Asymptotically Faster Multi-Key Homomorphic Encryption from Homomorphic Gadget Decomposition

Taechan Kim[1], Hyesun Kwak[2], Dongwon Lee[2], Jinyeong Seo[2], and Yongsoo Song[2]

[1] Samsung Research, South Korea
`taechan.kim@samsung.com`
[2] Seoul National University, South Korea
`{hskwak, dongwonlee95, jinyeong.seo, y.song}@snu.ac.kr`

Abstract. Homomorphic Encryption (HE) is a cryptosytem that allows us to perform an arbitrary computation on encrypted data. The standard HE, however, has a disadvantage in that the authority is concentrated in the secret key owner as the computation can be performed only on ciphertexts under the same key. In order to overcome this problem, research is underway on Multi-Key Homomorphic Encryption (MKHE), which enables operations between encrypted data possibly under different keys. Despite its strength to cover privacy of multiple parties, the existing MKHE schemes suffer from poor performance that the multiplication cost grows at least quadratically with the number of parties involved.
In this paper, we propose a new notion of the gadget decomposition, which enables arithmetic operations to be performed on the decomposed vectors with guarantee of functionality and noise bound. We redesign the multi-key multiplication algorithm of Chen et al. (ACM CCS 2019) using the homomorphic property of gadget decomposition and thereby reduce the complexity significantly from quadratic to linear in the number of parties involved. Finally, we implement our MKHE schemes and provide benchmarks which outperform the previous results.

1 Introduction

Homomorphic encryption (HE) is a cryptosystem that enables computation on encrypted messages without decrypting them first. It has been a long-standing open problem to construct a fully HE (which supports arbitrary computations) until Gentry's breakthrough [18]. Since then, there have been made lots of progress on construction of HE, to name a few, BFV [5, 16], GSW [20], BGV [6], TFHE [13], and CKKS [12]. HE inherently supports an on-the-fly secure computation, *i.e.,* no need for data owners to be online during the computation since the whole evaluation process can be done by a public server. Such characteristic is especially well-suited for the cases such as cloud-based environments.

However, the standard HE is less amenable to the multi-party setting. For instance, when there are multiple data sources, the standard HE causes an authority concentration issue. If one considers directly applying standard single-key HE, data should be encrypted under the same encryption key. In this case, the person who has the corresponding secret key gains access to all data and thus the privacy of data owners may be exposed. In the last decade, there have been several attempts to extend the functionality of HE to deal with the aforementioned issues. Threshold HE [4], multi-party HE [2, 26], and multi-key HE (MKHE) [25, 14, 27, 29, 8, 9] are some examples which overcome the limitation of single-key HE by distributing the decryption authority among multiple parties so that no single party has access to plain data. Moreover, these primitives can be naturally extended to build multi-party protocols that keeps the advantages of HE.

We focus on MKHE which enjoys considerable advantages in terms of interaction and flexibility. To be precise, an MKHE scheme allows a participant to generate secret and public keys which can be used to encrypt data without any knowledge of other parties. It supports homomorphic operations of ciphertexts under different keys so that all computation can be done by a public cloud. Moreover, recent MKHE schemes are fully dynamic, *i.e.,* the computational task does not have to be pre-determined but an arbitrary circuit can be evaluated over any ciphertexts on the fly, and new users (ciphertexts) can be introduced into the computation anytime. Therefore, one can build a secure multi-party computation(MPC) protocol on top of MKHE which inherits this dynamic nature [27].

While MKHE enables flexible and dynamic setup, it is technically challenging, compared to other HE variants, to design an efficient MKHE scheme due to the strong requirement on the functionality. After López-Alt at al. [25] presented the first MKHE scheme based on NTRU, there have been several studies [14, 27, 29, 7, 8, 10, 9] which convert the existing single-key HE schemes into multi-key versions, but the poor performance of MKHE still remains a major bottleneck. Earlier schemes were relatively impractical, but recent researches [8, 9] demonstrated viable instantiations with implementation results which are currently the best-performing MKHE schemes in terms of both asymptotic and concrete complexity.

This paper is an extension of the work by Chen, Dai, Kim and Song (CDKS) [9] which presents multi-key variants of the RLWE-based BFV and CKKS schemes supporting homomorphic operations in a SIMD manner. In CDKS, a multi-key ciphertext is of the form $(c_0, c_1, \ldots, c_n)$ where n is the number of associated parties and c_i's are elements of the base polynomial ring. It can be decrypted by the secret keys $s_1, \ldots, s_n$ of n participants so that $c_0 + c_1 \cdot s_1 + \cdots + c_n \cdot s_n$ is a randomized encoding of the plaintext. The most expensive operation is homomorphic multiplication which consists of two steps: tensor product and subsequent relinearization. For given encryptions $(c_i)_{0\le i\le n}$ and $(c'_i)_{0\le j\le n}$ of m and m', respectively, it first computes their product $(c_{i,j} := c_i \cdot c'_j)_{0\le i,j\le n}$ which can be viewed as a valid encryption of mm' decryptable by $s_i \cdot s_j$. Then, the relinearization procedure is followed that converts $(c_{i,j})_{0\le i,j\le n}$ back to the standard form with linear decryption structure. The total complexity of relinearization grows quadratically with n since the process should be repeated on $c_{i,j}$ for all $1 \le i, j \le n$.

1.1 Our Contributions

In this paper, we design new multi-key BFV and CKKS schemes with better performance by modifying the construction of CDKS. Let us give a technical overview on the previous method to explain our idea. The *gadget toolkit* [17] is a well-known technique in the construction of HE schemes which can be used to reduce the noise growth from homomorphic operations. A gadget toolkit over a modulus Q consists of a fixed *gadget vector* $\mathbf{g}$ and a *gadget decomposition* h which transforms an element a into a short vector $h(a)$ such that $\langle h(a), \mathbf{g}\rangle = a \pmod{Q}$.[3] The relinearization algorithm of CDKS operates the ciphertext components $c_{i,j}$ with the public keys, which also involves the computation of gadget decompositions $h(c_{i,j})$ for all $1 \le i, j \le n$ yielding $O(n^2)$ complexity in total.

To avoid the expensive computation of $h(c_{i,j})$, we define a new notion of *homomorphic gadget decomposition*. We say that a gadget decomposition is homomorphic if it supports the computation over decomposed vectors. In other words, we can perform arithmetic operations over the gadget decompositions $h(a), h(b)$ of any elements a, b so that $h(a) + h(b)$ and $h(a) \odot h(b)$ satisfy $\langle h(a) + h(b), \mathbf{g}\rangle = a + b \pmod{Q}$ and $\langle h(a) \odot h(b), \mathbf{g}\rangle = ab \pmod{Q}$ where $\odot$ denotes the component-wise product of vectors. Hence $h(a) + h(b)$ and $h(a) \odot h(b)$ can be considered valid decompositions of $a + b$ and ab, respectively.

In our MKHE construction, we first take advantage of homomorphic gadget decomposition and replace the term $h(c_{i,j})$ by $h(c_i) \odot h(c'_j)$. As a result, instead of repeating n^2 gadget decompositions for all pairs (i, j), we compute $h(c_i)$ and $h(c'_j)$ separately for $1 \le i, j \le n$ and combine them to represent a valid decomposition of $c_i \cdot c'_j$. Moreover, we depart from the conventional multiplication strategy based on tensor product and relinearization. Instead of computing $h(c_i) \odot h(c'_j)$ independently, we merge two steps and refactor the whole multiplication algorithm so that each ciphertext can be pre-processed before being multiplied to another ciphertext. As a result, we reduce the complexity of n-key homomorphic multiplication from $O(n^2)$ down to $O(n)$ operations which we believe is asymptotically optimal.

While our idea is directly applicable to design an efficient multi-key CKKS scheme, there still remains an issue for BFV. The tensor product and relinearization procedures are proceeded over different algebraic spaces in BFV, and such inconsistency inhibits applying our method. We resolve this issue by tweaking the public key structure so that the whole computation can be performed in the same ring. In addition, we present another implementation-friendly variant of our multi-key BFV scheme which requires no multi-precision arithmetic.

Finally, we implement our MKHE schemes and provide some benchmarks. We measure the performance for $n = 2, 4, \ldots, 64$ parties and the experimental results show that our construction rapidly outperforms the CDKS scheme [9] as n increases.

[3] The bit-decomposition is a typical example of gadget decomposition.

1.2 Related Works

As mentioned above, there are several directions to generalize HE. For example, threshold HE [4] also distributes the authority and provides t-out-of-n access structure, but the key generation is done by a trusted third party. On the other hand, multi-party HE [2, 26, 28] is another HE primitive where multiple parties jointly generate a shared public key while the corresponding secret is additively shared among the parties. Although MPHE has advantages in performance, it does not have the flexibility of MKHE in the sense that the set of parties should be fixed at the setup phase and the same key should be used for encryption.

MKHE schemes can be classified with respect to the underlying HE scheme. Early studies [14, 27, 29] constructed MKHE schemes from GSW [20], but they require huge space and time complexity. Brakerski and Perlman [7] designed an MKHE scheme from LWE with quasi-linear expansion rate, but its concrete performance was not clearly understood. A follow-up study was conducted by Chen, Chillotti and Song [8] who presented a multi-key variant of TFHE and demonstrated the first implementation result. On the other hand, there has been another line of work [10, 9] constructing multi-key variants of batch HE schemes such as BGV, BFV and CKKS. One common problem of the previous MKHE constructions is that they rely on the CRS assumption. Recently, Ananth et al. [1] constructed the first MKHE scheme in the plain model by combining the oblivious transfer protocol, MKHE with trusted setup, and MKHE in the plain model with interactive decryption.

2 Background

2.1 Notation

Let N be a power of two and Q be an integer. We denote by $R = \mathbb{Z}[X]/(X^N + 1)$ the ring of integers of the $(2N)$-th cyclotomic field and $R_Q = \mathbb{Z}_Q[X]/(X^N + 1)$ the residue ring of R modulo Q. We represent an element $a = \sum_{0 \le i < n} a_i \cdot X^i \in R_q$ by the vector of its coefficients $(a_0, \ldots, a_{n-1}) \in \mathbb{Z}_q^n$. For an integer q, we use $\mathbb{Z} \cap (-q/2, q/2]$ as a representative of $\mathbb{Z}_q$, and denote by $[a]_q$ the reduction of a modulo q. For a polynomial a in R or R_q, we define $\|a\|_\infty$ as the ℓ^∞-norm of its coefficient vector.

Throughout the paper, we write $x \leftarrow D$ to represent that x is sampled from the distribution D. We denote by $\mathcal{U}(S)$ the uniform distribution over a finite set S. For $\sigma > 0$, we denote by D_σ a distribution over R sampling N coefficients independently from the discrete Gaussian distribution of variance σ^2, and B_σ an (overwhelming probability) upper bound of D_σ with respect to the infinite norm.

2.2 Ring Learning with Errors

The Ring Learning with Errors (RLWE) assumption guarantees strong security of RLWE-based cryptosystems. Given the parameters (N, Q, χ, σ), consider the polynomial number of samples $(a_i, b_i) \in R_Q^2$ where $a_i \leftarrow \mathcal{U}(R_Q)$, $b_i = s \cdot a_i + e_i \pmod{Q}$ and $e_i \leftarrow D_\sigma$ for a fixed $s \leftarrow \chi$. The RLWE assumption states that the distribution of RLWE samples (a_i, b_i) is computationally indistinguishable from $\mathcal{U}(R_q^2)$. In this paper, we assume that the secret key s has ternary coefficients in $\{\pm 1, 0\}$.

2.3 Multi-Key Homomorphic Encryption

A multi-key homomorphic encryption (MKHE) is an encryption scheme which enables computation on encrypted data whose secret key may not be identical. Remark that in plain HE scheme, inputs should have the identical secret key to perform homomorphic operations. However, in MKHE scheme, inputs need not to have identical secret key, hence one can think of MKHE scheme as a superset of HE scheme. MKHE scheme consists of five PPT algorithms (`Setup`, `KeyGen`, `Enc`, `Eval`, `Dec`).

– **Setup:** $pp \leftarrow \texttt{MKHE.Setup}(1^\lambda)$. Given the security parameter λ, it returns the public parameter set pp.

- **Key Generation:** $\{\mathsf{sk}_i, \mathsf{pk}_i\}_{i\in I} \leftarrow \texttt{MKHE.KeyGen}(pp, I)$. Each party $i \in I$ initially holds pp and outputs the secret key sk_i and the public key pk_i.
- **Encryption:** $\overline{\mathsf{ct}} \leftarrow \texttt{MKHE.Enc}(\mu; \mathsf{pk}_i)$. A party i encrypts its plaintext μ in the message space $\mathcal{M}$ and outputs the ciphertext $\overline{\mathsf{ct}}$.
- **Evaluation:** $\overline{\mathsf{ct}} \leftarrow \texttt{MKHE.Eval}(\mathcal{C}, \overline{\mathsf{ct}}_1, \ldots, \overline{\mathsf{ct}}_k; \mathsf{pk}_1, \ldots, \mathsf{pk}_l)$. Given a circuit $\mathcal{C}$ and ciphertexts $\overline{\mathsf{ct}}_1, \ldots, \overline{\mathsf{ct}}_k$ with the corresponding public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_l$, it returns a ciphertext $\overline{\mathsf{ct}}$. We assume for convenience that the reference to the associated parties is contained in the output ciphertext.
- **Decryption:** $\mu \leftarrow \texttt{MKHE.Dec}(\overline{\mathsf{ct}}; \mathsf{sk}_1, \ldots, \mathsf{sk}_k)$. Given a ciphertext $\overline{\mathsf{ct}}$ and the corresponding secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_k$, it outputs a plaintext μ.

Note that it requires all the secret keys to decrypt a ciphertext. However, in practice, there can be an authority issue if a specific party holds other parties' secret keys. We can solve this issue with a distributed decryption which is a protocol that multiple key owners jointly decrypt the ciphertext. In the protocol, each party partially decrypts the ciphertext using their own secret and recover the message by merging partial decryptions of all parties. More details about distributed decryption are described in [27, 9].

A semantic security of MKHE is achieved if following distributions are computationally indistinguishable for encryption of any two messages μ_1 and μ_2:

$$(pp, \mathsf{pk}_i, \texttt{MKHE.Enc}(\mu_1, \mathsf{pk}_i)) \overset{\text{comp}}{\approx} (pp, \mathsf{pk}_i, \texttt{MKHE.Enc}(\mu_2, \mathsf{pk}_i))$$

where $pp \leftarrow \texttt{MKHE.Setup}(1^\lambda)$ and $\{\mathsf{sk}_i, \mathsf{pk}_i\}_{i\in I} \leftarrow \texttt{MKHE.KeyGen}(pp, I)$.
MKHE scheme is also said to be secure if it is semantically secure.

3 Homomorphic Gadget Decomposition

The gadget decomposition technique is conventionally used in HE schemes to manage the noise growth from homomorphic operations such as homomorphic multiplication. Informally, the purpose of gadget decomposition is to represent an arbitrary element of R_q as a linear combination of the entries of a fixed vector (called the *gadget vector*) with small coefficients which determine the size of an error introduced by the key-switching procedure.

The RNS decomoposition is one of the most widely used as the gadget decomposition, since it can be efficiently implemented using techniques such as Number Theoretic Transform (NTT). In this work, while the most of the previous works merely focus on its advantages in the aspect of implementation, we interestingly observe that its inherent homomorphic properties can be exploited to improve the relinearization step in multi-key variants of HE.

To begin with, we briefly recall the definition of the gadget decomposition and define its homomorphic properties. Then, we observe that the RNS decomposition satisfies these homomoprhic properties.

3.1 Basic Terminology

We review basic terminology related to gadget decomposition. We first revisit the definition of gadget decomposition and gadget encryption, and then we remind special modulus method and gadget decomposition in smaller modulus.

Definition 1. *Let Q be an integer. We say that $h : R_Q \to R^k$ is a gadget decomposition if there exist a fixed vector $\mathbf{g} = (g_0, g_1, \ldots, g_{k-1}) \in R_Q^k$ and a constant $B_h > 0$ with the properties: $\langle h(a), \mathbf{g}\rangle = a \pmod Q$ and $\|h(a)\|_\infty \le B_h$ for all $a \in R_Q$.*

We call $\mathbf{g}$ a *gadget vector* and B_h a bound of the gadget decomposition h. We also denote by $g : R^k \to R_Q$ the inner product function defined by $g(\mathbf{u}) = \langle \mathbf{u}, \mathbf{g}\rangle \pmod Q$. We remark that h is a right inverse of g, *i.e.*, $g \circ h$ is the identity function on R_Q.[4] In general, the bound B is much smaller than

[4] This is why the gadget decomposition is often denoted by g^{-1} in the literature, however, this is an abuse of notation since g may have multiple preimages.

the modulus Q. In other words, a gadget decomposition aims to find a short vector in the inverse image $g^{-1}(a) = \{\mathbf{u} \in R^k : \langle \mathbf{u}, \mathbf{g} \rangle = a \pmod{Q}\}$ of input $a \in R_Q$.

The *base decomposition* [16, 6] is a typical example. For an integer $B > 1$, it represents the coefficients of an input polynomial a in base B: $h(a) = (a_0, a_1, \ldots, a_{k-1})$ such that $\sum_{0 \le i < k} a_i \cdot B^i = a$ where $k = \lceil \log_B Q \rceil$. Note that the corresponding gadget vector is $\mathbf{g} = (1, B, \ldots, B^{k-1})$ and $\|h(a)\|_\infty < B$ for any $a \in R_Q$.

We now introduce the notion of gadget encryption which is particularly useful when constructing a secure multiplication with a small noise growth.

Definition 2. *Let s be an RLWE secret. We call $\mathbf{U} = (\mathbf{u}_0, \mathbf{u}_1) \in R_Q^{k \times 2}$ a gadget encryption of $\mu \in R$ under s if $\mathbf{u}_0 + s \cdot \mathbf{u}_1 \approx \mu \cdot \mathbf{g} \pmod{Q}$.*

Definition 3. *For $a \in R_Q$ and $\mathbf{u} \in R_Q^k$, the external product of a and $\mathbf{u}$ is denoted and defined by $a \boxdot \mathbf{u} = \langle h(a), \mathbf{u} \rangle \pmod{Q}$. We also write $a \boxdot \mathbf{U} = (a \boxdot \mathbf{u}_0, a \boxdot \mathbf{u}_1)$ when $\mathbf{U} = (\mathbf{u}_0, \mathbf{u}_1) \in R_Q^{k \times 2}$.*

It is directly obtained from the definition that $a \boxdot \mathbf{g} = a \pmod{Q}$ for all $a \in R_Q$. Moreover, if $\mathbf{U} = (\mathbf{u}_0, \mathbf{u}_1) \in R_Q^{\ell \times 2}$ is a gadget encryption of $\mu \in R$ under s so that $\mathbf{u}_0 + s \cdot \mathbf{u}_1 = \mu \cdot \mathbf{g} + \mathbf{e} \pmod{Q}$ for some small $\mathbf{e} \in R^k$, then the external product $a \boxdot \mathbf{U} = (c_0, c_1)$ of a and $\mathbf{U}$ satisfies that

$$c_0 + s \cdot c_1 = \langle h(a), \mathbf{u}_0 + s \cdot \mathbf{u}_1 \rangle = \langle h(a), \mu \cdot \mathbf{g} + \mathbf{e} \rangle = a \cdot \mu + e \pmod{Q} \tag{1}$$

where the noise term is obtained as $e = \langle h(a), \mathbf{e} \rangle \in R$, which is bounded by $\|e\| \le kN \cdot B_h \|\mathbf{e}\|_\infty$.

Special modulus: The special modulus method [19] is a widely used optimization technique in HE which reduces the noise growth of homomorphic operations. Roughly speaking, it temporarily raises the ciphertext modulus from Q up to PQ for some integer P when performing an external product so that the noise is scaled by P while recovering the modulus into Q. We do not describe this technique specifically in the main body for simplicity, but we take this optimization technique in our implementation. We provide a detailed description of the special modulus method in Appendix A.

Gadget decomposition in a smaller modulus: We often need to define several gadget decompositions in different moduli since the ciphertext modulus may decrease after homomorphic evaluation of a circuit. The rescaling operation of CKKS is a typical example which reduces the ciphertext modulus. Fortunately, we can reuse a gadget decomposition over R_Q in smaller moduli. More precisely, if $(h, \mathbf{g})$ is a pair of gadget decomposition and gadget vector over R_Q and $Q'|Q$, then the restriction of h to $R_{Q'}$ is a valid gadget decomposition corresponding to $[\mathbf{g}]_{Q'}$. Hence we will define only one gadget decomposition with the largest modulus in scheme description.

3.2 Homomorphic Property

We introduce a new concept for the gadget framework which will play a major part in the construction of our MKHE scheme later.

Definition 4. *A homomorphic gadget decomposition $h : R_Q \to R^k$ is a gadget decomposition which satisfies that*

$$\langle h(a) + h(b), \mathbf{g} \rangle = a + b \pmod{Q},$$
$$\langle h(a) \odot h(b), \mathbf{g} \rangle = ab \pmod{Q}$$

for all $a, b \in R_Q$ where $\odot$ denotes the element-wise product of two vectors.

The first additive condition is always true for any gadget decomposition, but the other multiplicative property may not hold in general. Fortunately, many of the gadget decompositions currently in use in the state-of-the-art HE libraries have this homomorphic property, which we will show in the next section.

Our key observation is that the primary goal of gadget decomposition is not to compute a specific vector but it suffices to find a *good enough* decomposition, which is an element of the inverse image $g^{-1}(a)$ with a reasonably small size. For example, the correctness of (1) still holds even if we replace $h(a)$ by another vector in $g^{-1}(a)$ as long as its size is much smaller than Q.

If $h : R_Q \to R^k$ is a homomorphic gadget decomposition, we can perform homomorphic operations over two gadget decompositions of a, b to obtain valid decompositions $h(a)+h(b)$ of $a+b$ and $h(a) \odot h(b)$ of ab, which are bounded by $\|h(a)+h(b)\|_\infty \le 2B$ and $\|h(a) \odot h(b)\|_\infty \le B^2$, respectively.

We also remark that a gadget decomposition h cannot be a ring homomorphism in a mathematical manner, but g can be. Nevertheless, we still use the term "homomorphic gadget decomposition" to describe the properties above.[5]

3.3 Examples

In this section, we introduce some concrete examples of homomorphic gadget decompositions. First of all, we recall the Residue Number System (RNS). When we set the public parameters, the ciphertext modulus Q can be chosen as a product of pairwise coprime integers $q_0, \ldots, q_{\ell-1}$ so that we obtain the ring isomorphism $R_Q \to \prod_{0\le i<\ell} R_{q_i}$, $a \mapsto ([a]_{q_i})_{0\le i<\ell}$ from the Chinese Remainder Theorem (CRT). We call $([a]_{q_i})_{0\le i<\ell}$ the RNS representation of $a \in \mathbb{Z}_Q$ with respect to the base $\{q_0, \ldots, q_{\ell-1}\}$.

Informally, a function on R_Q is said to be RNS-friendly if it can be computed while staying in RNS. Currently, the RNS representation is widely used in design and implementation of HE schemes since performing independent arithmetic operations over R_{q_i} is much faster than one operation over the large ring R_Q. In particular, several HE libraries have been developed without relying on number theory libraries for multi-precision arithmetic after full RNS variants of HE schemes are designed using RNS-friendly gadget decompositions [3, 21, 11, 22]. Below we will provide formal descriptions of two RNS-friendly gadget decompositions and show their homomorphic property.

1. In the first example, we set the RNS base $\{q_0, q_1, \ldots, q_{\ell-1}\}$ as a set of distinct word-size (single-precision) prime numbers. The decomposition function is defined as

$$h : R_Q \to R^\ell, \qquad h(a) = ([a]_{q_0}, [a]_{q_1}, \ldots, [a]_{q_{\ell-1}}).$$

   We point out that this decomposition function looks similar to the CRT isomorphism, but its codomain is R^ℓ, *i.e.,* each component $[a]_{q_i}$ is treated as an element of R instead of R_{q_i}. The corresponding gadget vector is $\mathbf{g} = (g_0, \ldots, g_{\ell-1}) \in R_Q^\ell$ where g_i's are the constants such that $g_i = 1 \pmod{q_i}$ and $g_i = 0 \pmod{q_{i'}}$ for all $i' \ne i$. It is also clear that h is RNS-friendly.
   The primary condition $\langle h(a), \mathbf{g}\rangle = a \pmod{Q}$ of gadget decomposition is satisfied since $\langle h(a), \mathbf{g}\rangle = [a]_{q_i} \pmod{q_i}$ for all i, and its upper bound is $\|h(a)\|_\infty \le \frac{1}{2}\max_i\{q_i\}$. Finally, we can show that h has the homomorphic property since $[a]_{q_i} \cdot [b]_{q_i} = ab \pmod{q_i}$ for all i and thereby $\langle h(a) \odot h(b), \mathbf{g}\rangle = ab \pmod{Q}$ whenever $a, b \in R_Q$.
2. We also present a generalized version of the first example which allows us to reduce the degree of decomposition by using larger RNS bases. To be precise, we choose a partition $\{I_0, I_1, \ldots, I_{k-1}\}$ of $\{0, 1, \ldots, \ell-1\}$ and define partial products $D_i = \prod_{j\in I_i} q_j$ for $0 \le i < k$ which can be multi-precision integers. We also define maps $h_i : R_Q \to R$ as $h_i(a) = \sum_{j\in I_i} [(D_i/q_j)^{-1} \cdot a]_{q_j} \cdot (D_i/q_j)$ for $0 \le i < k$. Then, the decomposition function is defined as

$$h : R_Q \to R^k, \qquad h(a) = (h_0(a), h_1(a), \ldots, h_{k-1}(a))$$

   with the gadget vector $\mathbf{g} = (g_0, \ldots, g_{k-1}) \in R_Q^k$ where g_i's are the constants satisfying $g_i = 1 \pmod{D_i}$ and $g_i = 0 \pmod{D_{i'}}$ for all $i' \ne i$. From the definition of g_i and h_i, we have $\langle h(a), \mathbf{g}\rangle = h_i(a) = a \pmod{q_j}$ for all $0 \le i < k$ and $j \in I_i$. Hence, the primary condition $\langle h(a), \mathbf{g}\rangle = a \pmod{Q}$ is true for all $a \in R_Q$, and h has an upper bound $\|h(a)\|_\infty = \max\{\|h_0(a)\|, \ldots, \|h_{k-1}(a)\|\} \le$

[5] Similarly, the encryption procedure of a homomorphic encryption is not a homomorphism, but the decryption is.

Algorithm 1 Relinearization of CDKS

```
Input: $\overline{\mathsf{ct}}_{mul} = (c_{i,j})_{0 \le i,j \le n} \in R_{Q_\ell}^{(n+1)\times(n+1)}$, $\{\mathsf{pk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i)\}_{1 \le i \le n}$
Output: $\overline{\mathsf{ct}}^* = (c_i^*)_{0 \le i \le n} \in R_{Q_\ell}^{n+1}$
1: $c_0^* \leftarrow c_{0,0}$
2: for $1 \le i \le n$ do
3:     $c_i^* \leftarrow c_{0,i} + c_{i,0} \pmod{Q_\ell}$
4: end for
5: for $1 \le i, j \le n$ do
6:     $c_j^* \leftarrow c_j^* + c_{i,j} \boxdot \mathbf{d}_i \pmod{Q_\ell}$
7:     $c'_{i,j} \leftarrow c_{i,j} \boxdot \mathbf{b}_j$
8:     $(c_0^*, c_i^*) \leftarrow (c_0^*, c_i^*) + c'_{i,j} \boxdot (\mathbf{v}_i, \mathbf{u}_i) \pmod{Q_\ell}$
9: end for
```

$\frac{1}{2}\max_i\{|I_i| \cdot D_i\}$. In addition, h has the homomorphic property since $h_i(a) \cdot h_i(b) = ab \pmod{D_i}$ for $0 \le i < k$ and thereby $\langle h(a) \odot h(b), \mathbf{g}\rangle = ab \pmod{Q}$ whenever $a, b \in R_Q$.

Finally, h is RNS-friendly since the RNS representation of each $h_i(a)$ can be computed using only single-precision modular operations. We also note that there is another variant of h which computes the exact $[a]_{D_i}$ instead of $h_i(a)$ using the formula $[a]_{D_i} = h_i(a) - D_i \cdot \lfloor h_i(a)/D_i \rceil$ where the last term $h_i(a)/D_i = \sum_{j \in I_i} [(D_i/q_j)^{-1} \cdot a]_{q_j} \cdot q_j^{-1}$ is computed via floating-point operations [21]. This variant is also a homomorphic gadget decomposition and has a better bound $\frac{1}{2}\max_i\{D_i\}$.

Interestingly, RNS-friendly gadget decompositions were originally introduced to accelerate basic HE algorithms, but we take advantage of their homomorphic property and design a new multi-key homomorphic multiplication algorithm with asymptotically better complexity in the next section.

4 A Faster Multi-Key Variant of CKKS

In Sections 4 and 5, we design new MKHE schemes from CKKS and BFV. In each section, we will first recall the construction by Chen-Dai-Kim-Song (CDKS) [9], and then modify some algorithms to achieve better performance. In particular, the notion of homomorphic gadget decomposition plays a key role in our construction.

All MKHE schemes presented in the paper are based on the Common Random String (CRS) model, *i.e.,* all parties have access to the same random polynomials sampled in the setup phase. A fresh ciphertext looks like a standard (single key) RLWE encryption, but the ciphertext length may increase when we operate on multiple ciphertexts under different keys. For example, if $\mathsf{ct} = (c_0, c_1)$, $\mathsf{ct}' = (c'_0, c'_1)$ are two ciphertexts under secrets s and s', respectively, then their summation is defined as a *two-key* encryption $\overline{\mathsf{ct}}_{add} = (c_0 + c'_0, c_1, c'_1)$ which is decrytable by the secret (s, s').

More generally, a multi-key ciphertext takes the form of $\overline{\mathsf{ct}} = (c_0, c_1, \ldots, c_n) \in R_Q^{n+1}$ where n is the number of involved parties. It implicitly contains a tuple of the party indices to indicate which secret or public keys should be used in decryption or homomorphic evaluation. Moreover, when performing arithmetic operations on two ciphertexts associated with different sets of parties, the input ciphertexts are embedded into a larger space by padding zeros or permuting some entries to synchronize their secrets.

For simplicity, we assume that this pre-processing is always applied to input ciphertexts so that they are encrypted under the same secret $\overline{\mathsf{sk}} = (s_1, s_2, \ldots, s_n)$ even if it is not explicitly stated in scheme description.

4.1 Overview of Multi-key CKKS by CDKS

In this section, we revisit the multi-key CKKS scheme of CDKS [9].

• $\underline{\texttt{MK-CKKS.Setup}(1^\lambda)}$: Set the RLWE dimension N and the ciphertext modulus $Q = \prod_{i=0}^{L} q_i$ for some integers q_i. We write $Q_\ell = \prod_{i=0}^{\ell} q_i$ for $0 \le \ell \le L$. Set the key distribution χ over R and the error parameter σ. Sample $\mathbf{a} \leftarrow \mathcal{U}(R_Q^k)$. Choose a gadget decomposition $h : R_Q \to R^k$ with a gadget vector $\mathbf{g} \in R_Q^k$. Output the public parameter $pp = (N, Q, \chi, \sigma, \mathbf{a}, h, \mathbf{g})$.

• $\underline{\texttt{MK-CKKS.KeyGen}(i)}$: A party i generates secret and public keys as follows:

– Sample $s_i \leftarrow \chi$ and set the secret key as $\mathsf{sk}_i = s_i$.
– Sample $\mathbf{e}_{0,i} \leftarrow D_\sigma^k$ and let $\mathbf{b}_i = -s_i \cdot \mathbf{a} + \mathbf{e}_{0,i} \pmod Q$.
– Sample $r_i \leftarrow \chi$ and $\mathbf{e}_{1,i} \leftarrow D_\sigma^k$. Let $\mathbf{d}_i = -r_i \cdot \mathbf{a} + s_i \cdot \mathbf{g} + \mathbf{e}_{1,i} \pmod Q$.
– Sample $\mathbf{u}_i \leftarrow \mathcal{U}(R_Q^k)$ and $\mathbf{e}_{2,i} \leftarrow D_\sigma^k$. Let $\mathbf{v}_i = -s_i \cdot \mathbf{u}_i - r_i \cdot \mathbf{g} + \mathbf{e}_{2,i} \pmod Q$.
– Set the public key as $\mathsf{pk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i)$. We also denote the encryption key as $\mathsf{ek}_i = (\mathbf{b}_i[0], \mathbf{a}[0])$.

The key distribution is not specifically defined to keep the generality, but we assume in the noise analysis that χ is defined over the set of polynomials in R with ternary coefficients $\{\pm 1, 0\}$ for simplicity.

• $\underline{\texttt{MK-CKKS.Enc}(\mathsf{ek}; \mu)}$: Sample $w \leftarrow \chi$ and $e_0, e_1 \leftarrow D_\sigma$. Given a plaintext $\mu \in R$, output the ciphertext $\mathsf{ct} = w \cdot \mathsf{ek} + (\mu + e_0, e_1) \pmod Q$.

• $\underline{\texttt{MK-CKKS.Dec}(\{\mathsf{sk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}})}$: Given a ciphertext $\overline{\mathsf{ct}} = (c_0, c_1, \ldots, c_n) \in R_{Q_\ell}^{n+1}$ and associated secret keys $\{\mathsf{sk}_i\}_{1\le i\le n}$, return $\mu = c_0 + \sum_{1\le i\le n} c_i \cdot s_i \pmod{Q_\ell}$.

• $\underline{\texttt{MK-CKKS.Add}(\overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Given two ciphertexts $\overline{\mathsf{ct}}, \overline{\mathsf{ct}}' \in R_{Q_\ell}^{n+1}$, output $\overline{\mathsf{ct}}_{add} = \overline{\mathsf{ct}} + \overline{\mathsf{ct}}' \pmod{Q_\ell}$.

• $\underline{\texttt{MK-CKKS.Mult}(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Given two input ciphertexts $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c_i')_{0\le i\le n} \in R_{Q_\ell}^{n+1}$ and associated public keys $\{\mathsf{pk}_i\}_{1\le i\le n}$, compute $\overline{\mathsf{ct}}_{mul} = (c_{i,j})_{0\le i,j\le n}$ where $c_{i,j} = c_i \cdot c_j' \pmod{Q_\ell}$ for $0 \le i, j \le n$. Output the ciphertext $\texttt{Relin}(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}_{mul})$ where $\texttt{Relin}(\cdot)$ is the relinearization procedure described in Alg. 1.

• $\underline{\texttt{MK-CKKS.Rescale}(\overline{\mathsf{ct}})}$: Given a ciphertext $\overline{\mathsf{ct}} = (c_0, c_1, \ldots, c_n) \in R_{Q_\ell}^{n+1}$, output $\overline{\mathsf{ct}}' = (c_0', c_1', \ldots, c_n') \in R_{Q_{\ell-1}}^{n+1}$ where $c_i' = \lfloor q_\ell^{-1} \cdot c_i \rceil \pmod{Q_{\ell-1}}$ for $0 \le i \le n$.

In lines 5–9 of Alg 1, each entry $c_{i,j}$ of $\overline{\mathsf{ct}}_{mul}$ is relinearized by $\mathbf{b}_j$ of pk_j and $\mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i$ of pk_i to obtain a ciphertext decryptable by s_i and s_j instead of $s_i \cdot s_j$. More precisely, it adds $c_{i,j} \boxdot \mathbf{d}_i$ and $c_{i,j}' \boxdot (\mathbf{v}_i, \mathbf{u}_i)$ to c_j^* and (c_0^*, c_i^*), respectively, so that

$$\begin{aligned}(c_{i,j} \boxdot \mathbf{d}_i) \cdot s_j + c_{i,j}' \boxdot (\mathbf{v}_i + s_i \cdot \mathbf{u}_i) &\approx (c_{i,j} \boxdot \mathbf{d}_i) \cdot s_j - r_i \cdot c_{i,j}' \\ = c_{i,j} \boxdot (s_j \cdot \mathbf{d}_i - r_i \cdot \mathbf{b}_j) &\approx c_{i,j} \boxdot (r_i \cdot \mathbf{a} + \mathbf{d}_i) \cdot s_j \approx c_{i,j} \cdot s_i s_j \pmod{Q_\ell}.\end{aligned}$$

In addition, a relinearization error can be written as $e_{i,j} = c_{i,j}' \boxdot \mathbf{e}_{2,i} + c_{i,j} \boxdot (s_j \cdot \mathbf{e}_{1,i} - r_i \cdot \mathbf{e}_{0,j})$ which is bounded by$\|e_{i,j}\|_\infty \le kN \cdot B_h B_\sigma + 2kN^2 \cdot B_h B_\sigma \approx 2kN^2 \cdot B_h B_\sigma$. Therefore, the total relinearization noise has an upper bound

$$\left\| \sum_{1\le i,j\le n} e_{i,j} \right\|_\infty \lessapprox 2kn^2N^2 \cdot B_h B_\sigma. \tag{2}$$

The relinearization process involves four external products for each $c_{i,j}$, yielding $4n^2$ external products in total.

4.2 Accelerating Multi-Key CKKS Multiplication Using Homomorphic Gadget Decomposition

In this section, we present an improved multiplication method which is asymptotically faster than the previous algorithm. We are inspired by a recent work by Kwak et al. [24] which optimized the relinearization process of CDKS. They simplified the double iterations in lines 5–9 of Alg. 1 by rewriting them using

Algorithm 2 Simplified relinearization [24]

```
Input: $\overline{\mathsf{ct}}_{mul} = (c_{i,j})_{0\le i,j\le n} \in R_{Q_\ell}^{(n+1)\times(n+1)}$, $\{\mathsf{pk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i)\}_{1\le i\le n}$
Output: $\overline{\mathsf{ct}}^* = (c_i^*)_{0\le i\le n} \in R_{Q_\ell}^{n+1}$
 1: $c_0^* \leftarrow c_{0,0}$
 2: for $1 \le i \le n$ do
 3:     $c_i^* \leftarrow c_{0,i} + c_{i,0} \pmod{Q_\ell}$
 4: end for
 5: for $1 \le j \le n$ do
 6:     $c_j^* \leftarrow c_j^* + \sum_{1\le i\le n} c_{i,j} \boxdot \mathbf{d}_i \pmod{Q_\ell}$
 7: end for
 8: for $1 \le i \le n$ do
 9:     $x_i \leftarrow \sum_{1\le j\le n} c_{i,j} \boxdot \mathbf{b}_j \pmod{Q_\ell}$
10:     $(c_0^*, c_i^*) \leftarrow (c_0^*, c_i^*) + x_i \boxdot (\mathbf{v}_i, \mathbf{u}_i) \pmod{Q_\ell}$
11: end for
```

the following single iterations, as described in as shown in Alg. 2.

$$\sum_{1\le i\le n} c_{i,j} \boxdot \mathbf{d}_i = \sum_{1\le i\le n} \langle h(c_{i,j}), \mathbf{d}_i \rangle \quad \text{for} \quad 1 \le j \le n, \tag{3}$$

$$\sum_{1\le j\le n} c_{i,j} \boxdot \mathbf{b}_j = \sum_{1\le j\le n} \langle h(c_{i,j}), \mathbf{b}_j \rangle \quad \text{for} \quad 1 \le i \le n. \tag{4}$$

Despite the optimization, the relinearization process still requires $O(n^2)$ external products since the term $h(c_{i,j})$ in the summands is doubly indexed by both i and j. To further reduce the complexity, we desire to separate out the summands into two parts so that each of them only contains the index either i or j. Then one can rule out the summands indexed by i (or j, resp) from the summation running over j (or i, resp). However, this idea does not work in general since $h(c_{i,j})$ seems unlikely to be separated out for general gadget decomposition.

This is where our homomorphic gadget decomposition comes into play. In our scheme, we choose a gadget decomposition h with the homomorphic property defined in Definition 4. Observing that $c_{i,j} = c_i \cdot c'_j$, we replace the term $h(c_{i,j})$ by $h(c_i) \odot h(c'_j)$. Then the above equations can be re-written as follows:[6]

$$\sum_{1\le i\le n} \langle h(c_i) \odot h(c'_j), \mathbf{d}_i \rangle = \left\langle h(c'_j), \sum_{1\le i\le n} h(c_i) \odot \mathbf{d}_i \right\rangle = c'_j \boxdot \left(\sum_{1\le i\le n} h(c_i) \odot \mathbf{d}_i \right),$$

$$\sum_{1\le j\le n} \langle h(c_i) \odot h(c'_j), \mathbf{b}_j \rangle = \left\langle h(c_i), \sum_{1\le j\le n} h(c'_j) \odot \mathbf{b}_j \right\rangle = c_i \boxdot \left(\sum_{1\le j\le n} h(c'_j) \odot \mathbf{b}_j \right).$$

Based on these equations, we design a new multiplication algorithm. As desired, its complexity can be reduced by precomputing $\sum_{1\le i\le n} h(c_i) \odot \mathbf{d}_i$ and $\sum_{1\le j\le n} h(c'_j) \odot \mathbf{b}_j$ which depend only on either i or j. We also stress that $h(c_i \cdot c'_j) \neq h(c_i) \odot h(c'_j)$ in general, so the modified equations are different from the original ones. Nevertheless, we will show that our algorithm still works correctly since the underlying plaintext information is unchanged.

Now we present a new construction of multi-key CKKS from homomorphic gadget decomposition. Our construction shares several algorithms with CDKS, but we mainly modify the setup and multiplication algorithms as follows:

- <u>`MK-CKKS.Setup`(1^λ)</u>: Set the RLWE dimension N and the ciphertext modulus $Q = \prod_{i=0}^{L} q_i$ for some integers q_i. We write $Q_\ell = \prod_{i=0}^{\ell} q_i$ for $0 \le i \le L$. Set the key distribution χ over R and the error

[6] Note that $\langle \mathbf{x} \odot \mathbf{y}, \mathbf{z} \rangle = \sum_i \mathbf{x}[i] \cdot \mathbf{y}[i] \cdot \mathbf{z}[i] = \langle \mathbf{x}, \mathbf{y} \odot \mathbf{z} \rangle$ for any vectors $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$.

Algorithm 3 New multi-key CKKS multiplication algorithm

```
Input: $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c'_i)_{0\le i\le n}$, $\{\mathsf{pk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{v}_i)\}_{1\le i\le n}$
Output: $\overline{\mathsf{ct}}^* = (c^*_i)_{0\le i\le n} \in R_{Q_\ell}^{n+1}$
 1: $c^*_0 \leftarrow c_0 \cdot c'_0 \pmod{Q_\ell}$
 2: for $1 \le i \le n$ do
 3:     $c^*_i \leftarrow c_0 \cdot c'_i + c_i \cdot c'_0 \pmod{Q_\ell}$
 4: end for
 5: $\mathbf{z} \leftarrow \sum_{1\le i\le n} h(c_i) \odot \mathbf{d}_i \pmod{Q_\ell}$
 6: $\mathbf{w} \leftarrow \sum_{1\le j\le n} h(c'_j) \odot \mathbf{b}_j \pmod{Q_\ell}$
 7: for $1 \le j \le n$ do
 8:     $c^*_j \leftarrow c^*_j + c'_j \boxdot \mathbf{z} \pmod{Q_\ell}$
 9: end for
10: for $1 \le i \le n$ do
11:     $(c^*_0, c^*_i) \leftarrow (c^*_0, c^*_i) + (c_i \boxdot \mathbf{w}) \boxdot (\mathbf{v}_i, \mathbf{u}_i) \pmod{Q_\ell}$
12: end for
```

parameter σ. Sample $\mathbf{a} \leftarrow \mathcal{U}(R_Q^k)$. Choose a *homomorphic* gadget decomposition $h : R_Q \to R^k$ with a gadget vector $\mathbf{g} \in R_Q^k$. Output the public parameter $pp = (N, Q, \chi, \sigma, \mathbf{a}, h, \mathbf{g})$.

• $\underline{\texttt{MK-CKKS.Mult}(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Given two ciphertexts $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c'_i)_{0\le i\le n} \in R_{Q_\ell}^{n+1}$ and associated public keys $\{\mathsf{pk}_i\}_{1\le i\le n}$, execute Alg. 3 and return the output ciphertext $\overline{\mathsf{ct}}^*$.

As mentioned above, our multiplication algorithm does not follow the conventional approach where the tensor product and relinearization are performed sequentially, rather it performs both operations in a simultaneous manner.

Security. The construction of CDKS relies its security on the hardness of RLWE with parameter (N, Q, χ, σ) since it uses the same encryption algorithm as CKKS. In addition, the cryptosystem remains secure even if a public key pk_i is given to the adversary since pk_i is computationally indistinguishable from the uniform distribution over $R_Q^{k\times 4}$ under a circular security assumption (see [9] for detail). Our scheme is also semantically secure under the same assumptions since our scheme shares the same key generation and encryption algorithms as CDKS, and our modification on the multiplication algorithm is irrelevant to the security proof.

Correctness. We focus on the correctness of our new multiplication algorithm. We first remark that a public key $\mathsf{pk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i)$ satisfies the properties $\mathbf{b}_i \approx -s_i \cdot \mathbf{a} \pmod{Q}$, $\mathbf{d}_i \approx -r_i \cdot \mathbf{a} + s_i \cdot \mathbf{g} \pmod{Q}$, and $\mathbf{v}_i + s_i \cdot \mathbf{u}_i \approx -r_i \cdot \mathbf{g} \pmod{Q}$. Therefore,

$$s_j \cdot \mathbf{d}_i \approx -r_i s_j \cdot \mathbf{a} + s_i s_j \cdot \mathbf{g} \approx r_i \cdot \mathbf{b}_j + s_i s_j \cdot \mathbf{g} \pmod{Q}. \tag{5}$$

Now suppose that $\overline{\mathsf{ct}}$ and $\overline{\mathsf{ct}}'$ are multi-key ciphertexts under a secret key $(1, \overline{\mathsf{sk}}) = (1, s_1, \ldots, s_n)$ such that $\langle \overline{\mathsf{ct}}, (1, \overline{\mathsf{sk}}) \rangle = \mu \pmod{Q_\ell}$ and $\langle \overline{\mathsf{ct}}', (1, \overline{\mathsf{sk}}) \rangle = \mu' \pmod{Q_\ell}$ and let $\overline{\mathsf{ct}}^* = (c^*_i)_{0\le i\le n} \leftarrow$ $\texttt{MK-CKKS.Mult}(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')$. Our goal is to show $\langle \overline{\mathsf{ct}}^*, (1, \overline{\mathsf{sk}}) \rangle \approx \mu\mu' \pmod{Q_\ell}$.

First of all, we have

$$\begin{aligned}\langle \overline{\mathsf{ct}}^*, (1, \overline{\mathsf{sk}}) \rangle &= c^*_0 + \sum_{1\le i\le n} c^*_i \cdot s_i \\ &= c_0 \cdot c'_0 + \sum_{1\le i\le n} (c_0 \cdot c'_i + c_i \cdot c'_0) \cdot s_i \\ &\quad + \sum_{1\le j\le n} (c'_j \boxdot \mathbf{z}) \cdot s_j + \sum_{1\le i\le n} (c_i \boxdot \mathbf{w}) \boxdot (\mathbf{v}_i + s_i \cdot \mathbf{u}_i) \pmod{Q_\ell}.\end{aligned}$$

from Alg. 3. In addition, thanks to the homomorphic property of gadget decomposition and (5), the third and fourth terms can be written as

$$\begin{aligned}\sum_{1\le j\le n}(c'_j \boxdot \mathbf{z})\cdot s_j &= \sum_{1\le j\le n}\left(c'_j \boxdot \sum_{1\le i\le n}(h(c_i)\odot \mathbf{d}_i)\right)\cdot s_j\\ &= \sum_{1\le i,j\le n}\left\langle h(c'_j), h(c_i)\odot \mathbf{d}_i\right\rangle\cdot s_j = \sum_{1\le i,j\le n}\left\langle h(c_i)\odot h(c'_j), \mathbf{d}_i\right\rangle\cdot s_j\\ &\approx \sum_{1\le i,j\le n} r_i\cdot\left\langle h(c_i)\odot h(c'_j),\mathbf{b}_j\right\rangle + \sum_{1\le i,j\le n} c_i c'_j\cdot s_i s_j \pmod{Q_\ell} \qquad (6)\end{aligned}$$

and

$$\begin{aligned}\sum_{1\le i\le n}(c_i\boxdot\mathbf{w})\boxdot(\mathbf{v}_i+s_i\cdot\mathbf{u}_i) &\approx -\sum_{1\le i\le n} r_i\cdot(c_i\boxdot\mathbf{w})\\ = -\sum_{1\le i\le n} r_i\cdot\left(c_i\boxdot\sum_{1\le j\le n}(h(c'_j)\odot\mathbf{b}_j)\right) &= -\sum_{1\le i,j\le n} r_i\cdot\left\langle h(c_i), h(c'_j)\odot\mathbf{b}_j\right\rangle\\ = -\sum_{1\le i,j\le n} r_i\cdot\left\langle h(c_i)\odot h(c'_j),\mathbf{b}_j\right\rangle &\pmod{Q_\ell}. \qquad (7)\end{aligned}$$

Putting it all together, we obtain $\langle\overline{\mathsf{ct}}^*,(1,\overline{\mathsf{sk}})\rangle \approx c_0c'_0 + \sum_{1\le i\le n}(c_0c'_i + c_ic'_0)\cdot s_i + \sum_{1\le i,j\le n} c_ic'_j\cdot s_is_j = \langle\overline{\mathsf{ct}},(1,\overline{\mathsf{sk}})\rangle\cdot\langle\overline{\mathsf{ct}}',(1,\overline{\mathsf{sk}})\rangle \pmod{Q_\ell}$ which completes the correctness proof of our multiplication algorithm.

Noise growth and complexity. We first provide a worst-case bound of the multiplication noise of our scheme. We refer the reader to Appendix B.1 for a tighter average-case analysis based on the noise variance.

As shown above, $\overline{\mathsf{ct}}^* \leftarrow \texttt{MK-CKKS.Mult}(\{\mathsf{pk}_i\}_{1\le i\le n};\overline{\mathsf{ct}},\overline{\mathsf{ct}}')$ satisfies that

$$\langle\overline{\mathsf{ct}}^*,(1,\overline{\mathsf{sk}})\rangle = \langle\overline{\mathsf{ct}},(1,\overline{\mathsf{sk}})\rangle\cdot\langle\overline{\mathsf{ct}}',(1,\overline{\mathsf{sk}})\rangle + e_1 + e_2 \pmod{Q_\ell}$$

where e_1 and e_2 are the errors introduced from approximate equalities in (6) and (7), respectively. To be precise, these error terms can be written as

$$\begin{aligned}e_1 &= \sum_{1\le i,j\le n}\left\langle h(c_i)\odot h(c'_j), s_j\cdot\mathbf{e}_{1,i} - r_i\cdot\mathbf{e}_{0,j}\right\rangle,\\ e_2 &= \sum_{1\le i\le n}(c_i\boxdot\mathbf{w})\boxdot\mathbf{e}_{2,i}\end{aligned}$$

which are bounded by $\|e_1\|_\infty \le 2kn^2N^3\cdot B_h^2B_\sigma$ and $\|e_2\|_\infty \le knN\cdot B_hB_\sigma$. As a result, we get a worst-case bound $2kn^2N^3\cdot B_h^2B_\sigma + knN\cdot B_hB_\sigma \approx 2kn^2N^3\cdot B_h^2B_\sigma$ of the multiplication noise.

On the other hand, our multiplication algorithm requires $O(n)$ external products. More specifically, the required number of gadget decompositions (including NTT operations) is $3n$ which is reduced by a factor of $O(n)$ compared to $O(n^2)$ complexity of CDKS relinearization.

The major drawback of our construction is its multiplication noise, whose upper bound is about $N\cdot B_h$ times larger than the previous method. This extra factor is introduced from the additional gadget decomposition and polynomial product of $h(c_i)\odot h(c'_j)$ replacing $h(c_{i,j})$. However, this issue can be addressed easily by the special modulus method. Roughly speaking, we cancel out the extra factor from homomorphic gadget decomposition by taking a special modulus but the maximal level L of cryptosystem can be reduced by one (see Appendix A for details).

In conclusion, our scheme achieves an asymptotically better computation cost while its disadvantage with respect to the noise growth can be easily minimized by a well-known technique.

5 A Faster Multi-Key Variant of BFV

In this section, we design a new multi-key variant of BFV scheme with better performance. The multi-key CKKS and BFV schemes by CDKS are technically very similar since they share the same relinearization algorithm. However, our multi-key CKKS multiplication algorithm is not compatible with BFV due to the scaling factor involved with message encoding. We use a similar approach based on homomorphic gadget decomposition, but present new ideas to resolve the issues from BFV-style multiplication.

5.1 Overview of Multi-Key BFV by CDKS

We provide a description of the multi-key BFV scheme by CDKS as follows. As noted above, the same key generation algorithm as in multi-key CKKS is used to perform the relinearization procedure.

• $\underline{\texttt{MK-BFV.Setup}(1^\lambda)}$: Set the RLWE dimension N, the plaintext modulus t, the ciphertext modulus Q, the key distribution χ over R, and the error parameter σ. Sample $\mathbf{a} \leftarrow \mathcal{U}(R_q^k)$ and choose a gadget decomposition $h : R_Q \to R^k$ with a gadget vector $\mathbf{g} \in R_Q^k$. Output the parameter set $pp = (N, t, Q, \chi, \sigma, \mathbf{a}, h, \mathbf{g})$. We also denote $\Delta = \lfloor Q/t \rceil$.

• $\underline{\texttt{MK-BFV.KeyGen}(i)}$: A party i generates secret and public keys as follows:

– Sample $s_i \leftarrow \chi$ and set the secret key as $\mathsf{sk}_i = s_i$.
– Sample $\mathbf{e}_{0,i} \leftarrow D_\sigma^k$ and let $\mathbf{b}_i = -s_i \cdot \mathbf{a} + \mathbf{e}_{0,i} \pmod Q$.
– Sample $r_i \leftarrow \chi$ and $\mathbf{e}_{1,i} \leftarrow D_\sigma^k$. Let $\mathbf{d}_i = -r_i \cdot \mathbf{a} + s_i \cdot \mathbf{g} + \mathbf{e}_{1,i} \pmod Q$.
– Sample $\mathbf{u}_i \leftarrow \mathcal{U}(R_Q^k)$ and $\mathbf{e}_{2,i} \leftarrow D_\sigma^k$. Let $\mathbf{v}_i = -s_i \cdot \mathbf{u}_i - r_i \cdot \mathbf{g} + \mathbf{e}_{2,i} \pmod Q$.
– Set the public key as $\mathsf{pk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i)$. We also denote the encryption key as $\mathsf{ek}_i = (\mathbf{b}_i[0], \mathbf{a}[0])$.

• $\underline{\texttt{MK-BFV.Enc}(\mathsf{ek}; m)}$: Sample $w \leftarrow \chi$ and $e_0, e_1 \leftarrow D_\sigma$. Given a message $m \in R_t$, output the ciphertext $\mathsf{ct} = w \cdot \mathsf{ek} + (\Delta \cdot m + e_0, e_1) \pmod Q$.

• $\underline{\texttt{MK-BFV.Dec}(\{\mathsf{sk}_i\}_{1\le i\le k}; \overline{\mathsf{ct}})}$: Given a ciphertext $\overline{\mathsf{ct}} = (c_0, c_1, \ldots, c_k) \in R_Q^{n+1}$ and associated secret keys $\{\mathsf{sk}_i\}_{1\le i\le n}$, return $m = \left\lfloor (t/Q) \cdot (c_0 + \sum_{1\le j\le k} c_j \cdot s_j) \right\rceil \pmod t$.

• $\underline{\texttt{MK-BFV.Add}(\overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Given two ciphertexts $\overline{\mathsf{ct}}, \overline{\mathsf{ct}}' \in R_Q^{n+1}$, output $\overline{\mathsf{ct}}_{add} = \overline{\mathsf{ct}} + \overline{\mathsf{ct}}' \pmod Q$.

• $\underline{\texttt{MK-BFV.Mult}(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Given two ciphertexts $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c_i')_{0\le i\le n} \in R_Q^{n+1}$ and associated public keys $\{\mathsf{pk}_i\}_{1\le i\le n}$, compute $\overline{\mathsf{ct}}_{mul} = (c_{i,j})_{0\le i,j\le n} \pmod Q$ where $c_{i,j} = \lfloor (t/Q) \cdot c_i c_j' \rceil \pmod Q$. Output the ciphertext $\texttt{Relin}(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}_{mul})$ where $\texttt{Relin}(\cdot)$ is the relinearization procedure described in Alg. 1.

We remark that $\overline{\mathsf{ct}}_{mul}$ is obtained from the tensor product of two input ciphertexts by scaling it with a factor of (t/Q). In particular, the same relinearization algorithm is used for two multi-key schemes by CDKS.

This multi-key BFV scheme is IND-CPA secure under the same RLWE and circular security assumptions as multi-key CKKS in the previous section. In addition, we can show the correctness of multiplication algorithm from

$$\sum_{0\le i,j\le n} c_{i,j} \cdot s_i s_j \approx (t/Q) \cdot \left(\sum_{0\le i\le n} c_i \cdot s_i \right) \left(\sum_{0\le j\le n} c_j' \cdot s_j \right) \approx \Delta \cdot mm' \pmod Q.$$

whenever $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c_i')_{0\le i\le n}$ are multi-key BFV ciphertexts such that $\sum_{0\le i\le n} c_i \cdot s_i \approx \Delta \cdot m \pmod Q$ and $\sum_{0\le i\le n} c_i' \cdot s_i \approx \Delta \cdot m' \pmod Q$.

5.2 Accelerating Multi-Key BFV Multiplication Using Homomorphic Gadget Decomposition

Recall that our multi-key CKKS multiplication algorithm (Section 4.2) achieves a linear complexity by merging two-step procedure consisting of tensor product and subsequent relinearization via homomorphic gadget decomposition. To be precise, we exploited the following homomorphic property

$$\left\langle h(c_i) \odot h(c'_j), \mathbf{g} \right\rangle = c_i \cdot c'_j \pmod{Q}$$

This approach, however, is not directly applicable to multi-key BFV since it involves an unnatural product beyond the base ring. In multi-key BFV, an entry $c_{i,j}$ for the relinearzation algorithm is not a mere product of two elements, instead, it is $c_{i,j} = \lfloor (t/Q) \cdot c_i c'_j \rceil \pmod{Q}$ where the product of c_i and c'_j is performed in R, not R_Q. As a result, our multi-key CKKS multiplication algorithm is not compatible with BFV since one cannot apply the homomorphic property on $c_{i,j}$ as it involves arithmetic operations in R.

To resolve the issue, we first note that $c_{i,j}$ can be computed properly if we raise the modulus up to $\tilde{Q} := Q^2$ and perform the multiplication in $R_{\tilde{Q}}$. In addition, if $\tilde{h} : R_{\tilde{Q}} \to R^{\tilde{k}}$ is a homomorphic gadget decomposition with a gadget vector $\tilde{\mathbf{g}} \in R^{\tilde{k}}_{\tilde{Q}}$, then it holds that

$$\left\langle \tilde{h}(c_i) \odot \tilde{h}(c_j), t \cdot \tilde{\mathbf{g}} \right\rangle = t \cdot c_i c'_j \approx Q \cdot c_{i,j} \pmod{\tilde{Q}} \tag{8}$$

where c_i and c'_j in the equation are regarded as elements of $R_{\tilde{Q}}$ via the embedding $R_Q \hookrightarrow R_{\tilde{Q}}$. If we scale the above by $(1/Q)$, we can obtain $c_{i,j}$.

Hence, we can use a similar idea as in the previous section to design a new multi-key BFV scheme from homomorphic gadget decomposition that switches the ciphertext modulus from Q to $\tilde{Q}$ and vice versa during multiplication. Unfortunately, this approach may cause security and performance degradation issues since public keys also should be generated over $R_{\tilde{Q}}$.

To cope with such issues, we apply the modulus switching technique to stay in the base ring R_Q. To be precise, instead of simply switching the modulus to $\tilde{Q}$, we rescale (8) by a factor of Q and obtain $\left\langle \tilde{h}(c_i) \odot \tilde{h}(c_j), \lfloor (t/Q) \cdot \tilde{\mathbf{g}} \rceil \right\rangle \approx c_{i,j} \pmod{Q}$. As a result, a public key can be generated over R_Q which addresses the security and efficiency issues.

Finally, we observe that a fixed gadget decomposition is used in the construction of multi-key CKKS, but in fact there are two distinguished layers where we can apply different gadget techniques. This separation is particularly useful in BFV since we can choose an appropriate modulus for each gadget decomposition. In the following, we present a new multi-key BFV scheme based on these ideas and provide security and performance analysis.

- MK-BFV.Setup(1^λ): Set the RLWE dimension N, the plaintext modulus t, the ciphertext modulus Q, the key distribution χ over R, and the error parameter σ. We write $\tilde{Q} = Q^2$. Sample $\mathbf{a} \leftarrow \mathcal{U}(R^{\tilde{k}}_Q)$. Choose homomorphic gadget decompositions $h : R_Q \to R^k$ and $\tilde{h} : R_{\tilde{Q}} \to R^{\tilde{k}}$ with gadget vectors $\mathbf{g} \in R^k_Q$ and $\tilde{\mathbf{g}} \in R^{\tilde{k}}_{\tilde{Q}}$, respectively. Output the public parameter $pp = (N, Q, \chi, \sigma, \mathbf{a}, h, \mathbf{g}, \tilde{h}, \tilde{\mathbf{g}})$. We also denote $\Delta = \lfloor Q/t \rceil$.

We denote the external product with respect to $\tilde{h}$ by $\tilde{\boxdot}$.

- MK-BFV.KeyGen(i): A party i generates secret and public keys as follows:
  - Sample $s_i \leftarrow \chi$ and set the secret key as $\mathsf{sk}_i = s_i$.
  - Sample $\mathbf{e}_{0,i} \leftarrow D^{\tilde{k}}_\sigma$ and let $\mathbf{b}_i = -s_i \cdot \mathbf{a} + \mathbf{e}_{0,i} \pmod{Q}$.
  - Sample $r_i \leftarrow \chi$ and $\mathbf{e}_{1,i} \leftarrow D^{\tilde{k}}_\sigma$. Let $\mathbf{d}_i = -r_i \cdot \mathbf{a} + s_i \cdot \lfloor (t/Q) \cdot \tilde{\mathbf{g}} \rceil + \mathbf{e}_{1,i} \pmod{Q}$.
  - Sample $\mathbf{u}_i \leftarrow \mathcal{U}(R^k_Q)$ and $\mathbf{e}_{2,i} \leftarrow D^k_\sigma$. Let $\mathbf{v}_i = -s_i \cdot \mathbf{u}_i - r_i \cdot \mathbf{g} + \mathbf{e}_{2,i} \pmod{Q}$.
  - Set the public key as $\mathsf{pk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i)$. We also denote the encryption key as $\mathsf{ek}_i = (\mathbf{b}_i[0], \mathbf{a}[0])$.

Algorithm 4 New multi-key BFV multiplication algorithm

```
Input: $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c'_i)_{0\le i\le n}$, $\{\mathsf{pk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i)\}_{1\le i\le n}$
Output: $\overline{\mathsf{ct}}^* = (c^*_i)_{0\le i\le n} \in R_Q^{n+1}$
1: $c^*_0 \leftarrow \lfloor (t/Q)\cdot(c_0c'_0)\rceil \pmod Q$
2: for $1\le i\le n$ do
3:     $c^*_i \leftarrow \lfloor (t/Q)\cdot(c_0c'_i + c_ic'_0)\rceil \pmod Q$
4: end for
5: $\mathbf{z} \leftarrow \sum_{1\le i\le n} \tilde{h}(c_i)\odot \mathbf{d}_i \pmod Q$
6: $\mathbf{w} \leftarrow \sum_{1\le j\le n} \tilde{h}(c'_j)\odot \mathbf{b}_j \pmod Q$
7: for $1\le j\le n$ do
8:     $c^*_j \leftarrow c^*_j + c'_j \,\tilde{\boxdot}\, \mathbf{z} \pmod Q$
9: end for
10: for $1\le i\le n$ do
11:     $(c^*_0, c^*_i) \leftarrow (c^*_0, c^*_i) + (c_i \,\tilde{\boxdot}\, \mathbf{w}) \boxdot (\mathbf{v}_i, \mathbf{u}_i) \pmod Q$
12: end for
```

• $\underline{\texttt{MK-BFV.Mult}(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Given two ciphertexts $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c'_i)_{0\le i\le n} \in R_Q^{n+1}$ and associated public keys $\{\mathsf{pk}_i\}_{1\le i\le n}$, run Alg. 4 and return the ciphertext $\overline{\mathsf{ct}}^* = (c^*_i)_{0\le i\le n} \in R_Q^{n+1}$.

As discussed above, we assume that the entries of input ciphertexts $\overline{\mathsf{ct}}$ and $\overline{\mathsf{ct}}'$ are embedded into $R_{\tilde{Q}}$ so that they can be taken as input of the gadget decomposition $\tilde{h}$, even if it is not explicitly mentioned in Alg. 4.

Security. Our multi-key BFV scheme has the usual BFV encryption algorithm, so it is semantically secure under the RLWE assumption of parameter (N, χ, Q, σ). Similar to the case of multi-key CKKS, it also requires a circular security assumption since $(\mathbf{d}, \mathbf{a})$ and $(\mathbf{v}_i, \mathbf{u}_i)$ form a chain of encryptions of $s_i \cdot \lfloor (t/Q)\cdot \tilde{\mathbf{g}}\rceil$ and $-r_i\cdot \mathbf{g}$ under r_i and s_i, respectively.

Correctness. We prove the correctness of our multiplication algorithm. Suppose $\overline{\mathsf{ct}}^* \leftarrow \texttt{MK-BFV.Mult}(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')$ for some multi-key ciphertexts $\overline{\mathsf{ct}}$ and $\overline{\mathsf{ct}}'$. Our goal is to show that $\left\langle \overline{\mathsf{ct}}^*, (1, \overline{\mathsf{sk}})\right\rangle \approx (t/Q)\cdot \sum_{0\le i,j\le n} c_ic'_j \cdot s_is_j \approx \Delta\cdot mm' \pmod Q$ whenever $\left\langle \overline{\mathsf{ct}}, (1, \overline{\mathsf{sk}})\right\rangle \approx \Delta\cdot m$ and $\left\langle \overline{\mathsf{ct}}', (1, \overline{\mathsf{sk}})\right\rangle \approx \Delta\cdot m'$. From Alg. 4, we have

$$\begin{aligned}\left\langle \overline{\mathsf{ct}}^*, (1, \overline{\mathsf{sk}})\right\rangle &= c^*_0 + \sum_{1\le i\le n} c^*_i \cdot s_i \\ &= \lfloor (t/Q)\cdot(c_0c'_0)\rceil + \sum_{1\le i\le n} \lfloor (t/Q)\cdot(c_0c'_i + c_ic'_0)\rceil \cdot s_i \\ &\quad + \sum_{1\le j\le n} (c'_j \,\tilde{\boxdot}\, \mathbf{z})\cdot s_j + \sum_{1\le i\le n} (c_i \,\tilde{\boxdot}\, \mathbf{w}) \boxdot (\mathbf{v}_i + s_i\cdot \mathbf{u}_i).\end{aligned}$$

The last two terms satisfy that

$$\begin{aligned}\sum_{1\le j\le n} (c'_j \,\tilde{\boxdot}\, \mathbf{z})\cdot s_j &= \sum_{1\le j\le n}\left(c'_j \,\tilde{\boxdot} \sum_{1\le i\le n} (\tilde{h}(c_i)\odot \mathbf{d}_i)\right)\cdot s_j \\ = \sum_{1\le i,j\le n} \left\langle \tilde{h}(c'_j), \tilde{h}(c_i)\odot \mathbf{d}_i\right\rangle\cdot s_j &= \sum_{1\le i,j\le n} \left\langle \tilde{h}(c_i)\odot \tilde{h}(c'_j), \mathbf{d}_i\right\rangle \cdot s_j \\ \approx \sum_{1\le i,j\le n} \left\langle \tilde{h}(c_i)\odot\tilde{h}(c'_j), -r_i\cdot \mathbf{a} + s_i\cdot \lfloor (t/Q)\cdot\tilde{\mathbf{g}}\rceil\right\rangle\cdot s_j & \\ \approx \sum_{1\le i,j\le n} r_i \cdot \left\langle \tilde{h}(c_i)\odot \tilde{h}(c'_j), \mathbf{b}_j\right\rangle + c_{i,j}\cdot s_is_j \pmod Q, & \qquad (9)\end{aligned}$$

and

$$
\begin{aligned}
&\sum_{1\le i\le n} (c_i \,\tilde{\boxdot}\, \mathbf{w}) \boxdot (\mathbf{v}_i + s_i \cdot \mathbf{u}_i) \approx - \sum_{1\le i\le n} r_i \cdot (c_i \,\tilde{\boxdot}\, \mathbf{w}) \\
&= - \sum_{1\le i\le n} r_i \cdot \left\langle \tilde{h}(c_i), \sum_{1\le j\le n} \tilde{h}(c'_j) \odot \mathbf{b}_j \right\rangle = - \sum_{1\le i\le n} r_i \cdot \left\langle \tilde{h}(c_i), \sum_{1\le j\le n} \tilde{h}(c'_j) \odot \mathbf{b}_j \right\rangle \\
&= - \sum_{1\le i,j\le n} r_i \cdot \left\langle \tilde{h}(c_i) \odot \tilde{h}(c'_j), \mathbf{b}_j \right\rangle \pmod{Q} \qquad (10)
\end{aligned}
$$

Therefore, we obtain

$$
\begin{aligned}
\langle \overline{\mathsf{ct}}^*, (1, \overline{\mathsf{sk}}) \rangle &\approx (t/Q) \cdot (c_0 c'_0) + \sum_{1\le i\le n} (t/Q)(c_0 c'_i + c_i c'_0) \cdot s_i + (t/Q) \cdot \sum_{1\le i,j\le n} c_i c'_j \cdot s_i s_j \\
&= (t/Q) \cdot \sum_{0\le i,j\le n} c_i c'_j \cdot s_i s_j \approx \Delta \cdot m m' \pmod{Q}
\end{aligned}
$$

as desired.

Noise growth and complexity. In our noise analysis, we focus on the dominating noise terms from external products and omit noises from rounding. The error terms from (9) and (10) can be written as

$$
\begin{aligned}
e_1 &= \sum_{1\le i,j\le n} \left\langle \tilde{h}(c_i) \odot \tilde{h}(c'_j), s_j \cdot \mathbf{e}_{1,i} - r_i \cdot \mathbf{e}_{0,j} \right\rangle \\
e_2 &= \sum_{1\le i\le n} (c_i \,\tilde{\boxdot}\, \mathbf{w}) \boxdot \mathbf{e}_{2,i}.
\end{aligned}
$$

Therefore, the total multiplication noise is bounded by $\|e_1\|_\infty + \|e_2\|_\infty \le 2\tilde{k} n^2 N^3 \cdot B_{\tilde{h}}^2 B_\sigma + knN \cdot B_h B_\sigma$.

Similar to the case of CKKS, our new multiplication algorithm requires $O(n)$ external products (or gadget decompositions) compared to $O(n^2)$ of the relinearization algorithm by CDKS. By contrast, our scheme has a larger multiplication noise but it has almost no adverse effect on the overall performance of multi-key BFV.

6 Implementation

In this section, we discuss practical implementation issues of our MKHE schemes and provide experimental results. As discussed in Section 3.3, we take advantages of RNS representation via the isomorphism between R_Q and $R_{q_0} \times \cdots \times R_{q_L}$ where $Q = \prod_{i=0}^{L} q_i$ is a product of pairwise coprime integers. Although our multiplication algorithms achieve better performance from the homomorphic property of gadget decomposition, there remains a minor issue with our multi-key BFV scheme due to the gadget decomposition defined over R_{Q^2}. To be precise, our scheme described in Section 5.2 is not fully RNS friendly since it requires multi-precision arithmetic over the moduli q_i^2 as $R_{Q^2} \cong R_{q_0^2} \times \cdots \times R_{q_L^2}$.

We first modify our multi-key BFV scheme in Section 6.1 to enable a full-RNS implementation. Then, we implement both multi-key BFV and CKKS schemes and provide their benchmarks in Section 6.2 to show that our schemes achieve better concrete performance compared to the previous construction by CDKS [9].

6.1 RNS-friendly variant of our multi-key BFV scheme

In the original BFV scheme or its multi-key variant by CDKS, the multiplication algorithm embeds the entries of input ciphertexts from R_Q into R and computes their scaled product $\lfloor (t/Q) \cdot c_i c'_j \rceil \pmod{Q}$ for relinearization. This algorithm can be implemented RNS friendly since the product $c_i c'_j$ returns a valid

Algorithm 5 RNS-friendly multi-key BFV multiplication algorithm

```
Input: $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c'_i)_{0\le i\le n}$, $\{\mathsf{pk}_j = (\mathbf{b}_j, \mathbf{d}_j, \mathbf{u}_j, \mathbf{v}_j)\}_{1\le j\le n}$
Output: $\overline{\mathsf{ct}}^* = (c^*_j)_{0\le j\le n} \in R_Q^{n+1}$
 1: for $0 \le j \le n$ do
 2:     $c''_j \leftarrow \left\lfloor \frac{Q'}{Q} c'_j \right\rceil \pmod{Q'}$
 3: end for
 4: $c^*_0 \leftarrow \lfloor (t/Q') \cdot (c_0 c''_0) \rceil \pmod{Q}$
 5: for $1 \le j \le n$ do
 6:     $c^*_j \leftarrow \lfloor (t/Q') \cdot (c_0 c''_j + c_j c''_0) \rceil \pmod{Q}$
 7: end for
 8: $\mathbf{z} \leftarrow \sum_{1\le i\le n} \tilde{h}(c_i) \odot \mathbf{d}_i \pmod{Q}$
 9: $\mathbf{w} \leftarrow \sum_{1\le j\le n} \tilde{h}(c''_j) \odot \mathbf{b}_j \pmod{Q}$
10: for $1 \le j \le n$ do
11:     $c^*_j \leftarrow c^*_j + c''_j \,\tilde{\boxdot}\, \mathbf{z} \pmod{Q}$
12: end for
13: for $1 \le i \le n$ do
14:     $(c^*_0, c^*_i) \leftarrow (c^*_0, c^*_i) + (c_i \,\tilde{\boxdot}\, \mathbf{w}) \boxdot (\mathbf{v}_i, \mathbf{u}_i) \pmod{Q}$
15: end for
```

result over an arbitrary large modulus. For example, Halevi et al. [21] presented a full-RNS implementation of BFV by performing the computation over $R_{QQ'}$ where Q' is another RNS-friendly modulus coprime to Q.

While our multi-key BFV scheme is designed over R_Q, it is nevertheless not RNS friendly since the multiplication algorithm involves a gadget decomposition over the modulus Q^2. Unfortunately, simply replacing the modulus Q^2 by QQ' for some Q' does not solve the issue since it causes another problem about the scaling factor (Q/t) while performing external products, different from the previous schemes which have a separate relinearization process.

Recently, Kim et al. [23] presented a BFV multiplication algorithm which switches a scale factor of one ciphertext from (Q/t) to (Q'/t) so that the product of ciphertext entries can be computed in an RNS-friendly manner. We use a similar technique to design a full-RNS variant of multi-key BFV. In particular, we modify not only multiplication but also the key generation algorithm so that public keys are generated over $R_{QQ'}$. A detailed description of our scheme is given below.

- $\underline{\texttt{MK-BFV.Setup}(1^\lambda)}$: Set the RLWE dimension N, the plaintext modulus t, the ciphertext modulus Q, the key distribution χ over R, and the error parameter σ. We write $\tilde{Q} = QQ'$. Sample $\mathbf{a} \leftarrow \mathcal{U}(R_Q^{\tilde{k}})$. Choose homomorphic gadget decompositions $h : R_Q \to R^k$ and $\tilde{h} : R_{\tilde{Q}} \to R^{\tilde{k}}$ with gadget vectors $\mathbf{g} \in R_Q^k$ and $\tilde{\mathbf{g}} \in R_{\tilde{Q}}^{\tilde{k}}$, respectively. Output the public parameter $pp = (N, Q, Q', \chi, \sigma, \mathbf{a}, h, \mathbf{g}, \tilde{h}, \tilde{\mathbf{g}})$. We also denote $\Delta = \lfloor Q/t \rceil$.

We denote the external product with respect to the gadget decomposition $\tilde{h}$ by $\tilde{\boxdot}$.

- $\underline{\texttt{MK-BFV.KeyGen}(i)}$: A party i generates secret and public keys as follows:
  - Sample $s_i \leftarrow \chi$ and set the secret key as $\mathsf{sk}_i = s_i$.
  - Sample $\mathbf{e}_{0,i} \leftarrow D_\sigma^{\tilde{k}}$ and let $\mathbf{b}_i = -s_i \cdot \mathbf{a} + \mathbf{e}_{0,i} \pmod{Q}$.
  - Sample $r_i \leftarrow \chi$ and $\mathbf{e}_{1,i} \leftarrow D_\sigma^{\tilde{k}}$. Let $\mathbf{d}_i = -r_i \cdot \mathbf{a} + s_i \cdot \lfloor (t/Q') \cdot \tilde{\mathbf{g}} \rceil + \mathbf{e}_{1,i} \pmod{Q}$.
  - Sample $\mathbf{u}_i \leftarrow \mathcal{U}(R_Q^k)$ and $\mathbf{e}_{2,i} \leftarrow D_\sigma^k$. Let $\mathbf{v}_i = -s_i \cdot \mathbf{u}_i - r_i \cdot \mathbf{g} + \mathbf{e}_{2,i} \pmod{Q}$.
  - Set the public key as $\mathsf{pk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i)$. We also denote the encryption key as $\mathsf{ek}_i = (\mathbf{b}_i[0], \mathbf{a}[0])$.

- $\underline{\texttt{MK-BFV.Mult}(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: Given two ciphertexts $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c'_i)_{0\le i\le n} \in R_Q^{n+1}$ and public keys $\{\mathsf{pk}_i\}_{1\le i\le n}$, run Alg. 5 and return the ciphertext $\overline{\mathsf{ct}}^*$.

We scale an input ciphertext and compute $\overline{\mathsf{ct}}'' = \lfloor (Q'/Q) \cdot \overline{\mathsf{ct}}' \rceil \pmod{Q'}$ in lines 1–3 of Alg. 5. Similar to the previous version, we raise the modulus of $\overline{\mathsf{ct}}$ and $\overline{\mathsf{ct}}''$ up to $\tilde{Q}$ using the embeddings $R_Q \hookrightarrow R_{\tilde{Q}}$ and $R_{Q'} \hookrightarrow R_{\tilde{Q}}$, respectively, from line 4. All experimental results in the next section is based on our implementation of this RNS-friendly variant.

6.2 Experimental results

We implement our multi-key CKKS and BFV schemes and provide some benchmark results. Our implementation is based on the Lattigo library v2.3.0 [15] written in Go. All experiments were performed with a single thread on a server machine with Intel(R) Xeon(R) Platinum 8268 @ 2.90GHz CPU and 192GB RAM running Ubuntu 20.04.3 LTS.

Ours				CDKS			
$\log N$	$\#q_i$	$\#p_i$	$\lceil \log QP \rceil$	$\log N$	$\#q_i$	$\#p_i$	$\lceil \log QP \rceil$
14	6	2	439	14	7	1	439
15	14	2	880	15	15	1	880

Table 1. Parameter sets. $\#q_i$ and $\#p_i$ indicate the number of primes used for ciphertext modulus $Q = \prod_i q_i$ and special modulus $P = \prod_i p_i$, respectively.

In our implementation, the key distribution χ samples each coefficients from $\{0, \pm 1\})$ with probability 0.25 for each of -1 and 1 and with probability 0.5 for 0. The error parameter is $\sigma = 3.2$. We also use an RNS-friendly homomorphic gadget decomposition $h(a) = ([a]_{q_0}, [a]_{q_1}, \ldots, [a]_{q_L})$ together with the special modulus method to control the noise growth. Since our multiplication algorithm introduces an extra noise factor from homomorphic gadget decomposition, we keep two primes to form a special modulus compared to one of CDKS implementation. As a result, our implementation has a smaller ciphertext modulus Q for the same RLWE dimension N.

We choose two 60-bit primes for the special modulus and others for the ciphertext modulus are 52–55 bits prime numbers. Table 6.2 presents two parameter sets used in our implementation, both of which achieve at least 128-bit security level against the best known attack for RLWE.

N	n	Ours		CDKS	
		CKKS	BFV	CKKS	BFV
14	2	0.17 s	0.31 s	0.17 s	0.26 s
	4	0.34 s	0.59 s	0.52 s	0.72 s
	8	0.67 s	1.16 s	1.85 s	2.35 s
	16	1.30 s	2.30 s	6.94 s*	8.47 s*
	32	2.59 s	4.59 s	26.93 s*	32.11 s*
	64	5.18 s	9.16 s	106.06 s*	125.03 s*
15	2	1.53 s	2.82 s	1.36 s	1.72 s
	4	3.10 s	6.02 s	4.29 s	5.03 s
	8	5.95 s	10.91 s	15.16 s	17.45 s
	16	11.76 s	21.62 s	57.01 s*	65.52 s*
	32	23.53 s	44.93 s	221.12 s*	254.54 s*
	64	48.84 s	88.30 s	871.01 s*	1,004.10 s*

Table 2. Performance of multiplication algorithms of CDKS and our MKHE schemes (*: estimated results).

In Table 2, we give execution times of our multiplication algorithms for $n = 2, 4, \ldots, 64$ parties.[7] As expected from the complexity analysis, the running time grows linearly with n which rapidly outperforms the performance of CDKS with quadratic complexity as n increases as shown in Fig. 1.

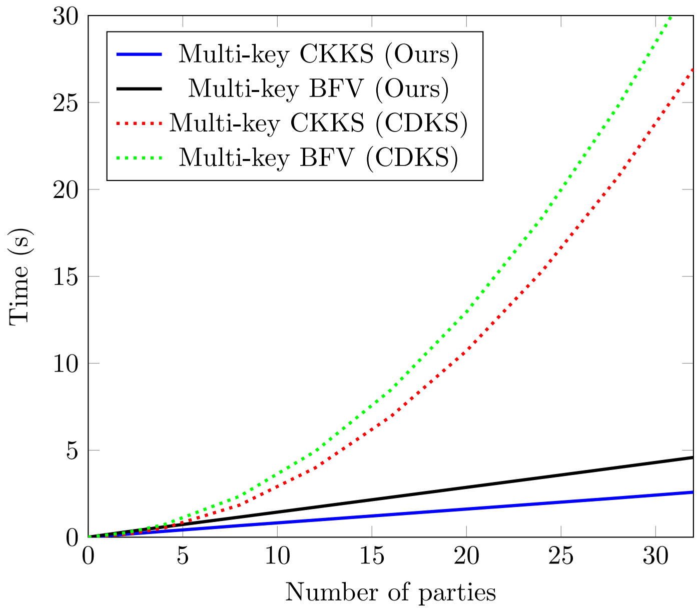

Fig. 1. Performance of multiplication algorithms when $\log N = 14$

References

1. Ananth, P., Jain, A., Jin, Z., Malavolta, G.: Multi-key fully-homomorphic encryption in the plain model. In: Theory of Cryptography Conference. pp. 28–57. Springer (2020)
2. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold fhe. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 483–501. Springer (2012)
3. Bajard, J.C., Eynard, J., Hasan, M.A., Zucca, V.: A full rns variant of fv like somewhat homomorphic encryption schemes. In: International Conference on Selected Areas in Cryptography. pp. 423–442. Springer (2016)
4. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Annual International Cryptology Conference. pp. 565–596. Springer (2018)
5. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual Cryptology Conference. pp. 868–886. Springer (2012)
6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
7. Brakerski, Z., Perlman, R.: Lattice-based fully dynamic multi-key FHE with short ciphertexts. In: Annual Cryptology Conference. pp. 190–213. Springer (2016)
8. Chen, H., Chillotti, I., Song, Y.: Multi-key homomorphic encryption from TFHE. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 446–472. Springer (2019)

[7] We also present benchmarks of Chen et al. [9] as reference, but the previous experimental results were generated on a different machine with Intel Xeon E-2176M @ 4.00 GHz. We also estimate the running time of CDKS based on the complexity analysis for $n \geq 16$.

9. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 395–412 (2019)
10. Chen, L., Zhang, Z., Wang, X.: Batched multi-hop multi-key FHE from Ring-LWE with compact ciphertext extension. In: Theory of Cryptography Conference. pp. 597–627. Springer (2017)
11. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: A full rns variant of approximate homomorphic encryption. In: International Conference on Selected Areas in Cryptography. pp. 347–368. Springer (2018)
12. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 409–437. Springer (2017)
13. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: international conference on the theory and application of cryptology and information security. pp. 3–33. Springer (2016)
14. Clear, M., McGoldrick, C.: Multi-identity and multi-key leveled fhe from learning with errors. In: Annual Cryptology Conference. pp. 630–656. Springer (2015)
15. EPFL-LDS: Lattigo v2.3.0. Online: `https://github.com/ldsec/lattigo` (Oct 2021)
16. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
17. Genise, N., Micciancio, D., Polyakov, Y.: Building an efficient lattice gadget toolkit: Subgaussian sampling and more. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 655–684. Springer (2019)
18. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing. pp. 169–178. ACM (2009)
19. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the aes circuit. In: Annual Cryptology Conference. pp. 850–867. Springer (2012)
20. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Annual Cryptology Conference. pp. 75–92. Springer (2013)
21. Halevi, S., Polyakov, Y., Shoup, V.: An improved rns variant of the bfv homomorphic encryption scheme. In: Cryptographers' Track at the RSA Conference. pp. 83–105. Springer (2019)
22. Han, K., Ki, D.: Better bootstrapping for approximate homomorphic encryption. In: Cryptographers' Track at the RSA Conference. pp. 364–390. Springer (2020)
23. Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 608–639. Springer (2021)
24. Kwak, H., Lee, D., Song, Y., Wagh, S.: A unified framework of homomorphic encryption for multiple parties with non-interactive setup. Cryptology ePrint Archive, Report 2021/1412 (2021), `https://ia.cr/2021/1412`
25. López-Alt, A., Tromer, E., Vaikuntanathan, V.: 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. pp. 1219–1234. ACM (2012)
26. Mouchet, C., Troncoso-Pastoriza, J., Bossuat, J.P., Hubaux, J.P.: Multiparty homomorphic encryption from ring-learning-with-errors. Proceedings on Privacy Enhancing Technologies **2021**(4), 291–311 (2021)
27. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 735–763. Springer (2016)
28. Park, J.: Homomorphic encryption for multiple users with less communications. IEEE Access **9**, 135915–135926 (2021)
29. Peikert, C., Shiehian, S.: Multi-key fhe from lwe, revisited. In: Theory of Cryptography Conference. pp. 217–238. Springer (2016)

A Special Modulus Variant

In this section, we describe the *special modulus technique* and apply it to our MKHE schemes. We introduce a new constant P, called a *special modulus*, and redefine the gadget encryption and external product as follows.

Definition 5. *Let s be an RLWE secret. We call $\mathbf{U} = (\mathbf{u}_0, \mathbf{u}_1) \in R_{QP}^{k\times 2}$ a gadget encryption of $\mu \in R$ under s if $\mathbf{u}_0 + s \cdot \mathbf{u}_1 \approx P\mu \cdot \mathbf{g} \pmod{QP}$.*

Definition 6. *Let $h : R_Q \to R^k$ be a gadget decomposition. For $a \in R_Q$ and $\mathbf{u} \in R_{QP}^k$, the external product of a and $\mathbf{u}$ is denoted and defined as follows.*

$$a \boxdot \mathbf{u} := \left\lfloor P^{-1} \cdot \langle h(a), \mathbf{u}\rangle \right\rceil \pmod{Q}$$

From the definitions, it is satisfied that $a \boxdot P\mathbf{g} = a \pmod{Q}$ for all $a \in R_Q$. If $\mathbf{U} = (\mathbf{u}_0, \mathbf{u}_1) \in R_Q^{k\times 2}$ is a gadget encryption of $\mu \in R$ under s so that $\mathbf{u}_0 + s \cdot \mathbf{u}_1 = \mu \cdot P\mathbf{g} + \mathbf{e} \pmod{Q}$ for some small $\mathbf{e} \in R^k$, then the external product $a \boxdot \mathbf{U} = (c_0, c_1)$ of a polynomial a and $\mathbf{U}$ holds that

$$\begin{aligned} c_0 + s \cdot c_1 &= \left\lfloor P^{-1} \cdot \langle h(a), \mathbf{u}_0\rangle \right\rceil + s \cdot \left\lfloor P^{-1} \cdot \langle h(a), \mathbf{u}_1\rangle \right\rceil \\ &= P^{-1} \langle h(a), \mu \cdot P\mathbf{g} + \mathbf{e}\rangle + e_{rd} \\ &= a \cdot \mu + e \pmod{Q} \end{aligned}$$

for the rounding noise e_{rd} and $e = P^{-1} \cdot \langle h(a), \mathbf{e}\rangle + e_{rd}$, which is bounded by $\|e\|_\infty \le P^{-1}kN \cdot B_h \|\mathbf{e}\|_\infty + \frac{1}{2}(N+1)$.

Note that the noise of external product is approximately reduced by a factor of P compared to the original external product. Therefore, we can choose a special modulus P properly to control the noise growth.

A.1 Multi-Key CKKS

• <u>`MK-CKKS.Setup`(1^λ)</u>: Set the RLWE dimension N, the ciphertext modulus $Q = \prod_{i=0}^{L} q_i$ for integers q_i, and the sepcial modulus P. We write $Q_\ell = \prod_{i=0}^{\ell} q_i$ for $0 \le i \le L$. Set the key distribution χ over R and the error parameter σ. Sample $\mathbf{a} \leftarrow \mathcal{U}(R_{QP}^k)$. Choose a gadget decomposition $h : R_Q \to R^k$ with a gadget vector $\mathbf{g} \in R_Q^k$. Output the public parameter $pp = (N, Q, P, \chi, \sigma, \mathbf{a}, h, \mathbf{g})$.

• <u>`MK-CKKS.KeyGen`(i)</u>: A party i generates secret and public keys as follows:

– Sample $s_i \leftarrow \chi$ and set the secret key as $\mathsf{sk}_i = s_i$.
– Sample $\mathbf{e}_{0,i} \leftarrow D_\sigma^k$ and let $\mathbf{b}_i = -s_i \cdot \mathbf{a} + \mathbf{e}_{0,i} \pmod{QP}$.
– Sample $r_i \leftarrow \chi$ and $\mathbf{e}_{1,i} \leftarrow D_\sigma^k$. Let $\mathbf{d}_i = -r_i \cdot \mathbf{a} + s_i \cdot P\mathbf{g} + \mathbf{e}_{1,i} \pmod{QP}$.
– Sample $\mathbf{u}_i \leftarrow \mathcal{U}(R_{QP}^k)$ and $\mathbf{e}_{2,i} \leftarrow D_\sigma^k$. Let $\mathbf{v}_i = -s_i \cdot \mathbf{u}_i - r_i \cdot P\mathbf{g} + \mathbf{e}_{2,i} \pmod{QP}$.
– Set the public key as $\mathsf{pk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i)$. We also denote the encryption key as $\mathsf{ek}_i = (\mathbf{b}_i[0], \mathbf{a}[0])$.

• <u>`MK-CKKS.Enc`$(\mathsf{ek}; \mu)$</u>: Sample $w \leftarrow \chi$ and $e_0, e_1 \leftarrow D_\sigma$. Given a plaintext $\mu \in R$, output the ciphertext $\mathsf{ct} = \left\lfloor P^{-1} \cdot (w \cdot \mathsf{ek} + (e_0, e_1)) \right\rceil + (\mu, 0) \pmod{Q}$.

• <u>`MK-CKKS.Mult`$(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')$</u>: Given two ciphertexts $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c_i')_{0\le i\le n} \in R_{Q_\ell}^{n+1}$ and associated public keys $\{\mathsf{pk}_i\}_{1\le i\le n}$, execute Alg. 6 and return the output.

Algorithm 6 Multi-key CKKS multiplication algorithm with special modulus

```
Input: $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c'_i)_{0\le i\le n}$, $\{\mathsf{pk}_j = (\mathbf{b}_j, \mathbf{d}_j, \mathbf{v}_j)\}_{1\le j\le n}$
Output: $\mathsf{ct}^* = (c^*_j)_{0\le j\le n} \in R_{Q_\ell}^{n+1}$
 1: $c^*_0 \leftarrow c_0 \cdot c'_0 \pmod{Q_\ell}$
 2: for $1 \le i \le n$ do
 3:     $c^*_i \leftarrow c_0 \cdot c'_i + c_i \cdot c'_0 \pmod{Q_\ell}$
 4: end for
 5: $\mathbf{z} \leftarrow \sum_{1\le i\le n} h(c_i) \odot \mathbf{d}_i \pmod{Q_\ell P}$
 6: $\mathbf{w} \leftarrow \sum_{1\le j\le n} h(c'_j) \odot \mathbf{b}_j \pmod{Q_\ell P}$
 7: for $1 \le j \le n$ do
 8:     $c^*_j \leftarrow c^*_j + c'_j \boxdot \mathbf{z} \pmod{Q_\ell}$
 9: end for
10: for $1 \le i \le n$ do
11:     $(c^*_0, c^*_i) \leftarrow (c^*_0, c^*_i) + (c_i \boxdot \mathbf{w}) \boxdot (\mathbf{v}_i, \mathbf{u}_i) \pmod{Q_\ell}$
12: end for
```

A.2 Multi-Key BFV

- $\underline{\texttt{MK-BFV.Setup}(1^\lambda)}$: Set the RLWE dimension N, the plaintext modulus t, the ciphertext modulus Q, the key distribution χ over R, and the error parameter σ. We write $\tilde{Q} = QQ'$ and the special modulus P. Sample $\mathbf{a} \leftarrow \mathcal{U}(R_{QP}^{\tilde{k}})$. Choose homomorphic gadget decompositions $h : R_Q \to R^k$ and $\tilde{h} : R_{\tilde{Q}} \to R^{\tilde{k}}$ with gadget vectors $\mathbf{g} \in R_Q^k$ and $\tilde{\mathbf{g}} \in R_{\tilde{Q}}^{k}$, respectively. Output the public parameter $pp = (N, Q, Q', P, \chi, \sigma, \mathbf{a}, h, \mathbf{g}, \tilde{h}, \tilde{\mathbf{g}})$. We also denote $\Delta = \lfloor Q/t \rceil$.

- $\underline{\texttt{MK-BFV.KeyGen}(i)}$: A party i generates secret and public keys as follows:
  - Sample $s_i \leftarrow \chi$ and set the secret key as $\mathsf{sk}_i = s_i$.
  - Sample $\mathbf{e}_{0,i} \leftarrow D_\sigma^{\tilde{k}}$ and let $\mathbf{b}_i = -s_i \cdot \mathbf{a} + \mathbf{e}_{0,i} \pmod{QP}$.
  - Sample $r_i \leftarrow \chi$ and $\mathbf{e}_{1,i} \leftarrow D_\sigma^{\tilde{k}}$. Let $\mathbf{d}_i = -r_i \cdot \mathbf{a} + s_i \cdot \lfloor (t/Q') \cdot P\tilde{\mathbf{g}} \rceil + \mathbf{e}_{1,i} \pmod{QP}$.
  - Sample $\mathbf{u}_i \leftarrow \mathcal{U}(R_{QP}^k)$ and $\mathbf{e}_{2,i} \leftarrow D_\sigma^k$. Let $\mathbf{v}_i = -s_i \cdot \mathbf{u}_i - r_i \cdot P\mathbf{g} + \mathbf{e}_{2,i} \pmod{QP}$.
  - Set the public key as $\mathsf{pk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{u}_i, \mathbf{v}_i)$. We also denote the encryption key as $\mathsf{ek}_i = (\mathbf{b}_i[0], \mathbf{a}[0])$.

- $\underline{\texttt{MK-BFV.Enc}(\mathsf{ek}; m)}$: Sample $w \leftarrow \chi$ and $e_0, e_1 \leftarrow D_\sigma$. Given a message $m \in R_t$, output the ciphertext $\mathsf{ct} = \lfloor P^{-1} \cdot (w \cdot \mathsf{ek} + (e_0, e_1)) \rceil + (\mu, 0) \pmod{Q}$.

- $\underline{\texttt{MK-BFV.Mult}(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')}$: For two input ciphertexts $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c'_i)_{0\le i\le n} \in R_Q^{n+1}$ and public keys$\{\mathsf{pk}_i\}_{1\le i\le n}$, execute Algorithm 7 and then, return the output $\overline{\mathsf{ct}}^*$.

B Average-Case Noise Analysis

We analyze an average-case noise growth of our novel multi-key CKKS/BFV multiplication algorithms. In this section, we make a heuristic assumption that each ciphertext component behaves as if it is a uniform random variable over R_Q. We denote the variance of coefficients for a polynomial $a = \sum_i a_i \cdot X^i$ over the ring R by $\mathsf{Var}(a) = \mathsf{Var}(a_i)$. Then, the variance of the product of two independent polynomials $c = a \cdot b$ with degree N is evaluated as $\mathsf{Var}(c) = N \cdot \mathsf{Var}(a) \cdot \mathsf{Var}(b)$. More generally, we define the variance of a vector $\mathbf{a} \in R^k$ of independent random variables as $\mathsf{Var}(\mathbf{a}) = \frac{1}{k}\sum_{0\le i<k} \mathsf{Var}(\mathbf{a}[i])$.

Let $V_h = \mathsf{Var}(h(a))$ for a uniform random polynomial $a \in R_Q$ and a gadget decomposition h. In the prime decomposition $h : R_Q \to \prod_{0\le i\le L} R_{q_i}, a \mapsto ([a]_{q_i})_{0\le i\le L}$, we have $V_h \approx \frac{1}{12}\sum_{0\le i\le L} q_i^2$.

Algorithm 7 Multi-key BFV multiplication algorithm with special modulus

```
Input: $\overline{\mathsf{ct}} = (c_i)_{0\le i\le n}$, $\overline{\mathsf{ct}}' = (c'_i)_{0\le i\le n}$, $\{\mathsf{pk}_j = (\mathbf{b}_j, \mathbf{d}_j, \mathbf{u}_j, \mathbf{v}_j)\}_{1\le j\le n}$
Output: $\overline{\mathsf{ct}}^* = (c^*_j)_{0\le j\le n} \in R_Q^{n+1}$
 1: for $0 \le j \le n$ do
 2:     $c''_j \leftarrow \left\lfloor \frac{Q'}{Q} c'_j \right\rceil \pmod{Q'}$
 3: end for
 4: $c^*_0 \leftarrow \lfloor (t/Q') \cdot (c_0 c''_0) \rceil \pmod{Q}$
 5: for $1 \le j \le n$ do
 6:     $c^*_j \leftarrow \lfloor (t/Q') \cdot c_0 c''_j \rceil + \lfloor (t/Q') \cdot c_j c''_0 \rceil \pmod{Q}$
 7: end for
 8: $\mathbf{z} \leftarrow \sum_{1\le i\le n} \tilde{h}(c_i) \odot \mathbf{d}_i \pmod{QP}$
 9: $\mathbf{w} \leftarrow \sum_{1\le j\le n} \tilde{h}(c''_j) \odot \mathbf{b}_j \pmod{QP}$
10: for $1 \le j \le n$ do
11:     $c^*_j \leftarrow c^*_j + c''_j \,\tilde{\boxdot}\, \mathbf{z} \pmod{Q}$
12: end for
13: for $1 \le i \le n$ do
14:     $(c^*_0, c^*_i) \leftarrow (c^*_0, c^*_i) + (c_i \,\tilde{\boxdot}\, \mathbf{w}) \boxdot (\mathbf{v}_i, \mathbf{u}_i) \pmod{Q}$
15: end for
```

B.1 Multi-key CKKS

We showed in Section 4.2 that the output ciphertext $\overline{\mathsf{ct}}^*$ of our multi-key CKKS multiplication algorithm satisfies that

$$\langle \overline{\mathsf{ct}}^*, (1, \overline{\mathsf{sk}}) \rangle = \langle \overline{\mathsf{ct}}, (1, \overline{\mathsf{sk}}) \rangle \cdot \langle \overline{\mathsf{ct}}', (1, \overline{\mathsf{sk}}) \rangle + e_1 + e_2$$

where

$$\begin{aligned} e_1 &= \sum_{1\le i,j\le n} \left\langle h(c_i) \odot h(c'_j), s_j \cdot \mathbf{e}_{1,i} - r_i \cdot \mathbf{e}_{0,j} \right\rangle, \\ e_2 &= \sum_{1\le i\le n} (c_i \boxdot \mathbf{w}) \boxdot \mathbf{e}_{2,i}. \end{aligned}$$

We can show that $\mathsf{Var}(e_1) = n^2 k N^3 \sigma^2 V_h^2$, $\mathsf{Var}(e_2) = nkN\sigma^2 V_h$ and $\mathsf{Var}(e_1 + e_2) = \mathsf{Var}(e_1) + \mathsf{Var}(e_2)$ since $\mathbf{e}_{1,i}, \mathbf{e}_{0,j}, \mathbf{e}_{2,i}$ are zero-mean and the covariance between any two of terms is zero. Hence, the variance of the total noise is approximately $n^2 k N^3 \sigma^2 V_h^2$.

B.2 Multi-key BFV

In our multi-key BFV multiplication algorithm $\overline{\mathsf{ct}}^* \leftarrow \texttt{MK-BFV.Mult}(\{\mathsf{pk}_i\}_{1\le i\le n}; \overline{\mathsf{ct}}, \overline{\mathsf{ct}}')$ in Section 5.2, we focus on the noise term $e_1 + e_2$ where

$$\begin{aligned} e_1 &= \sum_{1\le i,j\le n} \left\langle \tilde{h}(c_i) \odot \tilde{h}(c'_j), s_j \cdot \mathbf{e}_{1,i} - r_i \cdot \mathbf{e}_{0,j} \right\rangle, \\ e_2 &= \sum_{1\le i\le n} (c_i \,\tilde{\boxdot}\, \mathbf{w}) \boxdot \mathbf{e}_{2,i}. \end{aligned}$$

Then we have $\mathsf{Var}(e_1) = n^2 \tilde{k} N^3 \sigma^2 V_{\tilde{h}}^2$, $\mathsf{Var}(e_2) = nkN\sigma^2 V_h$, and the variance of the multiplication error $e_1 + e_2$ is $\mathsf{Var}(e_1 + e_2) = n^2 \tilde{k} N^3 \sigma^2 V_{\tilde{h}}^2 + nkN\sigma^2 V_h$.