the noise $\mathbf{M}_j \mathbf{e}_{i,1} + r_i \mathbf{e} + s_j \mathbf{e}_{i,2} \in R^d$ of the output ciphertext has the variance of

$$V_{conv} = n\sigma^2 \cdot \left(d \cdot V_g + 1\right).$$

We now consider an extended ciphertext $\overline{\mathsf{ct}} = (c_{i,j})_{0 \le i,j \le k}$ and the output $\overline{\mathsf{ct}}' \leftarrow \mathtt{Relin}(\overline{\mathsf{ct}}; \{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k})$ of the relinearization procedure. As shown in Section 3.3.1, it satisfies that

$$\begin{aligned}
\langle \overline{\mathsf{ct}}', \overline{\mathsf{sk}} \rangle &= c_0' + \sum_{i=1}^{k} c_i' \cdot s_i \\
&= c_{0,0} + \sum_{i=1}^{k} (c_{0,i} + c_{i,0}) s_i + \sum_{i,j=1}^{k} \mathbf{g}^{-1}(c_{i,j}) \cdot \mathbf{K}_{i,j} \cdot (1, s_i, s_j) \pmod{q} \\
&= \langle \overline{\mathsf{ct}}, \overline{\mathsf{sk}} \otimes \overline{\mathsf{sk}} \rangle + \sum_{i,j=1}^{k} \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{e}_{i,j} \rangle \pmod{q},
\end{aligned}$$

where $\mathbf{e}_{i,j} = \mathbf{K}_{i,j} \cdot (1, s_i, s_j) - s_i s_j \cdot \mathbf{g} \pmod{q}$ denotes the error of $\mathbf{K}_{i,j}$. Hence, the relinearization error $e_{lin} = \sum_{i,j=1}^{k} \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{e}_{i,j} \rangle$ has the variance of

$$V_{lin} = k^2 \cdot nd \cdot V_g \cdot V_{conv} \approx k^2 \cdot n^2 d^2 \sigma^2 \cdot V_g^2.$$

This variance can be reduced to about half by eliminating the duplicated entries of $\overline{\mathsf{sk}} \otimes \overline{\mathsf{sk}}$ as explained in Section 6.1. We also note that the factor k^2 in the formula can be reduced down to $k_1 k_2$ if $\overline{\mathsf{ct}} = \overline{\mathsf{ct}}_1 \otimes \overline{\mathsf{ct}}_2$ is the tensor product of two sparse ciphertexts $\overline{\mathsf{ct}}_i$ corresponding to $k_i \le k$ secrets.

Method 2. Suppose that $\overline{\mathsf{ct}} = (c_{i,j})_{0 \le i,j \le k}$ is an extended ciphertext and let $\overline{\mathsf{ct}}' \leftarrow \mathtt{Relin}(\overline{\mathsf{ct}}; \{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le k})$ be the output of the relinearization procedure. As noted in Section 3.3.2, we have that

$$\mathbf{g}^{-1}(c'_{i,j}) \cdot [\mathbf{d}_{i,0}|\mathbf{d}_{i,1}] \cdot (1, s_i) = r_i \cdot c'_{i,j} + \langle \mathbf{g}^{-1}(c'_{i,j}), \mathbf{e}_{i,1}\rangle \pmod q,$$

and

$$\begin{aligned}\langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{d}_{i,2}\rangle \cdot s_j &= \langle \mathbf{g}^{-1}(c_{i,j}), r_i s_j \cdot \mathbf{a} + s_j \cdot \mathbf{e}_{i,2} + s_i s_j \cdot \mathbf{g}\rangle \\ &= \langle \mathbf{g}^{-1}(c_{i,j}), r_i \cdot (-\mathbf{b}_j + \mathbf{e}_j) + s_j \cdot \mathbf{e}_{i,2} + s_i s_j \cdot \mathbf{g}\rangle \\ &= -r_i \cdot c'_{i,j} + c_{i,j} \cdot s_i s_j + e_{i,j} \pmod q.\end{aligned}$$

where $e_{i,j} = \langle \mathbf{g}^{-1}(c_{i,j}), r_i \cdot \mathbf{e}_j + s_j \cdot \mathbf{e}_{i,2}\rangle \pmod q$. The variance of $e_{i,j}$ is $n^2 d \cdot \sigma^2 \cdot V_g$.

From the equation

$$\langle \overline{\mathrm{ct}}', \overline{\mathrm{sk}}\rangle = \langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}}\rangle + \sum_{i,j=1}^{k} \left(\langle \mathbf{g}^{-1}(c'_{i,j}), \mathbf{e}_{i,1}\rangle + e_{i,j}\right) \pmod q,$$

the variance of a relinearization error e_{lin} is obtained by

$$V_{lin} = k^2 \cdot (n^2 + n)d \cdot \sigma^2 \cdot V_g \approx k^2 \cdot n^2 d \cdot \sigma^2 \cdot V_g.$$

Special Modulus Variant of Method 2. In lines 3 ~ 6 of Alg. 3, we add $\mathbf{g}^{-1}(c'_{i,j}) \cdot [\mathbf{d}_{i,0}|\mathbf{d}_{i,1}]$ and $\langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{d}_{i,2}\rangle$ to the temporary ciphertext. We first note that $p \cdot c'_{i,j} = c''_{i,j} - [c''_{i,j}]_p \pmod{pq}$ and

$$\begin{aligned}&\mathbf{g}^{-1}(c'_{i,j}) \cdot (\mathbf{d}_{i,0} + s_i \cdot \mathbf{d}_{i,1}) \\ =& p r_i c'_{i,j} + \langle \mathbf{g}^{-1}(c'_{i,j}), \mathbf{e}_{i,1}\rangle \pmod{pq} \\ =& \langle \mathbf{g}^{-1}(c_{i,j}), r_i \mathbf{b}_j\rangle - r_i \cdot [c''_{i,j}]_p + \langle \mathbf{g}^{-1}(c'_{i,j}), \mathbf{e}_{i,1}\rangle \pmod{pq}.\end{aligned}$$

Meanwhile, the other term satisfies that

$$\begin{aligned}&\langle \mathbf{g}^{-1}(c_{i,j}), s_j \cdot \mathbf{d}_{i,2}\rangle \\ =& \langle \mathbf{g}^{-1}(c_{i,j}), -r_i \mathbf{b}_j + r_i \mathbf{e}_j + p s_i s_j \mathbf{g} + s_j \mathbf{e}_{i,2}\rangle \pmod{pq} \\ =& p c_{i,j} \cdot s_i s_j - \langle \mathbf{g}^{-1}(c'_{i,j}), r_i \mathbf{b}_j\rangle + \langle \mathbf{g}^{-1}(c_{i,j}), r_i \mathbf{e}_j + s_j \mathbf{e}_{i,2}\rangle \pmod{pq}.\end{aligned}$$

Consequently, the phase of the temporary ciphertext $(c''_i)_{0 \le i \le k}$ is increased by $p c_{i,j} \cdot s_i s_j + e_{i,j}$ for the error

$$e_{i,j} = -r_i \cdot [c''_{i,j}]_p + \langle \mathbf{g}^{-1}(c'_{i,j}), \mathbf{e}_{i,1}\rangle + \langle \mathbf{g}^{-1}(c_{i,j}), r_i \mathbf{e}_j + s_j \mathbf{e}_{i,2}\rangle,$$

whose variance is $\mathrm{Var}(e_{i,j}) = (n^2 + n)d \cdot V_g \cdot \sigma^2 + \frac{1}{24} n p^2$. We repeat it for all $1 \le i, j \le k$, so the temporary ciphertext will satisfy

$$c''_0 + \sum_{i=1}^{k} c''_i \cdot s_i = e + p \cdot \sum_{i,j=1}^{k} c_{i,j} \cdot s_i s_j \pmod{pq}$$

for the error $e = \sum_{i,j=1}^{k} e_{i,j}$ of varianc $\mathrm{Var}(e) = k^2 \cdot \mathrm{Var}(e_{i,j})$. After reducing its modulus down to q, we get the output ciphertext $\overline{\mathrm{ct}}' = (c'_i)_{0 \le i \le k}$ whose phase is

$$\langle \overline{\mathrm{ct}}', \overline{\mathrm{sk}}\rangle = \langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}}\rangle + p^{-1} \cdot e + e_{rd}$$

with an additional rounding error $e_{rd} = -p^{-1} \cdot \langle [(c''_i)_{0 \le i \le k}]_p, \overline{\mathrm{sk}}\rangle$ of variance $\frac{1}{24} kn + \frac{1}{12}$. The final relinearization error $e_{lin} = p^{-1} \cdot e + e_{rd}$ has the variance

$$\begin{aligned}V_{lin} &= p^{-2} \cdot k^2 (n^2 + n) d\sigma^2 \cdot V_g + \frac{1}{24} k^2 n + \frac{1}{24} kn + \frac{1}{12} \\ &\approx p^{-2} \cdot k^2 n^2 d \cdot \sigma^2 \cdot V_g + \frac{1}{24}(k^2 + k)n.\end{aligned}$$

C.2 Multi-Key BFV

Encryption. Let $\mathrm{ct} = (c_0, c_1) \in R_q^2$ be an encryption of $m \in R_t$ generated by the randomness $r \leftarrow \chi$ and $e_0, e_1 \leftarrow \psi$. Then we have

$$\begin{aligned}c_0 + c_1 \cdot s &= \Delta \cdot m + r \cdot (b + a \cdot s) + (e_0 + e_1 \cdot s) \pmod q \\ &= \Delta \cdot m + (r \cdot e + e_0 + e_1 \cdot s) \pmod q,\end{aligned}$$

where $e = \mathbf{e}[0]$ is a noise of public key. Therefore, the encryption noise $e_{enc} = r \cdot e + e_0 + e_1 \cdot s$ has the variance of

$$V_{enc} = \sigma^2 \cdot (1 + n) \approx \sigma^2 n.$$

Multiplication. Suppose that $\overline{\mathrm{ct}}_i$ is an encryption of m_i for $i = 1, 2$, i.e., $\langle \overline{\mathrm{ct}}_i, \overline{\mathrm{sk}}\rangle = q \cdot I_i + \Delta \cdot m_i + e_i$ for some I_i and e_i in R. The variance of $I_i = \left\lfloor \frac{1}{q} \langle \overline{\mathrm{ct}}_i, \overline{\mathrm{sk}}\rangle \right\rceil$ is computed by $\mathrm{Var}(I_i) \approx \frac{1}{12}\left(1 + \frac{1}{2} kn\right) \approx \frac{1}{24} kn$ since $\frac{1}{q} \cdot \overline{\mathrm{ct}}_i$ behaves like a uniform random variable over $\frac{1}{q} \cdot R_q^{k+1}$ whose variance is approximately equal to $\frac{1}{12}$.

The tensor product of the input ciphertexts satisfies

$$\begin{aligned}\langle \overline{\mathrm{ct}}_1 \otimes \overline{\mathrm{ct}}_2, \overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}}\rangle &= \langle \overline{\mathrm{ct}}_1, \overline{\mathrm{sk}}\rangle \cdot \langle \overline{\mathrm{ct}}_2, \overline{\mathrm{sk}}\rangle \\ &= \Delta^2 \cdot m_1 m_2 + q \cdot (I_1 e_2 + I_2 e_1) \\ &\quad + \Delta \cdot (m_1 e_2 + m_2 e_1) + e_1 e_2 \pmod{q \cdot \Delta},\end{aligned}$$

and consequently the ciphertext $\overline{\mathrm{ct}} = \lfloor (t/q) \cdot \overline{\mathrm{ct}}_1 \otimes \overline{\mathrm{ct}}_2 \rceil$ has the phase

$$\begin{aligned}\langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}}\rangle &= \Delta \cdot m_1 m_2 + (t \cdot (I_1 e_2 + I_2 e_1) \\ &\quad + (m_1 e_2 + m_2 e_1) + \Delta^{-1} \cdot e_1 e_2 + e_{rd})\end{aligned}$$

for the rounding error $e_{rd} = \langle (t/q) \cdot \overline{\mathrm{ct}}_1 \otimes \overline{\mathrm{ct}}_2 - \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}}\rangle$. Therefore, the multiplication error

$$e_{mul} = \left(t \cdot (I_1 e_2 + I_2 e_1) + (m_1 e_2 + m_2 e_1) + \Delta^{-1} \cdot e_1 e_2 + e_{rd}\right)$$

is dominated by the first term $t \cdot (I_1 e_2 + I_2 e_1)$ whose variance is

$$\begin{aligned}V_{mul} &= nt^2 \cdot (\mathrm{Var}(I_1) \cdot \mathrm{Var}(e_2) + \mathrm{Var}(I_2) \cdot \mathrm{Var}(e_1)) \\ &\approx \frac{1}{24} kn^2 t^2 \cdot (\mathrm{Var}(e_1) + \mathrm{Var}(e_2)).\end{aligned}$$

C.3 Multi-Key CKKS

Encryption. The CKKS scheme has the same encryption error as BFV.

Multiplication. For $i = 1, 2$, let $\mu_i = \langle \overline{\mathrm{ct}}_i, \overline{\mathrm{sk}}\rangle$ be the phase of an input ciphertext $\overline{\mathrm{ct}}_i$. Then $\overline{\mathrm{ct}} = \overline{\mathrm{ct}}'_1 \otimes \overline{\mathrm{ct}}'_2$ has the phase $\langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \otimes \overline{\mathrm{sk}}\rangle = \langle \overline{\mathrm{ct}}'_1, \overline{\mathrm{sk}}\rangle \cdot \langle \overline{\mathrm{ct}}'_2, \overline{\mathrm{sk}}\rangle = \mu_1 \cdot \mu_2 \pmod q$. Therefore, the output $\overline{\mathrm{ct}}' \leftarrow \mathtt{Relin}(\overline{\mathrm{ct}}; \overline{\mathrm{rlk}})$ satisfies that $\langle \overline{\mathrm{ct}}', \overline{\mathrm{sk}}\rangle = \mu_1 \mu_2 + e_{lin}$ for a relinearization error e_{lin} of variance V_{lin}. As explained above, this variance can be reduced down if $\overline{\mathrm{ct}}_i$ has some zero entries.

Rescaling. Let $\overline{\mathrm{ct}}' \leftarrow \mathtt{MK\text{-}CKKS.Rescale}(\overline{\mathrm{ct}}) = \left\lfloor p_\ell^{-1} \cdot \overline{\mathrm{ct}} \right\rceil \in R_{q_{\ell-1}}^{k+1}$ for $\overline{\mathrm{ct}} \in R_{q_\ell}^{k+1}$. Then, we have

$$\langle \overline{\mathrm{ct}}', \overline{\mathrm{sk}}\rangle = p_\ell^{-1} \cdot [\langle \overline{\mathrm{ct}}, \overline{\mathrm{sk}}\rangle]_{q_\ell} + e_{rd} \pmod{q_{\ell-1}}$$

for the rescaling error $e_{rd} = \left\langle \left\lfloor p_\ell^{-1} \cdot \overline{\mathrm{ct}} \right\rceil - p_\ell^{-1} \cdot \overline{\mathrm{ct}}, \overline{\mathrm{sk}} \right\rangle$ from rounding. Note that each component of $\left\lfloor p_\ell^{-1} \cdot \overline{\mathrm{ct}} \right\rceil - p_\ell^{-1} \cdot \overline{\mathrm{ct}}$ behaves like a

Algorithm 4 Encryption of an image

```
Input: Image $\mathbf{X} \in \mathbb{R}^{28\times 28}$, public key of the image provider $\mathbf{b}_D$.
Output: ct.$X$.
for $0 \le i, j < 2$ do
    $\mathbf{X}'_{i,j} \leftarrow (x_{2k+i,2l+j})_{0\le k,l\le 13} \in \mathbb{R}^{14\times 14}$
    $\overline{\mathbf{X}}_{i,j} \leftarrow (\mathbf{X}'_{i,j}; \ldots ; \mathbf{X}'_{i,j}) \in \mathbb{R}^{70\times 14}$
    $\mathbf{v}_{i,j} \leftarrow \texttt{PadZeros}(\texttt{vec}(\bar{\mathbf{X}}_{i,j}), 2^{10} - 70 \cdot 14) \in \mathbb{R}^{1024}$
end for
$\mathbf{v} \leftarrow (\mathbf{v}_{0,0}|\mathbf{v}_{0,1}|\mathbf{v}_{1,0}|\mathbf{v}_{1,1}) \in \mathbb{R}^{4096}$
$\overline{\mathbf{v}} \leftarrow (\mathbf{v}|\mathbf{v}) \in \mathbb{R}^{8192}$
ct.$X \leftarrow \texttt{MK-CKKS.Enc}(\overline{\mathbf{v}}; \mathbf{b}_D, \mathbf{a})$
```

Algorithm 5 Encryption of multiple channels

```
Input: $\{\mathrm{Y}^{(k)} = (y^{(k)}_{i,j}) \in \mathbb{R}^{4\times 4}\}_{0\le k<5}$, public key of the model
    provider $\mathbf{b}_M$.
Output: $\{\mathsf{ct}.Y_l\}_{0\le l<4}$.
for $0 \le i, j < 4$ do
    $\mathbf{v}_{i,j} \leftarrow \varnothing$                                  ▷ Null string
    for $0 \le k < 5$ do
        $\mathrm{Y}_{k,i,j} \leftarrow (y^{(k)}_{i,j}, \ldots , y^{(k)}_{i,j}) \in \mathbb{R}^{13\times 13}$
        $\mathrm{Y}'_{k,i,j} \leftarrow \texttt{PadZeros}(\mathrm{Y}_{k,i,j}, (1, 1), (R, B)) \in \mathbb{R}^{14\times 14}$
        $\mathbf{v}_{i,j} \leftarrow (\mathbf{v}_{i,j}|\texttt{vec}(\mathrm{Y}'_{k,i,j}))$
    end for
    $\mathbf{v}_{i,j} \leftarrow \texttt{PadZeros}(\mathbf{v}_{i,j}, 2^{10} - 5 \cdot 14 \cdot 14) \in \mathbb{R}^{1024}$
end for
for $0 \le i, j < 2$ do
    $\mathbf{v}_{2i+j} \leftarrow (\mathbf{v}_{2i,2j}|\mathbf{v}_{2i,2j+1}|\mathbf{v}_{2i+1,2j}|\mathbf{v}_{2i+1,2j+1}) \in \mathbb{R}^{4096}$
    $\overline{\mathbf{v}}_{2i+j} \leftarrow (\mathbf{v}_{2i+j}|\mathbf{v}_{2i+j}) \in \mathbb{R}^{8192}$
    ct.$Y_{2i+j} \leftarrow \texttt{MK-CKKS.Enc}(\overline{\mathbf{v}}_{2i+j}; \mathbf{b}_M, \mathbf{a})$
end for
```

uniform random variable on $p_\ell^{-1} \cdot R_{p_\ell}$ whose variance is $\frac{1}{12}$. Therefore, the rescaling error has the variance of

$$V_{res} = \frac{1}{12}\left(1 + \frac{1}{2}kn\right) \approx \frac{1}{24}kn.$$

D HOMOMORPHIC EVALUATION OF CNN

For a $d_1 \times d$ matrix $\mathbf{A}_1$ and a $d_2 \times d$ matrix $\mathbf{A}_2$, $(\mathbf{A}_1; \mathbf{A}_2)$ denotes the $(d_1 + d_2) \times d$ matrix obtained by concatenating two matrices in a vertical direction. If two matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ have the same number of rows, $(\mathbf{A}_1|\mathbf{A}_2)$ denotes a matrix formed by horizontal concatenation. As defined in [34], there is a row ordering encoding map to transform a vector of dimension $n = d^2$ into a matrix in $\mathbb{R}^{d\times d}$. For a vector $\mathbf{a} = (a_k)_{0\le k<n}$, we define the encoding map $\texttt{mat} : \mathbb{R}^n \to \mathbb{R}^{d\times d}$ by $\texttt{mat} : \mathbf{a} \mapsto \mathbf{A} = (a_{d\cdot i+j})_{0\le i,j<d}$, i.e., $\mathbf{a}$ is the concatenation of row vectors of $\mathbf{A}$. Let vec denote its inverse mapping. We use $\mathbf{A}_{i,\ell_1:\ell_2}$ to denote a vector of length $(\ell_2 - \ell_1)$ formed by extracting from the ℓ_1-th entry to the $(\ell_2 - 1)$-th entry of the i-th row of $\mathbf{A}$. $\texttt{PadZeros}(\mathbf{A}, v, dir)$ pads a matrix $\mathbf{A}$ with zeros in the direction specified by dir where v is a vector of non-negative integers that specifies both the amount of padding to add and the dimension along which to add it. For example, direction can be specified as one of the following values: L (left), R (right), U (upper), and B (bottom). For simplicity, when $\mathbf{A}$ is given as a vector and we want to pad zeros to the right side of the vector, we will omit the direction in the notation.

Algorithm 6 Encryption of weights & biases of FC layers

```
Input: $\mathbf{W} \in \mathbb{R}^{64\times 845}$, $\mathbf{z}^{(1)} \in \mathbb{R}^{64}$, $\mathbf{U} \in \mathbb{R}^{10\times 64}$, $\mathbf{z}^{(2)} \in \mathbb{R}^{10}$, public
    key of the model provider $\mathbf{b}_M$.
Output: $\{\mathsf{ct}.W_k\}_{0\le k<8}$, ct.$z_1$, ct.$U$, ct.$z_2$.
    [Encryption of weight of FC-1]
$\mathbf{W}' \leftarrow \varnothing$
for $0 \le i < 64$ do
    $\mathbf{w} \leftarrow \varnothing$
    for $0 \le j < 5$ do
        $\mathbf{W}_j \leftarrow \texttt{mat}(\mathbf{W}_{i,13^2\cdot j:13^2\cdot(j+1)}) \in \mathbb{R}^{13\times 13}$
        $\mathbf{W}'_j \leftarrow \texttt{PadZeros}(\mathbf{W}_j, (1, 1), (R, B)) \in \mathbb{R}^{14\times 14}$
        $\mathbf{w} \leftarrow (\mathbf{w}|\texttt{vec}(\mathbf{W}'_j))$
    end for
    $\mathbf{w} \leftarrow \texttt{PadZeros}(\mathbf{w}, 2^{10} - 5 \cdot 14 \cdot 14) \in \mathbb{R}^{1024}$
    $\mathbf{W}' \leftarrow (\mathbf{W}'; \mathbf{w})$
end for
for $0 \le k < 8$ do
    $\mathbf{v} \leftarrow \varnothing$
    for $0 \le i < 64$ do
        $\mathbf{v} \leftarrow \left(\mathbf{v} \mid \mathbf{W}'_{i,2^7\cdot((i+k)\%8):2^7\cdot((i+k+1)\%8)}\right)$
    end for
    ct.$W_k \leftarrow \texttt{MK-CKKS.Enc}(\mathbf{v}; \mathbf{b}_M, \mathbf{a})$
end for
    [Encryptions of bias of FC-1/weight & bias of FC-2]
$\mathbf{v}_1 \leftarrow \varnothing, \mathbf{u} \leftarrow \varnothing, \mathbf{v}_2 \leftarrow \varnothing$
for $0 \le i < 64$ do
    $\mathbf{v}_1 \leftarrow (\mathbf{v}_1|\texttt{PadZeros}(z^{(1)}_i, 2^7 - 1))$   ▷ $z^{(1)}_i$ : $i$-th entry of $\mathbf{z}^{(1)}$
    $\mathbf{u} \leftarrow (\mathbf{u}|\texttt{PadZeros}(\mathbf{U}^t_i, 2^7 - 10))$        ▷ $\mathbf{U}_i$ : $i$-th column of $\mathbf{U}$
    $\mathbf{v}_2 \leftarrow (\mathbf{v}_2|\texttt{PadZeros}(\mathbf{z}^{(2)}, 2^7 - 10))$
end for
ct.$z_1 \leftarrow \texttt{MK-CKKS.Enc}(\mathbf{v}_1; \mathbf{b}_M, \mathbf{a})$
ct.$U \leftarrow \texttt{MK-CKKS.Enc}(\mathbf{u}; \mathbf{b}_M, \mathbf{a})$
ct.$z_2 \leftarrow \texttt{MK-CKKS.Enc}(\mathbf{v}_2; \mathbf{b}_M, \mathbf{a})$
```

D.1 Encryption of Data and Trained Model

Alg. 4 takes as the input an image $\mathbf{X} = (x_{i,j}) \in \mathbb{R}^{28\times 28}$ and the public key $\mathbf{b}_D$ of the image provider. Alg. 5 takes the multiple channels of the convolutional layer from the trained model as inputs and generates their encryptions using the public key of the model provider $\mathbf{b}_M$. Alg. 6 takes as the inputs the weights and biases of the FC layers and outputs their encryptions using the public key.

D.2 Homomorphic Evaluation of CNN

We start with a useful aggregation operation across plaintext slots from the literature [31, 38]. This algorithm is referred as `AllSum`, which is parameterized by integers φ and α. See Alg. 7 for its description. We denote by $\texttt{MKHE.Rot}(\mathsf{ct}; k)$ a multi-key rotation which transforms an input ciphertext ct into a new ciphertext that encrypts a shifted plaintext vector by k of the original message. Given a ciphertext ct representing a plaintext vector $\mathbf{m} = (m_0, \ldots, m_{\ell-1}) \in \mathbb{R}^{\ell}$ for $\ell = \alpha \cdot \varphi$, the `AllSum` algorithm outputs a ciphertext $\mathrm{ct}_{\mathrm{out}}$ encrypting α copies of the vector $(\sum_{j=0}^{\alpha-1} m_{j\cdot\varphi}, \sum_{j=0}^{\alpha-1} m_{j\cdot\varphi+1}, \ldots, \sum_{j=0}^{\alpha-1} m_{(j+1)\cdot\varphi-1}) \in \mathbb{R}^{\varphi}$. This can be implemented using rotations and additions.

Matrix-Vector Multiplication. Assume that n_i is smaller than the number of plaintext slots n_s and the input $n_o \times n_i$ matrix $\mathbf{W}$ is split into $1 \times (n_i/\ell)$ sized sub-matrices, denoted by $\mathbf{w}_{i,j}$ for $0 \le i < n_o$ and $0 \le j < \ell$. Then, we can pack $(\ell \cdot n_s)/n_i$ many different sub-matrices into a single ciphertext, and $n_c = (n_s/n_i)$ copies of the input vector $\mathbf{v}$ in a single ciphertext. For example, consider the first $(\ell \cdot n_c)$ rows of $\mathbf{W}$. We encode n_c many the first diagonal vectors of the matrix into a plaintext vector, i.e.,

$$\big((\mathbf{w}_{0,0}|\mathbf{w}_{1,1}|\ldots|\mathbf{w}_{\ell-1,\ell-1})|(\mathbf{w}_{\ell,0}|\mathbf{w}_{\ell+1,1}|\ldots|\mathbf{w}_{2\ell-1,\ell-1})|\ldots$$
$$(\mathbf{w}_{\ell\cdot(n_c-1),0}|\mathbf{w}_{\ell\cdot(n_c-1)+1,1}|\ldots|\mathbf{w}_{\ell\cdot(n_c-1)+\ell-1,\ell-1})\big) \in \mathbb{R}^{n_s}.$$

As such, each extended diagonal vectors are encrypted in a single ciphertext and these ℓ many ciphertexts are multiplied with ℓ rotations of the encrypted vector $\mathbf{v}$. Then we add them together similar to the original diagonal method and the output ciphertext represents (n_i/ℓ)-sized $(\ell \cdot n_c)$ chunks, each of which contains partial sums of ℓ entries of n_i inputs. Finally, we can accumulate these using a rotate-and-sum algorithm with $\log(n_i/\ell)$ rotations, yielding in a ciphertext that contains the first $(\ell \cdot n_c)$ entries of $\mathbf{W}\mathbf{v} \in \mathbb{R}^{n_o}$. We repeat these procedures $n_o/(\ell \cdot n_c)$ times. In the end, we get $n_o/(\ell \cdot n_c)$ ciphertexts, each containing $(\ell \cdot n_c)$ entries of the result. The computational cost is $\ell \cdot (n_o/(\ell \cdot n_c)) = (n_o \cdot n_i)/n_s$ homomorphic multiplications, $(\ell - 1)$ rotations of the encrypted vector $\mathbf{v}$, and $(n_o/(\ell \cdot n_c)) \cdot \log(n_i/\ell)$ rotation on distinct ciphertexts.

Evaluation Strategy of CNN. Alg. 8 provides an explicit description of homomorphic evaluation of CNN. We denote by $\mathtt{SMult}(\mathrm{ct}, \mathbf{u})$ the multiplication of ct with a scalar $\mathbf{u}$. For simplicity, we omit the evaluation/public keys $\{(\mathbf{D}_i, \mathbf{b}_i)\}_{1 \le i \le 2}$ in homomorphic multiplication process.

Algorithm 7 `AllSum`$(\mathrm{ct}, \psi, \alpha)$

```
Input: ct, input ciphertext, the unit initial amount by which the
    ciphertext shifts $\psi$, the number of summands $\alpha > 1$
Output: $\mathrm{ct}_{\mathrm{out}}$
$\mathrm{ct}_{\mathrm{out}} \leftarrow \mathrm{ct}$
for $i = 0, 1, \ldots, \log\alpha - 1$ do
    $\mathrm{ct}_{\mathrm{out}} \leftarrow \mathtt{MKHE.Add}(\mathrm{ct}_{\mathrm{out}}, \mathtt{MKHE.Rot}(\mathrm{ct}_{\mathrm{out}}; \psi \cdot 2^i))$
end for
```

Algorithm 8 Homomorphic evaluation of CNN

```
Input: $\mathrm{ct}.X$, $\{\mathrm{ct}.Y_l\}_{0 \le l < 4}$, $\{\mathrm{ct}.W_k\}_{0 \le k < 8}$, $\mathrm{ct}.z_1$, $\mathrm{ct}.U$, $\mathrm{ct}.z_2$.
Output: $\mathrm{ct}_{\mathrm{out}}$.
    [Convolutional layer]
$\mathrm{ct}_0 \leftarrow \mathtt{MK\text{-}CKKS.Mult}(\mathrm{ct}.X, \mathrm{ct}.Y_0)$    ▷ Simple convolutions
for $1 \le l < 4$ do
    $\mathrm{ct}.X_l \leftarrow \mathtt{MK\text{-}CKKS.Rot}(\mathrm{ct}.X; 14i + j)$    ▷ $i = \lfloor l/2 \rfloor, j = (l\%2)$
    $\mathrm{ct} \leftarrow \mathtt{MK\text{-}CKKS.Mult}(\mathrm{ct}.X_l, \mathrm{ct}.Y_l)$
    $\mathrm{ct}_0 \leftarrow \mathtt{MK\text{-}CKKS.Add}(\mathrm{ct}_0, \mathrm{ct})$
end for
$\mathrm{ct}_0 \leftarrow \mathtt{MK\text{-}CKKS.Rescale}(\mathrm{ct}_0)$
$\mathrm{ct}_0 \leftarrow \mathtt{AllSum}(\mathrm{ct}_0, 1024, 4)$    ▷ Sum over multiple channels
    [$1^{\mathrm{st}}$ square layer]
$\mathrm{ct}_1 \leftarrow \mathtt{MK\text{-}CKKS.Rescale}(\mathtt{MK\text{-}CKKS.Mult}(\mathrm{ct}_0, \mathrm{ct}_0))$
    [FC-1 layer]
$\mathrm{ct}_2 \leftarrow \mathtt{MK\text{-}CKKS.Mult}(\mathrm{ct}_1, \mathrm{ct}.W_0)$
for $1 \le l < 8$ do
    $\mathrm{ct} \leftarrow \mathtt{MK\text{-}CKKS.Rot}(\mathrm{ct}_1; 2^7 \cdot l)$
    $\mathrm{ct} \leftarrow \mathtt{MK\text{-}CKKS.Mult}(\mathrm{ct}, \mathrm{ct}.W_l)$
    $\mathrm{ct}_2 \leftarrow \mathtt{MK\text{-}CKKS.Add}(\mathrm{ct}_2, \mathrm{ct})$
end for
$\mathrm{ct}_2 \leftarrow \mathtt{MK\text{-}CKKS.Rescale}(\mathrm{ct}_2)$
$\mathrm{ct}_2 \leftarrow \mathtt{AllSum}(\mathrm{ct}_2, 1, 64)$
$\mathbf{u} \leftarrow \mathtt{PadZeros}((1), 127) \in \mathbb{R}^{128}$
$\bar{\mathbf{u}} \leftarrow (\mathbf{u}|\mathbf{u}|\ldots|\mathbf{u}) \in \mathbb{R}^{8192}$
$\mathrm{ct}_2 \leftarrow \mathtt{MK\text{-}CKKS.SMult}(\mathrm{ct}_2, \bar{\mathbf{u}})$    ▷ Multiplicative masking
$\mathrm{ct}_2 \leftarrow \mathtt{MK\text{-}CKKS.Rescale}(\mathrm{ct}_2)$
$\mathrm{ct}_2 \leftarrow \mathtt{MK\text{-}CKKS.Add}(\mathrm{ct}_2, \mathrm{ct}.z_1)$
    [$2^{\mathrm{nd}}$ square layer]
$\mathrm{ct}_3 \leftarrow \mathtt{MK\text{-}CKKS.Rescale}(\mathtt{MK\text{-}CKKS.Mult}(\mathrm{ct}_2, \mathrm{ct}_2))$
    [FC-2 layer]
$\mathrm{ct}_4 \leftarrow \mathtt{AllSum}(\mathrm{ct}_3, -1, 16)$
$\mathrm{ct}_4 \leftarrow \mathtt{MK\text{-}CKKS.Rescale}(\mathtt{MK\text{-}CKKS.Mult}(\mathrm{ct}_4, \mathrm{ct}.U))$
$\mathrm{ct}_4 \leftarrow \mathtt{AllSum}(\mathrm{ct}_4, 128, 64)$
$\mathrm{ct}_{\mathrm{out}} \leftarrow \mathtt{MK\text{-}CKKS.Add}(\mathrm{ct}_4, \mathrm{ct}.z_2)$
```