

Multi-Key FHE with Compact Ciphertexts (or: MKFHE Strikes Back)

Intak Hwang , Seonhong Min , and Yongsoo Song 

Seoul National University, Seoul, Republic of Korea
{intak.hwang,minsh,y.song}@snu.ac.kr

Abstract. Multi-Key Fully Homomorphic Encryption (MKFHE) is a variant of fully homomorphic encryption (FHE) that allows computations over data encrypted under different keys. Its support for non-interactive, dynamic participation and on-the-fly computation makes it an attractive primitive for multi-party applications. However, MKFHE has been largely overlooked in practice as its performance scales poorly, since the ciphertext size grows linearly with the number of participants in all existing constructions. Resolving this issue was a major open problem for almost a decade.

In this work, we close this gap by providing the first construction of an MKFHE scheme with compact ciphertexts: the size of ciphertexts remains constant throughout the entire homomorphic evaluation, even when the number of participating parties changes during computation. As a result, our construction retains all the advantages of MKFHE while simultaneously offering fast evaluation performance similar to that of a single-key scheme.

Our methodology is generic and applies to a broad class of RLWE-based FHE schemes. Concretely, we construct efficient multi-key variants of RGSW and FV-like schemes, and further develop a compiler that transforms a wide range of RLWE-based FHE schemes into their multi-key counterparts.

We implement our multi-key FV scheme in Go. Empirical results show that our scheme achieves $30.6\times$ faster multiplication and $60.6\times$ faster automorphism for 16 parties compared to the state-of-the-art construction by Kwak et al. (CCS '23), demonstrating the concrete efficiency of our construction.

1 Introduction

Fully Homomorphic encryption (FHE) has been a promising ingredient for many cryptographic protocols as it allows arbitrary circuit evaluation on encrypted data. Since the groundbreaking work by Gentry [25], there have been significant advancements in both efficiency and functionality [12, 11, 24, 26, 23, 20, 19, 7, 36, 29]. Conventional FHE assumes a single-user setting, limiting its functionality to only two-party computation (2PC) protocols. However, many real-world protocols, including federated learning and blockchain applications, assume a multiple

user/client setting. Consequently, many different approaches to enabling homomorphic computation among multiple parties have been proposed. This research direction has attracted mass attention, leading to the U.S. National Institute of Standards and Technology (NIST) calling for the standardization of FHE for multiple parties. Currently, there are two major approaches in constructing FHE for multiple parties, each with their own pros and cons.

Multi-Key FHE (MKFHE) [39, 44, 21, 49, 13, 16, 37, 14, 15, 31, 34, 54, 47, 53], first introduced by Lopez-Alt et al. [39], allows computation over ciphertexts encrypted under different secret keys. In MKFHE schemes, each party can generate their secret and public evaluation keys during an offline phase, and can dynamically join a computation without interacting with other parties. This property allows MKFHE to be used in *on-the-fly* MPC protocols, where the participants need not be fixed in advance, achieving optimal round complexity and communication size independent of the circuit depth. Unfortunately, the ciphertext size of most existing MKFHE schemes grows linearly with the number of participants k . More importantly, the homomorphic computation needs to happen over this larger size, causing performance to also scale with k . Due to this limitation, MKFHE schemes quickly become unpractical as the number of parties grows.

ThFHE [6, 10, 51, 22, 8, 48, 45, 43, 42], introduced by Asharov et al. [5], introduced by Asharov et al. [5], takes the opposite approach. It distributes shares of the secret key of a single-key homomorphic encryption scheme to multiple parties using a t -out-of- n access structure, instead of allowing each party to have an independent secret key. As a result, ciphertexts remain compact, just as in single-key FHE schemes. Homomorphic evaluation proceeds the same way as well, exhibiting excellent performance independent of the number of parties. However, establishing a single-key FHE secret typically requires an expensive setup via a generic multi-party computation (MPC) protocol [51, 22, 8], or a trusted third party assumption [48, 45] in order to achieve the same key and noise distribution as single-key FHE schemes. Alternatively, some works [43, 42, 46] consider multi-party FHE (MPFHE), corresponding to the n -out-of- n setting, where the joint secret key is additively shared among the parties. This lightweight access structure allows for a much more efficient setup phase, in return for slightly larger key and noise distributions and sublinear overhead in homomorphic evaluation performance. Nevertheless, all existing ThFHE constructions require at least two communication rounds before computation and do not allow for dynamism.

As there are significant tradeoffs between these two paradigms, a natural question arises as to whether we can construct an FHE scheme for multiple parties that enjoys the best of both worlds. Indeed, this has been an important open problem for almost a decade, with partial solutions being suggested throughout the literature. Multi-Group FHE (MGFHE) [33, 38], proposed by Kwak et al. [33], combines MKFHE and ThFHE by allowing independently established groups to participate in on-the-fly computation. While it provides additional flexibility over MKFHE and ThFHE, it also inherits disadvantages from both. Some variants [2, 17] introduce some form of dynamicity to ThFHE by making the parties output additional materials for evaluation keys. However, their con-

structions require new parties to download the existing parties' key materials. These limitations motivate us to ask:

- Can we build an FHE scheme for multiple parties that supports both*
- (i) Dynamic, on-the-fly computation, and*
 - (ii) compact ciphertext size and evaluation performance comparable to a single-key scheme?*

1.1 Our Contribution

In this work, we answer the aforementioned question affirmatively. We construct multi-key encryption schemes that follow the MPFHE blueprint, unlike prior approaches. Specifically, we build upon a single-key FHE scheme with an additive joint key structure, enabling a compact ciphertext size and fast evaluation performance. At the same time, our schemes still remain categorically MKFHE, retaining their advantages such as non-interactive key generation and dynamic participation. In this sense, our result shows that MKFHE *subsumes* MPFHE.

Our construction relies on a novel key-switching method that can convert a single-key ciphertext into a multi-key ciphertext under the joint key, under the common random string (CRS) model. We also provide various tradeoffs between time and space complexity under the random oracle model (ROM). On top of this, we first construct multi-key variants of the RGSW scheme [26] and the FV scheme [11, 24]. As a byproduct, multi-key variants of the BGV [12] and the CKKS [19] can also be obtained, thanks to their structural similarity to the FV scheme.

Moreover, we provide a generic *compiler* that transforms a broad class of single-key FHE schemes into their multi-key counterparts, using the key-switching method and the multi-key RGSW (MKRGSW) scheme as ingredients. This compiler does not depend on a specific design of a particular FHE scheme. We also provide concrete instantiations for the AP-family FHE schemes (FHEW [23, 36], TFHE [20], Carousel [29]) to demonstrate the generality of our compiler.

Finally, we provide a proof-of-concept implementation of the multi-key FV (MKFV) scheme. For 16 parties, our benchmark results show that the multiplication and automorphism performance of our MKFV scheme is $30.6\times$ and $60.6\times$ faster than the prior state-of-the-art [31], respectively.

1.2 Related Works

Concurrent Work. Concurrently, Afshar and Goyal [1] proposed a generic distributed decryption protocol for LWE-based MKFHE schemes, which makes the output ciphertext independent of the number of parties, hence achieving fully succinct MKFHE. Roughly speaking, they use a cryptographic primitive called non-interactive compact oblivious tensor evaluation (OTE) to transform the output ciphertext into a constant-sized digest after computation. However, this transformation happens only after the evaluation, and the ciphertext size still remains linear with respect to the number of parties during homomorphic evaluation, just like existing MKFHE schemes.

2 Technical Overview

2.1 Background

For a power-of-two N and positive integer Q , let $R = \mathbb{Z}[X]/(X^N + 1)$ and $R_Q = R \bmod Q$. For some Boolean function f , $\delta_f(x)$ outputs 1 if $f(x)$ is true; otherwise outputs 0. In RLWE-based FHE schemes, an encryption of $\mu \in R$ under a secret key $s \in R$ is a tuple of elements $(b := -as + \mu + e, a) \in R_Q^2$, where a is sampled uniformly and e is a short error. Decryption follows from the linear relation $b + as \approx \mu \pmod{Q}$.

Existing RLWE-based MKFHE schemes naturally extend this relation by concatenating the ciphertexts associated with multiple secret keys. For k parties with secret keys $s_1, \dots, s_k$, an MKFHE ciphertext typically contains $k + 1$ ring elements $(b, a_1, \dots, a_k) \in R_Q^{k+1}$, where it can be decrypted using the parties' secret keys as $b + \sum_{i=1}^k a_i s_i \approx \mu \pmod{Q}$ ¹. This form naturally supports dynamic, on-the-fly participation, as a ciphertext can be extended to accommodate new parties without establishing a new joint secret key. However, this approach makes the ciphertext dimension grow linearly with the number of parties, thereby causing the evaluation cost to scale significantly with the number of parties. Indeed, the evaluation performance of state-of-the-art MKFHE constructions [31, 53] scales quasi-linearly with the number of parties.

MPFHE schemes take a different approach. Instead of extending the secret key, the parties set the joint secret key as $\bar{s} = \sum_{i=1}^k s_i$ and maintain the ciphertext as a tuple $(b, a) \in R_Q^2$, just as in the standard FHE scheme. Then, homomorphic evaluation can proceed essentially the same way as a single-key FHE scheme with the key $\bar{s}$, making evaluation substantially more efficient than MKFHE schemes. To enable such compact ciphertext, MPFHE schemes commonly use a common reference string (CRS) to collaboratively generate a joint public key corresponding to $\bar{s}$, sacrificing an additional round. Recall that in RLWE based schemes, the public key can be viewed as an encryption of zero. The parties first agree on a uniformly random element $u_1 \in R_Q$, provided as a CRS. Then, the i -th party samples a short error e_i and broadcasts $u_0^{(i)} := -u_1 s_i + e_i$. Aggregating these partial public keys gives a valid public key $(u_0 := \sum_i u_0^{(i)}, u_1) \in R_Q^2$. After the aggregation, any party can encrypt their messages directly under $\bar{s}$ using the joint public key.

One might instead try to avoid this additional round by using this method for encryption themselves, by outputting zero encryptions except for the encryptor, which outputs an encryption of its message under the same mask. However, this requires fresh CRS elements for different ciphertexts, since reusing the same CRS element allows the adversary to easily recover the secret key $\bar{s}$. Therefore, it would require the number of encryptions to be bounded in advance, which is problematic.

¹ A notable exception is the LTV scheme [39] and the MPS scheme [41], in which the joint key structure is multiplicative. In these schemes, the bit-length of the modulus, instead of the dimension, grows linearly. Moreover, it is known that the LTV scheme is insecure in relevant parameter regimes [3].

2.2 Removing the Joint Public Key

We replace the joint public key with *key-switching keys* that allow us to switch the secret of a ciphertext from s_i to $\bar{s}$. This allows each party to output encryption under s_i , rather than depending on a joint public key.

In our construction, each party first broadcasts partial key-switching keys along with their encryptions under s_i . These partial key-switching keys are then aggregated to generate key-switching keys from s_i to $\bar{s}$. Subsequently, each party's local encryptions under the key s_i can be key-switched to $\bar{s}$, making them compatible with the homomorphic evaluation algorithm of a single-key FHE scheme.

The main advantage of this approach is that the partial key-switching keys can be sent concurrently with the encryptions, unlike the sequential broadcasting of the MPFHE constructions. This allows us to eliminate an additional public key encryption round, achieving optimal round complexity.

However, constructing such key-switching keys is challenging. Below, we detail two unsuccessful attempts at building such a key-switching key for a better understanding, then we provide our final approaches.

First attempt. Recall that the key-switching key from s_i to $\bar{s}$ for RLWE-based FHE schemes is (a set of) RLWE ciphertexts of the form

$$\begin{bmatrix} \mathbf{b} = -\mathbf{a} \cdot \bar{s} + \vec{g} \cdot s_i + \mathbf{e} \\ \mathbf{a} \end{bmatrix} \in R_Q^{2 \times d}$$

where $\vec{g} \in R_Q^d$ is a gadget vector. Since we only need a single key-switching key per party, we can bound the total number of key-switching keys once the maximum number of parties is fixed. Therefore, we can use the same CRS-based key-generation method from MPFHE schemes to generate the key-switching keys.

Concretely, we first generate a list of CRS $\{\mathbf{a}_i\}_{1 \leq i \leq K}$, where K is the (maximum) number of parties. Then, the i -th party outputs a partial key-switching key

$$\mathbf{b}_j^{(i)} = -\mathbf{a}_j \cdot s_i + \delta_{i=j} \cdot \vec{g} s_i + \mathbf{e}_{i,j} \pmod{Q}$$

for $1 \leq j \leq K$. It is easy to verify that $(\mathbf{b}_i = \sum_j \mathbf{b}_i^{(j)} \pmod{Q}, \mathbf{a}_i)$ is indeed a key-switching key from s_i to the joint key $\bar{s}$, since

$$\mathbf{b}_i + \mathbf{a}_i \cdot \bar{s} = \sum_{j=1}^k \mathbf{b}_i^{(j)} + \mathbf{a}_i \cdot \sum_{j=1}^k s_j = \sum_{j=1}^k (\mathbf{b}_i^{(j)} + \mathbf{a}_i s_j) \approx \vec{g} \cdot s_i \pmod{Q}$$

for any $1 \leq i \leq K$ and $2 \leq k \leq K$. Given this key-switching key, the evaluator can convert any single-key ciphertext encrypted under s_i to a ciphertext encrypted under the joint key $\bar{s}$ on-the-fly.

However, each party should know its index i to generate correct partial key-switching keys since the i -th party should encrypt $\vec{g} s_i$ for the i -th CRS. This directly violates the syntax of multi-key FHE (Def. 5), as key generation should

be agnostic to the party executing it. Even in real-world protocols, establishing order among parties generally requires additional rounds or trusted setups, which contradicts the use of multi-key FHE as on-the-fly multi-party computation.

Second attempt. To solve this problem, we can use a probabilistic approach instead of prescribing a certain order. Suppose the maximum number of parties K is fixed. During key generation, each party simply picks a random index between 1 and M for a sufficiently large $M \gg K$. Then, with overwhelming probability, the indices each party chooses will be distinct, which we can then use directly in the above method.

Let us explain in more detail. Suppose a list of CRS $\{\mathbf{a}_i\}_{1 \leq i \leq M}$ is given, and let the i -th party's chosen index be $\text{idx}^{(i)}$. Then, i -th party outputs its index $\text{idx}^{(i)}$ and the set of partial key-switching keys $\{\mathbf{b}_j^{(i)}\}_{1 \leq j \leq M}$ where

$$\mathbf{b}_j^{(i)} = -\mathbf{a}_j \cdot s_i + \delta_{\text{idx}^{(i)}=j} \cdot \vec{g}s_i + \mathbf{e}_{i,j} \pmod{Q}.$$

Then, the key-switching key from s_i to the joint key $\bar{s}$ can be obtained by aggregating $(\mathbf{b}_i = \sum_j \mathbf{b}_{\text{idx}^{(i)}}^{(j)} \pmod{Q}, \mathbf{a}_{\text{idx}^{(i)}})$, similar to the discussion above.

Although this idea is valid, it is still too expensive to be used in real-world protocols. The main reason is that each party should output M partial key-switching keys. However, to achieve $2^{-\kappa}$ collision probability between indices, M should be exponentially large in κ .

Our Solution. To reduce the communication cost, we propose two approaches, which we call the *repetition-based* method and *tree-based* method. The Repetition-based method reduces the partial key-switching key size to $\text{poly}(K, \kappa)$, while the tree-based method further reduces the size to $\text{polylog}(K) + \text{poly}(\kappa)$, at the cost of $\text{poly}(K)$ key-switching operations per ciphertext.

Repetition-based Method. The core idea of the repetition-based method is to instantiate the above construction with a non-negligible failure probability and repeat it so that the overall failure probability drops exponentially.

Let M be the number of indices as above, and A be the number of repetitions. Given A sets of M CRS elements $\mathbf{a}_{\alpha,j}$ for $1 \leq \alpha \leq A$ and $1 \leq j \leq M$, i -th party first chooses A indices $\text{idx}_1^{(i)}, \dots, \text{idx}_A^{(i)}$ uniformly from $\{1, \dots, M\}$. Then, using the CRS, they output

$$\mathbf{b}_{\alpha,j}^{(i)} = -\mathbf{a}_{\alpha,j} \cdot s_i + \delta_{\text{idx}_\alpha^{(i)}=j} \cdot \vec{g}s_i + \mathbf{e}_{i,\alpha,j} \pmod{Q}$$

for all $\mathbf{a}_{\alpha,j}$. Upon receiving these partial key-switching keys, for a party i , we choose a repetition index α_i^* , where $\text{idx}_{\alpha_i^*}^{(i)} \neq \text{idx}_{\alpha_i^*}^{(j)}$ for all $j \neq i$. By aggregating partial key-switching keys in the α_i^* -th repetition, we can obtain a key-switching key from s_i to $\bar{s}$. The total number of partial key-switching keys that a single party should output equals AM which is $\text{poly}(K, \kappa)$.

Tree-based Method. The tree-based method considers an *iterative* key-switching approach rather than direct key-switching. In this method, for some factor B ,

a ciphertext is first key-switched to a joint key corresponding to B indices, and then to B^2 indices, continuing until it is switched to the final joint key. While this requires $\log_B K$ key-switches, only a subset of all key-switching keys depends on each party's secret key.

We can visualize this as a B -ary tree, where the leaf nodes consist of the secret keys of each party, and each parent node is the joint secret key formed by the summation of its children. Naturally, the root node becomes the final joint key $\bar{s}$. We can understand the key-switching process from s_i to $\bar{s}$ as following a path from the leaf node s_i to the root node. Then, each party only needs to contribute to the key-switching keys required for following such a path. To be specific, it needs to contribute on the key-switching keys needed between the nodes on the path from s_i to $\bar{s}$, and their direct children, which sums up to $B \lceil \log K \rceil$ key-switching keys at most. We visualize a toy example in Figure 1.

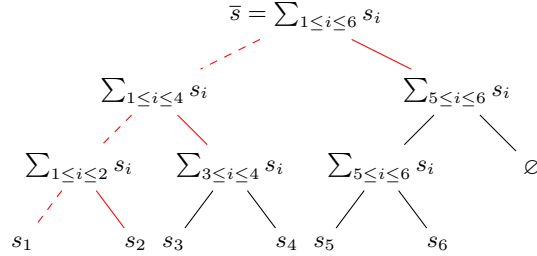


Fig. 1. Example of B -ary tree structure for key-switching when $B = 2, k = 6$. The key-switching path from s_1 to $\bar{s}$ is marked in dashed lines; The contributions of key-switching keys need for the first party is marked in red.

We formalize and extend this approach with random indexing as follows. First, the i -th party chooses a random index $\text{idx}^{(i)}$ uniformly from $\{1, \dots, M\}$. By decomposing $\text{idx}^{(i)}$ as base- B digits, we can identify it with a path from s_i to $\bar{s}$ in the key-switching tree. For simplicity, we write $\text{idx}^{(i)}[:\ell]$ as the prefix of length ℓ of this base- B string, and $\text{idx}^{(i)}[\ell]$ as the ℓ -th digit. Given CRS elements $\mathbf{a}_{\ell,j}$ for $1 \leq \ell \leq L$ and $1 \leq j \leq M$, where $L \leq \lceil \log_B M \rceil$ is the height of the key-switching tree, each party outputs

$$\mathbf{b}_{\ell,j}^{(i)} = -\mathbf{a}_{\ell,j} \cdot s_i + \delta_{\text{idx}^{(i)}[\ell]=j} \vec{g} s_i + \mathbf{e}_{i,\ell,\beta} \pmod{Q}$$

for $1 \leq \ell \leq L$ and $B \cdot \text{idx}^{(i)}[:\ell-1] \leq j < B(\text{idx}^{(i)}[:\ell-1] + 1)$. By aggregation, we can obtain key-switching keys to a partial joint key corresponding to the node $\text{idx}^{(i)}[\ell]$ from its child joint keys. The total number of partial key-switching keys that a single party should output is bounded by BL , which is $\text{polylog}(K) + \text{poly}(\kappa)$ with constant-sized B .

We note that the total number of CRS elements is still exponential, unlike the repetition-based method. However, only the CRS elements that lie on the

parties' key-switch paths are used. Across at most K parties, the number of CRS elements needed is at most $KB \log_B M = \text{poly}(K, \kappa)$. We can therefore generate each required CRS element on demand using a hash function modeled as a random oracle.

2.3 MKFHE With Compact Ciphertext

We now describe our construction of MKFHE schemes based on the key-switching algorithm described above and a new multi-key multiplicative operation called the *joint key multiplication*. As a result, we obtain concrete multi-key variants of the RGSW and FV schemes and a generic compiler that transforms a large class of RLWE-based single-key FHE schemes into their multi-key counterparts.

Joint Key Multiplication. The main challenge in achieving full homomorphism with multi-key is the multiplicative operation. Specifically, many existing FHE schemes rely on the RGSW cryptosystem, whose plaintext encoding requires information about the secret. However, in the multi-key setting, every party contributes to the joint secret, making it hard to design a multi-key RGSW.

We resolve this problem with a new multiplicative algorithm called *joint key multiplication*, inspired by the non-interactive relinearization key-generation algorithm due to Kwak et al. [33], which is based on the *uni-encryption* cryptosystem, a multi-key variant of RGSW-like encryption introduced by Chen, Chillotti, and Song [14]. We observe that this relinearization key can be used to multiply the joint key $\bar{s}$ homomorphically to any RLWE ciphertext encrypted under $\bar{s}$.

Multi-Key RGSW and FV. Our multi-key RGSW and FV schemes can be constructed from our key-switching method and the joint key multiplication method. Note that an RGSW encryption $\bar{C} \in R_Q^{2 \times 2d}$ of μ under the joint secret key $\bar{s}$ decrypts to

$$[1 \ \bar{s}] \cdot \bar{C} \approx [\mu \cdot \bar{g}^\top \ \mu \bar{s} \cdot \bar{g}^\top] \pmod{Q}.$$

Observe that this can be (roughly) seen as an encryption of two messages μ and $\mu \bar{s}$. Therefore, to convert a single-key encryption of μ to an RGSW encryption, we first key-switch it to the joint key $\bar{s}$ to obtain an encryption of μ under $\bar{s}$. Then, we homomorphically multiply $\bar{s}$ to generate an encryption of $\mu \bar{s}$. Combining, we obtain a valid RGSW encryption. Since the RGSW encryption, per se, supports addition and multiplication without any evaluation keys, this directly implies a multi-key RGSW scheme.

The plaintext encoding of the FV scheme does not require any information on the secret key $\bar{s}$, unlike RGSW. Therefore, its ciphertext can be easily converted to a multi-key ciphertext via our key-switching method. However, the main challenge in realizing a multi-key FV scheme is a special procedure called

relinearization. Briefly, it requires generating $a\bar{s}^2$ for any $a \in R_Q$. We observe that this can also be achieved with joint key multiplication by applying it to a trivial ciphertext $(0, a)$ of $a \cdot \bar{s}$. On top of this, we additionally provide an optimized automorphism key generation protocol, thereby implying a multi-key variant of the FV scheme. We remark that the BGV and CKKS schemes share a very similar structure to the FV scheme; their multi-key variants can also be constructed analogously.

Multi-Key FHE Compiler. We further extend our MKFHE construction to a broader class of RLWE-based FHE schemes, called *compilable FHE*, whose evaluation keys are RGSW ciphertexts and the plaintext encoding is independent of the secret key. Converting a single-key ciphertext to a multi-key ciphertext is easy, as their underlying messages are independent of the joint key $\bar{s}$, as in the FV scheme. However, the main challenge lies in the generation of the evaluation key, as it typically encrypts secret values. In the multi-key setting, they must be generated without requiring interaction among the participating parties, which poses additional complexity in scheme design.

Instead of directly generating the evaluation keys, we exploit the RGSW circuit evaluation. As the evaluation keys are RGSW ciphertexts themselves, the secret key-dependent computation performed during evaluation key generation can be evaluated homomorphically using our multi-key RGSW scheme. Consequently, we obtain the following pipeline: (1) Each party independently generates and broadcasts partial key-switching keys (Sec. 2.2), MK-RGSW encryptions of secret key-dependent values that are required by the evaluation key generation algorithm, and the input ciphertexts. (2) The evaluator generates key-switching keys and uses them to key-switch the input ciphertexts, running the evaluation key generation algorithm homomorphically to obtain the evaluation keys for $\bar{s}$. (3) Finally, circuits can be homomorphically evaluated with the circuit evaluation algorithm of the underlying single-key FHE.

We show that existing AP-like schemes such as LMKCDEY [36], TFHE [20, 30] and Carousel [29] fall into the compilable FHE category, hence can be compiled with our general MKFHE compiler to obtain their multi-key variants.

3 Preliminary

For additional preliminaries, we refer to Sec. A.

3.1 Notations

For a power-of-two integer N and a positive integer q , let $R = \mathbb{Z}[X]/(X^N + 1)$ be the $2N$ -th cyclotomic ring of integers, and $R_q = R \bmod q$. For a ring element $a \in R$ or R_q , we define $\|a\|_\infty$ (resp. $\|a\|_2$) to be the infinite norm (resp. 2-norm) of the coefficient vector of a .

3.2 (Ring) Learning With Errors

We recall the (decisional) Learning With Errors (LWE) problem [50] over the ring of integers of a cyclotomic field [52, 40]. This ring variant of LWE is often referred to as the *RLWE* problem. Thanks to its structural efficiency, it is used as the main foundation of most of today's FHE schemes.

Definition 1 ((Decisional) RLWE [52]). *Let m be a positive integer χ, ψ be a distribution over R . Then, the Decisional RLWE problem is to distinguish the following two distributions.*

$$\{(\mathbf{a}, \mathbf{a}s + \mathbf{e}) \mid \mathbf{a} \leftarrow R_Q^m, s \leftarrow \chi, \mathbf{e} \leftarrow \psi^m\}, \quad \{(\mathbf{a}, \mathbf{u}) \mid \mathbf{a}, \mathbf{u} \leftarrow R_Q^m\}.$$

Following this notation, we denote the key and noise distributions by χ and ψ . Now, based on the hardness of the RLWE problem, we can define an RLWE encryption scheme.

- **RLWE.Enc($s; \mu$):** Given a secret key $s \in R$ and a message $\mu \in R_Q$, sample $a \xleftarrow{\$} R_Q$ and $e \leftarrow \psi$. Compute $b = -as + \mu + e \bmod Q$ and output a vector $(b, a) \in R_Q^2$.

An RLWE encryption $(b, a) \in R_Q^2$ of the plaintext $\mu \in R_Q$ can be decrypted by computing the following linear relation

$$\begin{bmatrix} 1 & s \end{bmatrix} \cdot \begin{bmatrix} b \\ a \end{bmatrix} = b + as \approx \mu \pmod{Q}.$$

3.3 Gadget Decomposition and Gadget-based Cryptosystems

In this section, we recall the notion of gadget decomposition and cryptosystems based on it. We begin by introducing the notion of gadget decomposition, which is a widely-used noise management trick employed in many LWE-based FHE schemes.

Definition 2 (Gadget Decomposition). *Let Q and B be positive integers. Then, we call a vector $\vec{g} \in \mathbb{Z}^d$ and a function $h : R_Q \rightarrow R^d$ a gadget vector and gadget decomposition, respectively, if the following holds for every $a \in R_Q$.*

- $\langle g^{-1}(a), \vec{g} \rangle = a \pmod{Q}$
- $\|g^{-1}(a)\|_\infty \leq B$

Based on the gadget decomposition, we define ring leveled encryption (RLEV) as follows.

- **RLEV.Enc($s; \mu$):** Given a secret key $s \in R$ and a message $\mu \in R_Q$, sample $\mathbf{a} \xleftarrow{\$} R_Q^d$ and $\mathbf{e} \leftarrow \psi^d$. Compute and output

$$\begin{bmatrix} \mathbf{b} = -\mathbf{a}^\top s + \mu \cdot \vec{g}^\top + \mathbf{e}^\top \bmod Q \\ \mathbf{a}^\top \end{bmatrix} \in R_Q^{2 \times d}.$$

Notably, the RLEV encryption is also used for the key-switching operation in the RLWE-based FHE schemes. Essentially, a *key-switching key* of a secret key s to another secret key s' as $\text{ksk}_{s \rightarrow s'}$ is an RLEV encryption of s under s' . The key-switching procedure is given as follows.

- $\text{KeySwitch}(\text{ksk}_{s \rightarrow s'}, \text{ct})$: Given a key-switching key $\text{ksk}_{s \rightarrow s'} \in R_Q^{2 \times d}$ and a ciphertext $\text{ct} = (b, a) \in R_Q^2$ encrypted under a key s , output a new ciphertext

$$\text{ct}' = \begin{bmatrix} b \\ 0 \end{bmatrix} + \text{ksk}_{s \rightarrow s'} \cdot g^{-1}(a) \pmod{Q}.$$

We also recall the RGSW scheme, which is a ring variant of the Gentry-Sahai-Waters cryptosystem [26].

- $\text{RGSW.Enc}(s; \mu)$: Given a secret $s \in R$ and a message $\mu \in R_Q$, sample $\mathbf{a} \xleftarrow{\$} R_Q^{2d}$ and $\mathbf{e} \leftarrow \psi^{2d}$. Compute and output

$$\begin{bmatrix} \mathbf{b} = -\mathbf{a}^\top s + \mathbf{e}^\top \\ \mathbf{a}^\top \end{bmatrix} + \mu \cdot \vec{g}^\top \otimes I_2 \in R_Q^{2 \times 2d}.$$

We remark that an RGSW encryption of μ can be seen as a stack of two RLEV encryptions of μ and μs .

4 Non-Interactive Key-Switching Key Generation

In this section, we construct a non-interactive key-switching key generation protocol. The goal of this protocol is to generate key-switching keys needed for key-switching s_i to the joint secret key $\bar{s} = \sum_i s_i$, where s_i is the secret key independently sampled by the i -th party. We provide two orthogonal constructions, namely the *repetition-based* method and the *tree-based* method, each optimizing computation and communication costs.

4.1 Repetition-Based Method

We formally describe the repetition-based non-interactive key-switching key generation protocol below.

- $\text{KSKSetup}(K, N, Q, \vec{g}, g^{-1})$: On input the maximum number of parties K , ciphertext dimension N , ciphertext modulus Q , the gadget vector $\vec{g} \in \mathbb{Z}^d$, and the corresponding decomposition function $g^{-1} : R_Q \rightarrow R^d$, choose the number of repetitions A and the number of indices M . Sample $\mathbf{a}_{i,j} \xleftarrow{\$} R_Q^d$ for $1 \leq i \leq A$ and $1 \leq j \leq M$, and output $\text{crs}_{\text{ksk}} = \{\mathbf{a}_{i,j}\}_{1 \leq i \leq A, 1 \leq j \leq M}$.

- **KSKGen**(pp, s): On input the public parameters, which include the gadget vector $\vec{g}$, the noise distribution ψ , and the CRS $\text{crs}_{\text{ksk}} = \{\mathbf{a}_{\alpha,j}\}_{1 \leq \alpha \leq A, 1 \leq j \leq M}$, and the secret key s , pick A indices $\text{id}x_{\alpha}$ at random for $1 \leq \alpha \leq A$. Then, compute

$$\mathbf{b}_{\alpha,j} = -\mathbf{a}_{\alpha,j} \cdot s + \delta_{\text{id}x_{\alpha}=\ell} \cdot \vec{g}s + \mathbf{e}_{\alpha,j} \pmod{Q}$$

where $\mathbf{e}_{\alpha,j} \leftarrow \psi^d$ for $1 \leq \alpha \leq A$ and $1 \leq j \leq M$. Output the partial key-switching keys

$$\text{ksk} = \left(\{\text{id}x_{\alpha}\}_{1 \leq \alpha \leq A}, \{\mathbf{b}_{\alpha,j}\}_{1 \leq \alpha \leq A, 1 \leq j \leq M} \right).$$

- **JointKSKGen**($\{\text{ksk}_i\}_{1 \leq i \leq k}$): On input partial key-switching keys

$$\{\text{ksk}_i\}_{1 \leq i \leq k} = \left\{ \left(\{\text{id}x_{\alpha}^{(i)}\}_{1 \leq \alpha \leq A}, \{\mathbf{b}_{\alpha,j}^{(i)}\}_{1 \leq \alpha \leq A, 1 \leq j \leq M} \right) \right\}_{1 \leq i \leq k},$$

run Algorithm 1.

Algorithm 1 Repetition-Based Key-Switching key Aggregation

Input: Each party's key-switching keys $\{\mathbf{b}_{\alpha,j}^{(i)}\}_{1 \leq \alpha \leq A, 1 \leq j \leq M}$ and indices $\{\text{id}x_{\alpha}^{(i)}\}_{1 \leq \alpha \leq A}$ for $1 \leq i \leq k$.

1: **for** $1 \leq i \leq k$ **do**

2: Find an index $1 \leq \alpha^* \leq A$ such that $\text{id}x_{\alpha^*}^{(i)} \neq \text{id}x_{\alpha^*}^{(j)}$ for all $1 \leq j \neq i \leq k$. If no such index exists, abort.

3: $\text{ksk}_{s_i \rightarrow \vec{s}} \leftarrow \begin{bmatrix} \sum_{1 \leq j \leq k} \mathbf{b}_{\alpha^*, \text{id}x_{\alpha^*}^{(j)}}^{(j)} \pmod{Q} \\ \mathbf{a}_{\alpha^*, \text{id}x_{\alpha^*}^{(i)}} \end{bmatrix}$

4: **end for**

Output: $\overline{\text{ksk}} = \{\text{ksk}_{s_i \rightarrow \vec{s}}\}_{1 \leq i \leq k}$

- **JointKeySwitch**($\overline{\text{ksk}}; \text{ct}_i$): On input joint key-switching key $\overline{\text{ksk}} = \{\text{ksk}_{s_j \rightarrow \vec{s}}\}_{1 \leq j \leq k}$ and the i -th party's ciphertext ct_i , compute and output $\text{ct}'_i \leftarrow \text{KeySwitch}(\text{ksk}_{s_i \rightarrow \vec{s}}; \text{ct}_i)$.

Communication Cost. Now we investigate the communication cost of the proposed method. To achieve this, we begin by analyzing the failure probability of this protocol in Lem. 1.

Lemma 1. *For the number of (maximum) parties K , the number of repetitions A , the failure probability of our key-switching key generation protocol is bounded by $K \cdot \left(1 - \left(\frac{M-1}{M}\right)^{K-1}\right)^A$.*

Proof. See Sec. B.

With this method, the parties do not need to output exponentially many partial key-switching keys while bounding the failure probability of the protocol to be negligible, *i.e.*, less than $2^{-\kappa}$. Specifically, we show that we obtain an asymptotic communication size of $O(K(\log K + \kappa))$ per party. This is only quasi-linear with respect to both K and κ . We provide a concrete estimation of A and M in Lem. 2 which minimizes AM , which is the number of partial key-switching keys each party outputs.

Lemma 2. *For the number of (maximum) parties K , we have choices of A and M whose asymptotics are $O(\log K + \kappa)$ and K , respectively, which bound the failure probability with a negligible probability $2^{-\kappa}$. Concretely,*

$$A = \left\lceil \frac{\log K + \kappa \log 2}{-e \log(1 - 1/e)} \cdot e^{-(K-1) \log(1-1/K)} \right\rceil, \quad M = K. \quad (1)$$

Proof. See Sec. B.

Computational Cost. We also demonstrate the computational cost of this method. Essentially, the key-switching key aggregation algorithm requires only k^2 additions of ciphertexts, regardless of the number of repetitions. We emphasize that this is a one-time cost before the actual computation.

During key-switching, a single key ciphertext can be converted to a ciphertext under the joint key with only a single key-switching, as the key-switching key aggregation algorithm directly outputs the key-switching keys from s_i to the joint secret $\bar{s}$.

4.2 Tree-based Method

We formally describe the tree-based non-interactive key-switching key generation protocol below. For simplicity, for an integer idx and base B , we define $\text{idx}[:\ell] = \lfloor \text{idx} / B^{L-\ell} \rfloor$.

- **KSKSetup**($K, N, Q, \vec{g}, g^{-1}$): On input the maximum number of parties K , ciphertext dimension N , ciphertext modulus Q , the gadget vector $\vec{g} \in \mathbb{Z}^d$, and the corresponding decomposition function $g^{-1} : R_Q \rightarrow R^d$, choose a cryptographic hash function H modeled as a random oracle, and base B and height L of the tree. Output $\text{crs}_{\text{ksk}} \leftarrow \{0, 1\}^\lambda$.

- **KSKGen**(pp, s): On input public parameters, which include the gadget vector $\vec{g}$, the noise distribution ψ , base B , height L and the CRS crs_{ksk} , and the secret key s , picks an index $0 \leq \text{idx} < B^L$. Then, compute

$$\mathbf{b}_{\ell,j} = \begin{cases} -\mathbf{a}_{\ell,j} \cdot s + \vec{g} \cdot s + \mathbf{e}_{\ell,j} & \text{if } j = \text{idx}[:\ell] \\ -\mathbf{a}_{\ell,j} \cdot s + \mathbf{e}_{\ell,j} & \text{otherwise} \end{cases}$$

where $\mathbf{a}_{\ell,j} \leftarrow H(\text{crs}_{\text{ksk}} \parallel \ell \parallel j)$ and $\mathbf{e}_{\ell,j} \leftarrow \psi^d$ for $1 \leq \ell \leq L$ and $B \cdot \text{idx}[:\ell-1] \leq j < B(\text{idx}[:\ell-1] + 1)$. Output partial key-switching keys

$$\text{ksk} = \left(\text{idx}, \{\mathbf{b}_{\ell,j}\}_{1 \leq \ell \leq L, 0 \leq j < B^\ell} \right).$$

- $\text{JointKSKGen}(\{\text{ksk}_i\}_{1 \leq i \leq k})$: On input key-switching keys

$$\{\text{ksk}_i\}_{1 \leq i \leq k} = \left\{ \left(\text{idx}^{(i)}, \left\{ \mathbf{b}_{\ell,j}^{(i)} \right\}_{1 \leq \ell \leq L, 0 \leq j < B^\ell} \right) \right\}_{1 \leq i \leq k},$$

run Algorithm 2.

Algorithm 2 Tree-Based Key-switching Key Aggregation

Input: Partial key-switching keys $\{\text{ksk}_i\}_{1 \leq i \leq k} = \left\{ \left(\text{idx}^{(i)}, \left\{ \mathbf{b}_{\ell,j}^{(i)} \right\}_{1 \leq \ell \leq L, 0 \leq j < B^\ell} \right) \right\}_{1 \leq i \leq k}$ of k parties.

- 1: Generate a subtree T of a complete tree $\hat{T}$ with M leaf nodes, which takes the indices $\{\text{idx}^{(i)}\}_{1 \leq i \leq k}$ as the leaf nodes.
- 2: $\mathcal{I} \leftarrow \emptyset$
- 3: **for** $1 \leq \ell \leq L$ **do**
- 4: **for** all node n of level ℓ inside T **do**
- 5: $j \leftarrow$ the numbering of n inside level ℓ of $\hat{T}$.
- 6: $\mathcal{J}_{\text{in}} \leftarrow \{i \mid 1 \leq i \leq k \wedge j = \text{idx}^{(i)}[\ell]\}$.
- 7: $\mathcal{J}_{\text{out}} \leftarrow \{i \mid 1 \leq i \leq k \wedge \lfloor j/B \rfloor = \text{idx}^{(i)}[\ell-1]\}$
- 8: $\mathcal{I} \leftarrow \mathcal{I} \cup \{(\mathcal{J}_{\text{in}}, \mathcal{J}_{\text{out}})\}$
- 9: $\text{ksk}_{s_{\mathcal{J}_{\text{in}}} \rightarrow s_{\mathcal{J}_{\text{out}}}} \leftarrow \left[\sum_{i \in \mathcal{J}_{\text{out}}} \mathbf{b}_{\ell,j}^{(i)} \bmod Q \right] \mathbf{a}_{\ell,j}$
- 10: **end for**
- 11: **end for**

Output: $\overline{\text{ksk}} = \{\text{ksk}_{s_{\mathcal{J}_{\text{in}}} \rightarrow s_{\mathcal{J}_{\text{out}}}}\}_{(\mathcal{J}_{\text{in}}, \mathcal{J}_{\text{out}}) \in \mathcal{I}}$

- $\text{JointKeySwitch}(\overline{\text{ksk}}; \text{ct})$: On input the joint key-switching key

$$\overline{\text{ksk}} = \{\text{ksk}_{s_{\mathcal{J}_{\text{in}}} \rightarrow s_{\mathcal{J}_{\text{out}}}}\}_{(\mathcal{J}_{\text{in}}, \mathcal{J}_{\text{out}}) \in \mathcal{I}}$$

and the i -th party's ciphertext ct , run Algorithm 3.

Note that we rely on the random oracle model (ROM) for generating the CRS, unlike repetition-based method. The reason is that number of CRS elements needed is exponential. Observe that the number of CRS elements is the same as the number of nodes in the B -ary tree, except for the root node. With M being the number of leaf nodes, there are $M + M/B + \dots + B = \frac{B}{B-1}(M-1)$ nodes in total, excluding the root node. Hence, this method requires $\frac{B}{B-1}(M-1) = O(K^2 2^\kappa)$ CRS elements. However, the majority of these CRS elements are not used in our construction. In fact, since the total number of key-switching keys given by the parties is $KB \log_B M$, at most $O(K(\log K + \kappa))$ CRS are used. Therefore, if we generate CRS elements from random oracle, we only need polynomially many queries to the random oracle.

Algorithm 3 Key-switching from a Single-Key to the Joint-Key

Input: A ciphertext $\text{ct} \in R_Q^2$ of the i -th party, indices $\{\text{idx}^{(j)}\}_{1 \leq j \leq k}$, and the key switching key $\text{ksk} = \{\text{ksk}_{s_{\mathcal{J}_{\text{in}} \rightarrow s_{\mathcal{J}_{\text{out}}}}}\}_{(\mathcal{J}_{\text{in}}, \mathcal{J}_{\text{out}}) \in \mathcal{I}}$.

```

1:  $\text{res} \leftarrow \text{ct}$ 
2: for  $\ell = L; \ell \geq 1; \ell = \ell - 1$  do
3:    $\mathcal{J}_{\text{in}} \leftarrow \{j \mid 1 \leq j \leq k \wedge \text{idx}^{(i)}[:\ell] = \text{idx}^{(j)}[:\ell]\}$ 
4:    $\mathcal{J}_{\text{out}} \leftarrow \{j \mid 1 \leq j \leq k \wedge \text{idx}^{(i)}[:\ell - 1] = \text{idx}^{(j)}[:\ell - 1]\}$ 
5:   if  $\mathcal{J}_{\text{in}} \neq \mathcal{J}_{\text{out}}$  then
6:      $\text{res} \leftarrow \text{KeySwitch}(\text{ksk}_{s_{\mathcal{J}_{\text{in}} \rightarrow s_{\mathcal{J}_{\text{out}}}}}; \text{res})$ 
7:   end if
8: end for

```

Output: $\text{res} \in R_Q^2$

Correctness. We show the correctness of Algorithm 2 and Algorithm 3. Let us first fix a party i . Then, at each level ℓ , there exists the $\text{idx}^{(i)}[:\ell]$ -th node in the subtree by definition. Then, by definition, the key-switching key assigned at such a node is defined as $\left[\frac{\sum_{j \in \mathcal{J}_{\text{out}}^{i, \ell}} \mathbf{b}_{\ell, \text{idx}^{(i)}[:\ell]}^j \bmod Q}{\mathbf{a}_{\ell, \text{idx}^{(i)}[:\ell]}} \right]$, where $\mathcal{J}_{\text{out}}^{i, \ell} = \{j \mid 1 \leq j \leq k \wedge \text{idx}^{(i)}[:\ell - 1] = \text{idx}^{(j)}[:\ell - 1]\}$ since $\text{idx}^{(i)}[:\ell - 1] = \lfloor \text{idx}^{(i)}[:\ell] / B \rfloor$ by definition. Our claim is that this is a key-switching key from the secret $s_{\mathcal{J}_{\text{in}}^{i, \ell}} = \sum_{j \in \mathcal{J}_{\text{in}}^{i, \ell}} s_j$ to $s_{\mathcal{J}_{\text{out}}^{i, \ell}} = \sum_{j \in \mathcal{J}_{\text{out}}^{i, \ell}} s_j$, for $\mathcal{J}_{\text{in}}^{i, \ell} = \{j \mid 1 \leq j \leq k \wedge \text{idx}^{(i)}[:\ell] = \text{idx}^{(j)}[:\ell]\}$. Indeed, by definition, we have

$$\begin{aligned}
\sum_{j \in \mathcal{J}_{\text{out}}^{i, \ell}} \mathbf{b}_{\ell, \text{idx}^{(i)}[:\ell]}^j + \mathbf{a}_{\ell, \text{idx}^{(i)}[:\ell]} \cdot \sum_{j \in \mathcal{J}_{\text{out}}^{i, \ell}} s_j &= \sum_{j \in \mathcal{J}_{\text{out}}^{i, \ell}} \left(\mathbf{b}_{\ell, \text{idx}^{(i)}[:\ell]}^j + \mathbf{a}_{\ell, \text{idx}^{(i)}[:\ell]} \cdot s_j \right) \\
&\approx \sum_{\substack{j \in \mathcal{J}_{\text{out}}^{i, \ell} \\ \text{idx}^{(i)}[:\ell] = \text{idx}^{(j)}[:\ell]}} \vec{g} \cdot s_j = \vec{g} \cdot \sum_{j \in \mathcal{J}_{\text{in}}^{i, \ell}} s_j \pmod{Q}.
\end{aligned}$$

The last equality comes from the observation that $\text{idx}^{(i)}[:\ell] = \text{idx}^{(j)}[:\ell]$ implies $\text{idx}^{(i)}[:\ell - 1] = \text{idx}^{(j)}[:\ell - 1]$. Notice that this, indeed, has the form of the key-switching key from $s_{\mathcal{J}_{\text{in}}^{i, \ell}}$ to $s_{\mathcal{J}_{\text{out}}^{i, \ell}}$.

This discussion implies that we can conduct key-switching from the joint key of the party set $\{j \mid 1 \leq j \leq k \wedge \text{idx}^{(i)}[:\ell] = \text{idx}^{(j)}[:\ell]\}$ to the joint key of the party set $\{j \mid 1 \leq j \leq k \wedge \text{idx}^{(i)}[:\ell - 1] = \text{idx}^{(j)}[:\ell - 1]\}$ for all $1 \leq \ell \leq L$. Therefore, inductively, we can key-switch from the joint secret of the set $\{j \mid 1 \leq j \leq k \wedge \text{idx}^{(i)}[:L] = \text{idx}^{(j)}[:L]\} = \{i\}$ to the joint secret of the set $\{j \mid 1 \leq j \leq k \wedge \text{idx}^{(i)}[:0] = \text{idx}^{(j)}[:0]\} = \{1 \leq j \leq k\}$, thereby achieving key-switching from a single-key to the final joint key.

It's worth mentioning that in the actual algorithm (Algorithm 3), key-switching is not performed if the input party set $\mathcal{J}_{\text{in}}$ and the output party set $\mathcal{J}_{\text{out}}$ are the same, for the sake of computational cost.

Communication Cost. We analyze the communication cost of the proposed tree-based method. We first compute the failure probability.

Lemma 3. *For the number of (maximum) parties K and the number of indices M , the failure probability of our key-switching key generation protocol is bounded by $K(K-1)/2M$.*

Proof. The protocol fails only when the indices from each party collide. The collision probability is given as $K(K-1)/2M$ by Lem. 6, thereby proving the claim. $\square$

With an adequate choice of B and M , the parties only need to output partial key-switching keys that are quasi-constant with respect to the number of parties while achieving a negligible failure probability of the protocol. Specifically, we obtain an asymptotic communication size of $O(\kappa + \log K)$ per party. We provide a concrete choice of B and M in Lem. 4.

Lemma 4. *For the number of (maximum) parties K , we have choices for the base B and the number of indices M , which bound the failure probability with a negligible probability $2^{-\kappa}$ and make the communication cost $O(\kappa + \log K)$. Concretely, we have $B = 3$ and $M = \lceil K(K-1)2^{\kappa-1} \rceil$.*

Proof. See Sec. B.

Computational Cost. For the setup, the evaluator has to generate a subtree of key-switching keys. Observe that the size of this subtree is asymptotically $O(K)$ since there are only K root nodes. Therefore, its generation requires only $O(K)$ additions of ciphertexts.

During key-switching, a single-key ciphertext can be converted into a ciphertext under the joint-key by following a path in the key-switching key tree. We note that the number of key-switchings is bounded by $\min\{K-1, \log_B M\}$. In the worst case, the sets $\mathcal{J}_{\text{in}}$ and $\mathcal{J}_{\text{out}}$ in Algorithm 3 should differ by one party, which results in $K-1$ key-switchings. Therefore, our key-switching requires asymptotically $O(K)$ key-switchings².

It is worth noting that a direct key-switching key from s_i to the joint key $\bar{s}$ can be generated at the cost of higher noise growth. Roughly speaking, one can simply sequentially key-switch the initial key-switching key at the root node, following the tree path. This indeed outputs a key-switching key of s_i under $\bar{s}$ with amplified noise.

4.3 Discussion

See Sec. D for further optimizations and discussions on the proposed method.

² It is worth noting that the path length from the leaf node to the root node is bounded by $\log_B M$ by construction. Therefore, if $K > \log_B M = (\kappa + \log K)/\log B$, we can further bound the number of key-switchings to $\log_B M$.

Dynamicism. MKFHE allows parties to join computation dynamically. Therefore, at some point in the computation, the evaluator may need to switch the key of a ciphertext from a joint key $s_{\mathcal{J}_1} = \sum_{i \in \mathcal{J}_1} s_i$ to another joint key $s_{\mathcal{J}_2} = \sum_{i \in \mathcal{J}_2} s_i$, for some sets of parties $\mathcal{J}_1$ and $\mathcal{J}_2$. Note that $\mathcal{J}_1$ should always be a subset of $\mathcal{J}_2$.

Let us begin with a naïve method. Consider a ciphertext $(b, a) \in R_Q^2$. Then, rather than direct key-switching, we first decompose this ciphertext into a summation of a plaintext b and $|\mathcal{J}_1|$ copies of single-key ciphertext $(0, a)$, under s_i for $i \in \mathcal{J}_1$. This interpretation is valid since

$$b + a \cdot \sum_{i \in \mathcal{J}_1} s_i = b + \sum_{i \in \mathcal{J}_1} a s_i \pmod{Q}.$$

Therefore, one can simply key-switch each single-key ciphertext $(0, a)$ under s_i to the final joint key $s_{\mathcal{J}_2}$ for all $i \in \mathcal{J}_1$, and then aggregate them altogether with b . Due to the functionality of key-switching, the result should have the phase $\approx b + a s_{\mathcal{J}_1} \pmod{Q}$ under the secret $\mathcal{J}_2$.

When using the repetition-based method, there is a more optimized way to achieve this goal by generating a single key-switching key from $s_{\mathcal{J}_1}$ to $s_{\mathcal{J}_2}$ with a few additions. Concretely, it can be generated by summing key-switching keys from s_i to $s_{\mathcal{J}_2}$ for all $i \in \mathcal{J}_1$. This is implied by the linearity of key-switching keys.

5 MKFHE with Compact Ciphertexts

In this section, we show that our novel key-switching key generation algorithm (Sec. 4) and a new joint key multiplication algorithm based on the non-interactive relinearization key generation [33] can be combined to construct an MKFHE scheme with compact ciphertext sizes. Concretely, we provide multi-key variants of the RGSW [26] and FV [11, 24] schemes. Finally, we cover a broader family of FHE schemes in Sec. 5.4.

5.1 Joint Key Multiplication

We first review the non-interactive relinearization key generation protocol from [33], which is built upon a cryptosystem called uni-encryption [14]. Given two CRS's $\mathbf{a}_{\text{rlk}}^0, \mathbf{a}_{\text{rlk}}^1 \in R_Q^d$, the relinearization key generation aims to generate the following three vectors of ring elements:

$$\mathbf{b} \approx \mathbf{a}_{\text{rlk}}^0 \cdot \bar{s}, \quad \mathbf{d} \approx -\mathbf{a}_{\text{rlk}}^0 \cdot \bar{r} + \vec{g} \cdot \bar{s}, \quad \mathbf{f} \approx -\mathbf{a}_{\text{rlk}}^1 \cdot \bar{s} + \vec{g} \cdot \bar{r}. \quad (2)$$

In [33], this tuple $(\mathbf{b}, \mathbf{d}, \mathbf{f})$ was only used to perform relinearization for FV-like FHE schemes. However, we observe that this can be used to multiply the secret key $\bar{s}$ homomorphically to an RLWE ciphertext. Specifically, given an RLWE ciphertext $\text{ct} = (b, a) \in R_Q^2$, we compute

$$\text{ct}' = \mathbf{d}^\top \cdot \begin{bmatrix} g^{-1}(b) \\ g^{-1}(a) \end{bmatrix} + \begin{bmatrix} \mathbf{f}^\top \\ \mathbf{a}_{\text{rlk}}^1{}^\top \end{bmatrix} \cdot g^{-1} \left(\mathbf{a}_{\text{rlk}}^0{}^\top \cdot g^{-1}(b) + \mathbf{b}^\top \cdot g^{-1}(a) \right) \pmod{Q} \quad (3)$$

where $\bar{r}$ is some (small) randomness jointly generated by the parties.

This process indeed outputs an encryption of $\bar{s} \cdot (b + a\bar{s})$ as anticipated. See Lem. 5 for the correctness of this method.

Algorithms. Now, we describe how such a relinearization key can be generated non-interactively.

- **RLKGen**(pp, s): On input public parameters pp , which include the key and noise distributions χ and ψ , and $\text{crs}_{\text{rlk}} = (\mathbf{a}_{\text{rlk}}^0, \mathbf{a}_{\text{rlk}}^1)$, and the secret key s , sample a temporary key $r \leftarrow \chi$ and error vectors $\mathbf{e}_b, \mathbf{e}_d, \mathbf{e}_f \leftarrow \psi^d$. Compute

$$\begin{aligned} \mathbf{b}_i &= \mathbf{a}_{\text{rlk}}^0 \cdot s + \mathbf{e}_b \pmod{Q} \\ \mathbf{d}_i &= -\mathbf{a}_{\text{rlk}}^0 \cdot r + \vec{g} \cdot s + \mathbf{e}_d \pmod{Q} \\ \mathbf{f}_i &= -\mathbf{a}_{\text{rlk}}^1 \cdot s + \vec{g} \cdot r + \mathbf{e}_f \pmod{Q}, \end{aligned}$$

and output $\text{rlk} = (\mathbf{b}, \mathbf{d}, \mathbf{f})$.

- **JointRLKGen**($\{\text{rlk}_i\}_{1 \leq i \leq k}$): On input collection of relinearization keys $\text{rlk}_i = (\mathbf{b}_i, \mathbf{d}_i, \mathbf{f}_i)$, compute

$$\mathbf{b} = \sum_{1 \leq i \leq k} \mathbf{b}_i, \quad \mathbf{d} = \sum_{1 \leq i \leq k} \mathbf{d}_i, \quad \mathbf{f} = \sum_{1 \leq i \leq k} \mathbf{f}_i \pmod{Q},$$

and output $\overline{\text{rlk}} = (\mathbf{b}, \mathbf{d}, \mathbf{f})$.

- **MultJointKey**($\overline{\text{rlk}}; \text{ct}$): On input the relinearization key $\overline{\text{rlk}} = (\mathbf{b}, \mathbf{d}, \mathbf{f})$ and an RLWE ciphertext $\text{ct} = (b, a) \in R_Q^2$, compute Eq. (3).

Correctness. We show the correctness of the joint key multiplication algorithm below.

Lemma 5 (Joint Key Multiplication). *Given a relinearization key $\overline{\text{rlk}}$ and an RLWE ciphertext $\text{ct} = (b, a) \in R_Q^2$, let $\text{ct}' = (b', a') \leftarrow \text{MultJointKey}(\overline{\text{rlk}}; \text{ct})$. Then, $b' + a'\bar{s} = (b + a\bar{s}) \cdot \bar{s} + e \pmod{Q}$ for some small $e \in R$ such that $\|e\|_\infty \leq 2dKB\psi N(1 + KNB_\chi)$.*

Proof. See Sec. B.

5.2 Multi-Key RGSW

Building upon the joint key multiplication algorithm Sec. 5.1, we can construct a multi-key RGSW scheme. Recall that the RGSW encryption of the plaintext $\mu \in R_Q$ is composed of two RLEV encryptions of μ and $\mu\bar{s}$, respectively. The RLEV encryption of μ can be obtained by direct extension of Sec. 4; On the other hand, the RLEV encryption of $\mu\bar{s}$ can be easily computed by applying the joint key multiplication algorithm (Sec. 5.1) to the RLEV encryption of μ .

We provide the exact algorithms of our multi-key RGSW scheme below.

• **MKGSW.Setup**($1^\lambda, 1^K$): On input security parameter λ and the maximum number of parties K , choose the ciphertext dimension N , the ciphertext modulus Q , key and error distributions χ and ψ , a gadget vector $\vec{g} \in \mathbb{Z}^d$, and the corresponding decomposition $g^{-1} : R_Q \rightarrow R^d$. Let

- $\text{crs}_{\text{rlk}} = (\mathbf{a}_{\text{rlk}}^0, \mathbf{a}_{\text{rlk}}^1)$ where $\mathbf{a}_{\text{rlk}}^0, \mathbf{a}_{\text{rlk}}^1 \xleftarrow{\$} R_Q^d$
- $\text{crs}_{\text{ksk}} \leftarrow \text{KSKSetup}(K, N, Q, \vec{g}, g^{-1})$
- $\text{crs}_{\text{pk}} = a_{\text{pk}}$ for $a_{\text{pk}} \xleftarrow{\$} R_Q$

and output the public parameter $\text{pp} = (K, N, Q, \chi, \psi, \vec{g}, g^{-1}, \text{crs}_{\text{rlk}}, \text{crs}_{\text{ksk}}, \text{crs}_{\text{pk}})$.

• **MKGSW.KeyGen**(pp): On input public parameters pp , sample $s \leftarrow \chi$, and $e_{\text{pk}} \leftarrow \psi$. Let

- $\text{pk} = b_{\text{pk}} = -a_{\text{pk}}s + e_{\text{pk}} \bmod Q$
- $\text{rlk} \leftarrow \text{RLKGen}(\text{pp}, s)$
- $\text{ksk} \leftarrow \text{KSKGen}(\text{pp}, s)$

and output the tuple $(\text{sk} = s, \text{evk} = (\text{rlk}, \text{ksk}), \text{pk})$.

• **MKGSW.Enc**($\text{pk}; m$): On input public key $\text{pk} = b_{\text{pk}} \in R_Q$ and the plaintext $m \in R_Q$, sample $\mathbf{r} \leftarrow \chi^d$, $\mathbf{e}_0, \mathbf{e}_1 \leftarrow \psi^d$ and output $\text{ct} = (b_{\text{pk}}, a_{\text{pk}}) \cdot \mathbf{r}^\top + \begin{bmatrix} \vec{g}^\top \cdot m + \mathbf{e}_0^\top \\ \mathbf{e}_1^\top \end{bmatrix} \bmod Q$.

• **MKGSW.Extend**($\overline{\text{rlk}}, \overline{\text{ksk}}; \text{ct}$): On input relinearization and key-switching keys $\overline{\text{rlk}}$ and $\overline{\text{ksk}}$, and ciphertext ct of the i -th party, run:

- Parse $\text{ct} = [\text{ct}_1 \mid \dots \mid \text{ct}_d]$ where each $\text{ct}_j \in R_Q^2$ for $1 \leq j \leq d$.
- Run $\overline{\text{ct}}_j \leftarrow \text{JointKeySwitch}(\overline{\text{ksk}}; \text{ct}_j)$ for $1 \leq j \leq d$.
- Run $\overline{\text{ct}}_{j+d} \leftarrow \text{MultJointKey}(\overline{\text{rlk}}; \overline{\text{ct}}_j)$.
- Let $\overline{\text{ct}} := [\overline{\text{ct}}_1 \mid \dots \mid \overline{\text{ct}}_{2d}]$.

• **MKGSW.Eval**($C, \{(\text{evk}_i, \text{pk}_i, \text{ct}_i)\}_{1 \leq i \leq k}$):

1. On input $\text{evk}_i = (\text{rlk}_i, \text{ksk}_i)$ for $1 \leq i \leq k$,
 - $\overline{\text{rlk}} \leftarrow \text{JointRLKGen}(\{\text{rlk}_i\}_{1 \leq i \leq k})$
 - $\overline{\text{ksk}} \leftarrow \text{JointKSKGen}(\{\text{ksk}_i\}_{1 \leq i \leq k})$
2. For all $1 \leq i \leq k$, let $\overline{\text{ct}}_i \leftarrow \text{MKGSW.Extend}(\overline{\text{rlk}}, \overline{\text{ksk}}; \text{ct}_i)$.
3. Homomorphically evaluate the circuit C based on the following algorithms.
 - **MKGSW.Add**($\overline{\text{ct}}, \overline{\text{ct}}'$): On input ciphertexts $\overline{\text{ct}}$ and $\overline{\text{ct}}'$, compute and output $\text{res} = \overline{\text{ct}} + \overline{\text{ct}}' \pmod{Q}$.
 - **MKGSW.Sub**($\overline{\text{ct}}, \overline{\text{ct}}'$): On input ciphertexts $\overline{\text{ct}}$ and $\overline{\text{ct}}'$, compute and output $\text{res} = \overline{\text{ct}} - \overline{\text{ct}}' \pmod{Q}$.
 - **MKGSW.Mult**($\overline{\text{ct}}, \overline{\text{ct}}'$): On input ciphertexts $\overline{\text{ct}} = (c_{i,j})_{1 \leq i \leq 2, 1 \leq j \leq 2d}$ and $\overline{\text{ct}}' = (c'_{i,j})_{1 \leq i \leq 2, 1 \leq j \leq 2d}$, compute $\overline{\text{ct}} \cdot g^{-1}(\text{ct}') \pmod{Q}$.
4. Return the output multi-key GSW ciphertext $\overline{\text{ct}}$.

- $\text{MKGSW.Dec}(\{\text{sk}_i\}_{1 \leq i \leq k}; \overline{\text{ct}})$: On input secret keys $\text{sk}_i = s_i$ for $1 \leq i \leq k$ and a ciphertext $\overline{\text{ct}} \in R_Q^2$, find $\mu \in \mathbb{B}$ that minimizes

$$[1 \ \bar{s}] \cdot \text{ct} - \mu \cdot \vec{g} \otimes [1 \ \bar{s}] \pmod{Q}.$$

Security. The security of our multi-key RGSW scheme relies on the hardness of the RLWE assumption, and circular security (of the RLEV encryption). The proposed scheme builds upon three different ingredients: (public-key) RLEV encryptions, the partial relinearization key (Sec. 5.1), and the partial key-switching keys (Sec. 4). The public-key RLEV encryptions and partial key-switching keys assume only the hardness of the RLWE assumption. The partial relinearization key assumes a form of circular security, as it encrypts the temporary key r_i under s_i , and s_i under r_i using the same randomness to a zero encryption under s_i . When considering bootstrapping, we additionally assume that RLEV encryptions of the binary decomposition of the secret key s_i are provided. This reduces to another form of the circular security assumption.

Correctness. Below, we informally show the correctness of the multi-key GSW scheme.

Theorem 1 (Correctness). *Let B_m be the one-norm bound of the input messages. Then, a circuit C with depth d_C can be evaluated using the proposed multi-key GSW scheme if $Q = \tilde{O}(B_m^{d_C} B_\chi^2 B_\psi K^3 N^2)$.*

Proof. By Lem. 13, a multi-key GSW ciphertext has the noise bounded by $3dB_\chi^2 B_\psi K^3 N^2$. We can choose the gadget parameters $B = O(1)$ and $d = \log Q$. Then, the noise asymptotic becomes $O(B_\chi^2 B_\psi K^3 N^2 \log Q)$. Since the addition increases the noise only additively and the multiplication amplifies the noise by the one-norm of the underlying plaintext, the final noise after the evaluation of C is asymptotically $O(B_m^{d_C} B_\chi^2 B_\psi K^3 N^2 \log Q)$. To allow correct decryption, $Q = O(B_m^{d_C} B_\chi^2 B_\psi K^3 N^2 \log Q)$. $\square$

In this analysis, we assume an arbitrary message space below in our analysis. However, typically, the GSW scheme is used to evaluate branching programs with binary data, and the noise analysis can be a tad different in this case. Nevertheless, the noise analysis of the extended RGSW ciphertext from RLEV (Lem. 13) shows only an additional $O(K^3 N \log Q)$ factor compared to a fresh GSW ciphertext, which is polynomially large. Asymptotically, this adds only an additional $\log K$ factor to the ciphertext dimension ($n = O(\lambda + \log K)$), implying the existence of secure parameters for the bootstrappable MK-GSW scheme.

5.3 Fan-Vercauteren (FV)

The FV scheme is an FHE scheme for arithmetic over R_p for some prime power p . The main distinction of the FV scheme from other schemes is that the message

is encoded in the most significant bit (MSB), and the noise is stored in the least significant bit (LSB). In other words, it encodes the message $m \in R_p$ as $\approx Q/p \cdot m$, assuming $p \mid Q$ for simplicity.³ This allows us to perform homomorphic computation on plaintexts without being interfered with by the noise.

The FV scheme naturally supports addition, subtraction, multiplication, and automorphism operations. The addition and subtraction of the FV schemes can be done by adding or subtracting each entry of the ciphertexts, and the correctness directly follows from the linearity of RLWE. To realize FV multiplication, we need to realize a special operation called *relinearization*. Briefly speaking, it key-switches a ciphertext $(c_0, c_1, c_2) \in R_Q^3$ under a tensored secret $(1, \bar{s}, \bar{s}^2)$ to the original secret $(1, \bar{s})$. This can be easily achieved by homomorphically computing $(c_0, c_1) + \bar{s} \cdot (0, c_2)$ since it decrypts to $c_0 + c_1\bar{s} + c_2\bar{s}^2$. The second term can be easily computed with our joint key multiplication algorithm Sec. 5.1.

On the other hand, the automorphism keys have a linear structure, and therefore, our novel key-switching technique can be applied here. Specifically, one can simply switch the secret of each party's (single-key) automorphism keys to the joint key and then aggregate them to generate automorphism keys for the joint key. However, this would incur a higher noise level of the switching key. Therefore, we propose using a non-interactive automorphism key generation algorithm for multi-party FHE schemes [43, 33]. Essentially, they use the same underlying idea as our key-switching key generation. Given a dedicated CRS, each party outputs their own automorphism keys with the CRS as a random masking, and then the automorphism key of the joint key is obtained by summing them all together.

We now describe our multi-key FV scheme below. For the correctness of the scheme, we refer to Sec. F.

• **MKFV.Setup**($1^\lambda, 1^K$): On input security parameter λ and the maximum number of parties K , choose the ciphertext dimension N , the plaintext and ciphertext moduli p and Q , key and error distributions χ and ψ , gadget vector $\vec{g} \in \mathbb{Z}^d$, the corresponding decomposition $g^{-1} : R_Q \rightarrow R^d$, and the automorphism index set $\mathcal{I}_{\text{aut}} \subset \mathbb{Z}_{2N}^\times$. Let

- $\text{crs}_{\text{rlk}} = (\mathbf{a}_{\text{rlk}}^0, \mathbf{a}_{\text{rlk}}^1)$ where $\mathbf{a}_{\text{rlk}}^0, \mathbf{a}_{\text{rlk}}^1 \xleftarrow{\$} R_Q^d$
- $\text{crs}_{\text{atk}} = (\mathbf{a}_{\text{atk}}^i)_{i \in \mathcal{I}_{\text{aut}}}$ where $\mathbf{a}_{\text{atk}}^i \xleftarrow{\$} R_Q^d$
- $\text{crs}_{\text{ksk}} \leftarrow \text{KSKSetup}(K, N, Q, \vec{g}, g^{-1})$
- $\text{crs}_{\text{pk}} \leftarrow a_{\text{pk}}$ for $a_{\text{pk}} \xleftarrow{\$} R_Q$

and output the public parameter $\text{pp} = (K, N, p, Q, \chi, \psi, \vec{g}, g^{-1}, \mathcal{I}_{\text{aut}}, \text{crs}_{\text{rlk}}, \text{crs}_{\text{atk}}, \text{crs}_{\text{ksk}}, \text{crs}_{\text{pk}})$.

• **MKFV.KeyGen**(pp): On input public parameters pp , sample $s \leftarrow \chi$ and $e_{\text{pk}} \leftarrow \psi$. Let

- $\text{pk} = b_{\text{pk}} = -a_{\text{pk}}s + e_{\text{pk}} \bmod Q$.

³ We note that the FV scheme can be instantiated even when p does not divide Q . In this case, the only difference is that the encoding is now $\lfloor Q/p \cdot m \rfloor$ instead of $Q/p \cdot m$.

- $\text{rlk}_i \leftarrow \text{RLKGen}(\text{pp}, s)$.
- Sample $\mathbf{e}_j \leftarrow \psi^d$ and compute $\mathbf{b}_j = -\mathbf{a}_{\text{atk}}^j \cdot s + \vec{g}s(X^j) + \mathbf{e}_j \bmod Q$ for all $j \in \mathcal{I}_{\text{aut}}$. Let $\text{atk} \leftarrow \{\mathbf{b}_j\}_{j \in \mathcal{I}_{\text{aut}}}$.
- $\text{ksk} \leftarrow \text{KSKGen}(\text{pp}, s)$.

and output the tuple $(\text{sk} = s, \text{evk}_i = (\text{rlk}, \text{atk}, \text{ksk}), \text{pk})$.

• **MKFV.Enc**($\text{pk}; m$): On input public key $\text{pk} = b_{\text{pk}} \in R_Q$ and the plaintext $m \in R_p$, sample $r \leftarrow \chi$, $e_0, e_1 \leftarrow \psi$ and output $\text{ct} = r \cdot (b_{\text{pk}}, a_{\text{pk}}) + (Q/p \cdot m + e_0, e_1) \bmod Q$.

• **MKFV.Eval**($C, \{(\text{evk}_i, \text{pk}_i, \text{ct}_i)\}_{1 \leq i \leq k}$):

1. On input $\text{evk}_i = (\text{rlk}_i, \text{atk}_i, \text{ksk}_i)$ for $1 \leq i \leq k$,
 - $\overline{\text{rlk}} \leftarrow \text{JointRLKGen}(\{\text{rlk}_i\}_{1 \leq i \leq k})$
 - Parse $\text{atk}_i = \{\mathbf{b}_{i,j}\}_{j \in \mathcal{I}_{\text{aut}}}$ for all $1 \leq i \leq k$, and compute $\overline{\text{atk}}_j \leftarrow \left(\sum_{1 \leq i \leq k} \mathbf{b}_{i,j}, \mathbf{a}_{\text{atk}}^j \right)$ for all $j \in \mathcal{I}_{\text{aut}}$. Let $\overline{\text{atk}} = \{\overline{\text{atk}}_j\}_{j \in \mathcal{I}_{\text{aut}}}$.
 - $\overline{\text{ksk}} \leftarrow \text{JointKSKGen}(\{\text{ksk}_i\}_{1 \leq i \leq k})$.
2. Run $\overline{\text{ct}}_i \leftarrow \text{JointKeySwitch}(\overline{\text{ksk}}; \text{ct}_i)$ for all $1 \leq i \leq k$.
3. Homomorphically evaluate the circuit C based on the following algorithms.
 - **MKFV.Add**($\overline{\text{ct}}, \overline{\text{ct}}'$): On input ciphertexts $\overline{\text{ct}} = (b, a)$ and $\overline{\text{ct}}' = (b', a')$, compute and output $\text{res} = (b + b', a + a') \bmod Q$.
 - **MKFV.Sub**($\overline{\text{ct}}, \overline{\text{ct}}'$): On input ciphertexts $\overline{\text{ct}} = (b, a)$ and $\overline{\text{ct}}' = (b', a')$, compute and output $\text{res} = (b - b', a - a') \bmod Q$.
 - **MKFV.Mult**($\overline{\text{rlk}}; \overline{\text{ct}}, \overline{\text{ct}}'$): On input a joint relinearization key $\overline{\text{rlk}}$, and ciphertexts $\overline{\text{ct}} = (b, a)$ and $\overline{\text{ct}}' = (b', a')$, compute and output

$$\begin{aligned} \text{res} = & \left(\left\lfloor \frac{p}{Q} \cdot \hat{b}\hat{b}' \right\rfloor \bmod Q, \left\lfloor \frac{p}{Q} \cdot (\hat{a}\hat{b}' + \hat{a}'\hat{b}) \right\rfloor \bmod Q \right) \\ & + \text{MultJointKey} \left(\overline{\text{rlk}}, \left(0, \left\lfloor \frac{p}{Q} \cdot \hat{a}\hat{a}' \right\rfloor \right) \right) \end{aligned}$$

where $\hat{a}, \hat{b}, \hat{a}', \hat{b}'$ are the liftings of a, b, a', b' to the ring R , respectively ⁴.

- **MKFV.Aut**($\overline{\text{atk}}_i; \overline{\text{ct}}$): On input a joint automorphism key $\overline{\text{atk}}_i$ for the index $i \in \mathcal{I}_{\text{aut}}$ and the ciphertext $\overline{\text{ct}} = (\bar{b}, \bar{a})$, compute and output

$$\overline{\text{res}} = \begin{bmatrix} \bar{b}(X^i) \\ 0 \end{bmatrix} + \overline{\text{atk}}_i \cdot g^{-1}(\bar{a}(X^i)) \pmod{q}.$$

4. Return the output multi-key FV ciphertext ct .

• **MKFV.Dec**($\{\text{sk}_i\}_{1 \leq i \leq k}; \overline{\text{ct}}$): On input secret keys $\text{sk}_i = s_i$ for $1 \leq i \leq k$ and a ciphertext $\overline{\text{ct}} \in R_Q^2$, output $\mu = \left\lfloor \frac{p}{Q} \cdot [1 \ \bar{s}] \cdot \overline{\text{ct}} \right\rfloor \in R_p$.

⁴ In practice, the zero-body does not need to be decomposed for the computation of Eq. (3).

Security. The security of our multi-key FV scheme relies on the hardness of the RLWE assumption and circular security. Similar to the multi-key RGSW scheme, the ciphertext itself is independent of the secret, and hence its security follows from the conventional RLWE assumption. The evaluation key of the proposed scheme additionally consists of the automorphism key compared to the multi-key GSW scheme. However, unlike the multi-key RGSW scheme, the FV scheme does not additionally require bootstrapping keys (unless one utilizes the sparse key encapsulation technique [9]).

Discussion. This method can be applied to the CKKS [19] or BGV [12] schemes almost straightforwardly. In these schemes, an additional noise management technique called the special modulus is used in addition to the gadget decomposition. Since this technique preserves linearity, it can be easily integrated with our technique as well. In this work, we only focus on the FV scheme for simplicity.

5.4 Generic Compiler for Transforming FHE to MKFHE

In this section, we provide a compiler that can be used to build multi-key variants of other FHE schemes besides the RGSW or FV (and BGV/CKKS) schemes. Specifically, we provide a generic transformation from a single-key FHE to a multi-key variant with compact ciphertext sizes. Our transform works for a family of FHE schemes that we call *compilable FHE*.

Definition 3 (Compilable FHE). *For a positive integer $K > 0$, a fully homomorphic encryption scheme FHE is compilable if it satisfies the following properties for any $k \leq K$:*

1. *It accepts the secret key $s = \sum_{i=1}^k s_i$ where $s_i \leftarrow \chi$.*
2. *The ciphertext is an RLWE encryption whose plaintext is independent of the secret key.*
3. *The evaluation key is a set of RGSW encryptions of $C_{\text{evk}}(f_{\text{evk}}(s_1), \dots, f_{\text{evk}}(s_k))$ where $f_{\text{evk}} : R \rightarrow R_Q^\alpha$ is a function independent of k (only dependent to K), and C_{evk} is a circuit with a depth $d_{C_{\text{evk}}}$.*
4. *The correctness of the scheme is guaranteed with ciphertexts and an evaluation key with noise size*

$$O(B_\chi B_\psi K^2 N \log Q) \quad \text{and} \quad O(B_f^{d_c} B_\chi^2 B_\psi K^3 N^2 \log Q),$$

respectively, where B_f is the one-norm bound of the function f .

where B_χ and B_ψ are the probabilistic bounds of some distributions χ and ψ over R .

Now, given a compilable FHE scheme $\text{FHE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Eval}, \text{Dec})$, we can construct a multi-key FHE scheme MKFHE as follows.

- **MKFHE.Setup**($1^\lambda, 1^K$): On input security parameter λ and the maximum number of parties K , run and output $\text{pp} \leftarrow \text{MKGSW.Setup}(1^\lambda, 1^K)$.
- **MKFHE.KeyGen**(pp): Compute $(\text{sk}^0, \text{evk}^0 = (\text{rlk}, \text{ksk}), \text{pk}^0) \leftarrow \text{MKGSW.KeyGen}(\text{pp})$, and let $(b_1, \dots, b_\beta) \leftarrow f_{\text{evk}}(\text{sk}^0)$. Run $\text{evk}_j^1 \leftarrow \text{MKGSW.Enc}(\text{pk}^0; b_j)$ for $1 \leq j \leq \beta$, and output

$$(\text{sk} = \text{sk}^0, \text{evk} = (\text{evk}^0, \text{evk}^1 = (\text{evk}_1^1, \dots, \text{evk}_\beta^1)), \text{pk} = \text{pk}^0).$$

- **MKFHE.Enc**($\text{pk}; m$): Run $\text{FHE.Encrypt}(\text{pk}, m)$.
- **MKFHE.Eval**($C, \{\text{evk}_i, \text{pk}_i, \text{ct}_i\}_{1 \leq i \leq K}$):
 1. On input $\text{evk}_i = (\text{evk}_i^0, \text{evk}_i^1)$ for $1 \leq i \leq k$,
 - $\overline{\text{evk}} \leftarrow \text{MKGSW.Eval}(C_{\text{evk}}, \{(\text{evk}_i^0, \text{pk}_i^0, \text{evk}_i^1)\}_{1 \leq i \leq k})$.
 - $\overline{\text{ksk}} \leftarrow \text{JointKSKGen}(\{\text{ksk}_i\}_{1 \leq i \leq k})$.
 - $\overline{\text{pk}} \leftarrow (\sum_{1 \leq i \leq k} \text{pk}_i, a_{\text{pk}}) \pmod{Q}$.
 2. For all $1 \leq i \leq k$, let $\overline{\text{ct}}_i \leftarrow \text{JointKeySwitch}(\overline{\text{ksk}}; \text{ct}_i)$.
 3. Return the output ciphertext $\overline{\text{ct}} \leftarrow \text{FHE.Eval}(C, \overline{\text{evk}}, \overline{\text{pk}}, \{\overline{\text{ct}}_i\}_{1 \leq i \leq k})$.
- **MKFHE.Dec**($\{\text{sk}_i\}_{1 \leq i \leq k}; \overline{\text{ct}}$): On input secret keys $\text{sk}_i = s_i$ for $1 \leq i \leq k$ and a ciphertext $\overline{\text{ct}} \in R_Q^2$, output $\text{FHE.Dec}(\overline{\text{sk}} = \sum_{1 \leq i \leq k} \text{sk}_i, \overline{\text{ct}})$.

Theorem 2 (MKFHE Compiler). *If FHE is compilable, then MKFHE, compiled as above, is an MKFHE scheme under the RLWE assumption and circular security.*

Proof (Sketch). Correctness. The compiled multi-key FHE scheme MKFHE is a valid multi-key scheme. Let us provide a brief proof sketch. Since the plaintext of the FHE ciphertext is independent of the secret key, the ciphertexts ct_i 's should still count as a valid encryption of FHE under the joint secret $\overline{\text{sk}}$. Also, by the correctness of the multi-key GSW scheme, $\overline{\text{evk}}$ is a set of valid RGSW encryptions of $C_{\text{evk}}(f_{\text{evk}}(s_1), \dots, f_{\text{evk}}(s_k))$ under $\overline{\text{sk}}$, which is the evaluation key of FHE under the joint secret $\overline{\text{sk}}$.

Now, since a public key encryption has a noise size $O(B_\chi B_\psi N)$ by Lem. 12, the noise size of the ciphertexts $\overline{\text{ct}}_i$ after the key-switching is bounded by $O(B_\chi B_\psi K^2 N \log Q)$. On the other hand, by Thm. 1, the evaluation key $\overline{\text{evk}}$ is a set of multi-key RGSW ciphertexts with a noise bound $O(B_f^{dc} B_\chi^2 B_\psi K^3 N^2 \log Q)$. As FHE accepts the joint secret $\overline{\text{sk}}$ as its secret, and the ciphertexts and evaluation keys with these noise bounds by definition, the FHE computation with these ciphertexts and evaluation keys is correct. Therefore, $\overline{\text{ct}}$ decrypts to a correct plaintext with the multi-key decryption algorithm.

Security. The security of this compiled multi-key scheme relies on the security of the multi-key RGSW scheme (RLWE assumption and the circular security assumption) and yet another circular security assumption. Specifically, we additionally assume that the RLWE assumption with a secret s_i remains secure even with outputting RLEV encryptions of $f_{\text{evk}}(s_i)$.

Discussion. In the definition of compilable FHE, we state that the evaluation key is a set of RGSW encryptions. However, it is worth noting that any RLWE-based encryption should work since there are efficient conversions from RGSW to any RLWE-based encryption. We also note that the evaluation keys do not necessarily have to be generated from the circuit evaluation. For example, a handful of schemes rely on temporary secret keys for more efficient bootstrapping [20, 9], which cannot be captured with the current syntax. In this case, the parties may output additional key material that depends on this temporary secret. Alternatively, the automorphism keys can be generated in the same manner as the multi-key FV Sec. 5.3 in a non-interactive manner.

Nevertheless, many widely-used FHE schemes are compilable and therefore can be transformed into a multi-key variant exploiting our compiler. As a concrete example, we describe how the evaluation key generation circuit can be composed for AP-like schemes (LMKCDEY [36], TFHE [20, 30], and Carousel [29]) in Sec. G.

5.5 Discussion

Threshold FHE. Our construction can be naturally extended to a t -out-of- n threshold FHE scheme. When $t = n$, i.e. in the case of MPFHE, the adaptation is natural. For general t , we can adapt the share-reshare protocol [42] based on a linear secret sharing scheme, where each party distributes a t -out-of- n share to other parties using a secret channel, then aggregates them to agree on a final share. This one round protocol can be concurrently run with the first broadcast round of our MKFHE scheme, achieving non-interactive key generation.

Distributed Decryption. To decrypt a ciphertext, parties should engage in a *distributed decryption* protocol. We describe a simple, widely used solution based on the *noise flooding* technique that consumes one round. For the output ciphertext $\text{ct} = (b, a)$, each party broadcasts $d_i = as_i + f_i \pmod{Q}$, where f_i is the flooding noise that can statistically erase the noise of ct . After receiving the partial decryptions, parties can recover the plaintext by computing $b + \sum_i d_i \pmod{Q}$. For other constructions, we refer to [32, 10, 55, 53] for more details.

Malicious Security. Using a one-round distributed decryption protocol, our MKFHE scheme directly implies a two-round MPC protocol for arbitrary circuits. As our construction only depends on the standard (circular) RLWE assumption, this protocol can be made maliciously secure by attaching a non-interactive zero-knowledge proof of well-formedness of public materials [27], following the blueprint of [44].

6 Experimental Results

We implement our multi-key FV scheme (Sec. 5.3) and report performance to demonstrate the correctness and efficiency of the proposed method. We also

compare our scheme with the state-of-the-art multi-key FV scheme from [31], which we will refer to as the KKLSS scheme. Our implementation is based on the HIENAA library [28], and for the KKLSS scheme, we compare with the lattigo-based implementation provided by the authors of [31]. All benchmarks were measured in a single-thread setting on a desktop equipped with the Intel i9-11900 CPU and 128GB of RAM. Our implementation is available at <https://github.com/SNUCP/compact-mkfhe>.

Parameter Setting. We match the parameters with [31]. Concretely, we set the ring degree $N = 2^{14}$ and $\log Q = 5 \times 53 + 54 = 319$ bits. The key distribution χ is a ternary distribution where 1 and -1 are sampled with probability $1/4$, and the noise distribution ψ is a rounded Gaussian distribution with a standard deviation of $\sigma = 3.2$. The gadget decomposition is chosen as the RNS decomposition.

Also, for a fair comparison, we use an additional noise management technique called the *special modulus*. Roughly speaking, the key-switching is conducted over the ring R_{PQ} instead of R_Q , and later the ciphertext is scaled by P to reduce the size of the noise introduced from the key-switching. For a more detailed discussion of special modulus, we refer to [31, Appendix B]. We set $\log P = 2 \times 60 = 120$ bits.

Finally, for collision probability needed for our key-switching key generation Sec. 4, we set $\kappa = 40$.

Performance. We benchmark the performance of the main algorithms and key size of our multi-key FV scheme. First, we measure the evaluation key size and the latency of the joint key generation, the key-switching algorithms from Sec. 4, for both the repetition-based method (Sec. 4.1) and the tree-based method (Sec. 4.2). The evaluation key includes the relinearization key, the key-switching keys, and the automorphism key for the automorphism $X \mapsto X^5$. For the repetition-based method (Sec. 4.1), the repetition parameters A and M were set using the formula from Lem. 2 with respect to the number of parties K . For the communication-friendly method Sec. 4.2, we set the index pool size M based on Lem. 4 and choose the base $B = 2$ for simplicity. We also measure the performance of multiplication and automorphism operations, which are the most time-consuming algorithms in the FV scheme.

As anticipated, the communication cost grows linearly for the repetition-based method and remains almost constant for the tree-based method. The key-switching latency with the repetition-based method remains the same regardless of K , and the key-switching latency with the tree-based method grows logarithmically with K . Moreover, the multiplication and automorphism performances remain unchanged even with many parties.

Comparison. We now compare the performance of the multiplication and automorphism algorithms of our multi-key FV scheme and the KKLSS scheme.

K		4	8	12	16
Repetition-based (Sec. 4.1)	Key Size	586MiB	1266MiB	1986MiB	2685MiB
	JointKeyGen	8ms	20ms	37ms	61ms
	KeySwitch	4ms			
Tree-based (Sec. 4.2)	Key Size	232MiB	242MiB	252MiB	262MiB
	JointKeyGen	23ms	53ms	83ms	113ms
	KeySwitch	10ms	14ms	17ms	18ms
Mult		30ms			
Aut		5ms			

Table 1. The performances of the joint key generation, the key switching from a single key to the joint key, the multiplication, and the automorphism algorithms. K denotes the number of the parties.

Unlike the prior state-of-the-art MKFV scheme [31], for which the multiplication and automorphism performances grow linearly with the number of parties, the latency of our multiplication and automorphism algorithms remains constant. Consequently, when $K = 16$, our multiplication and automorphism performances become $30.6\times$ and $60.6\times$ times faster compared to those of the KKLSS scheme.

K	KKLSS [31]		Ours		Speedup	
	Mult	Aut	Mult	Aut	Mult	Aut
4	239ms	77ms	30ms	5ms	$8.0\times$	$15.4\times$
8	467ms	153ms			$15.6\times$	$30.6\times$
12	692ms	228ms			$23.1\times$	$45.6\times$
16	918ms	303ms			$30.6\times$	$60.6\times$

Table 2. Performance of multiplication and automorphism algorithms of our multi-key FV scheme and the KKLSS scheme. K denotes the number of participants.

It is worth noting that our multiplication algorithm is still faster even when the pre-processing phase, namely the joint key generation and key-switching, is taken into account. For example, for $K = 4$, a single multiplication with the pre-processing takes 42ms and 63ms with the repetition-based method and the tree-based method, respectively. This is still $5.7\times$ and $3.8\times$ times faster than the KKLSS multiplication. Furthermore, this pre-processing needs to be performed only once at the start of the evaluation, making the speedup better in deeper circuits.

References

1. Afshar, A., Goyal, R.: Fully-succinct multi-key fhe and rate-1 simulatable threshold decryption from lwe. In: Heninger, N., Rosulek, M. (eds.) *Advances in Cryptology – CRYPTO 2026*. pp. 591–623. Springer Nature Switzerland, Cham (2026)
2. Akin, Y., Klemsa, J., Önen, M.: A practical TFHE-based multi-key homomorphic encryption with linear complexity and low noise growth. In: Tsudik, G., Conti, M., Liang, K., Smaragdakis, G. (eds.) *ESORICS 2023, Part I*. LNCS, vol. 14344, pp. 3–23. Springer, Cham (Sep 2023). https://doi.org/10.1007/978-3-031-50594-2_1
3. Albrecht, M.R., Bai, S., Ducas, L.: A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In: Robshaw, M., Katz, J. (eds.) *CRYPTO 2016, Part I*. LNCS, vol. 9814, pp. 153–178. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_6
4. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) *CRYPTO 2014, Part I*. LNCS, vol. 8616, pp. 297–314. Springer, Berlin, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44371-2_17
5. 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: Pointcheval, D., Johansson, T. (eds.) *EUROCRYPT 2012*. LNCS, vol. 7237, pp. 483–501. Springer, Berlin, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_29
6. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M.R., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Shacham, H., Boldyreva, A. (eds.) *CRYPTO 2018, Part I*. LNCS, vol. 10991, pp. 565–596. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_19
7. Bonte, C., Iliashenko, I., Park, J., Pereira, H.V.L., Smart, N.P.: FINAL: Faster FHE instantiated with NTRU and LWE. In: Agrawal, S., Lin, D. (eds.) *ASIACRYPT 2022, Part II*. LNCS, vol. 13792, pp. 188–215. Springer, Cham (Dec 2022). https://doi.org/10.1007/978-3-031-22966-4_7
8. Bootland, C., Cong, K., Demmler, D., Frederiksen, T.K., Libert, B., Orfila, J.B., Rotaru, D., Smart, N.P., Tanguy, T., Tap, S., Walter, M.: Threshold (fully) homomorphic encryption. *Cryptology ePrint Archive*, Report 2025/699 (2025), <https://eprint.iacr.org/2025/699>
9. Bossuat, J.P., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In: Ateniese, G., Venturi, D. (eds.) *ACNS 2022*. LNCS, vol. 13269, pp. 521–541. Springer, Cham (Jun 2022). https://doi.org/10.1007/978-3-031-09234-3_26
10. Boudgoust, K., Scholl, P.: Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. In: Guo, J., Steinfeld, R. (eds.) *ASIACRYPT 2023, Part I*. LNCS, vol. 14438, pp. 371–404. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8721-4_12
11. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) *CRYPTO 2012*. LNCS, vol. 7417, pp. 868–886. Springer, Berlin, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_50

12. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012. pp. 309–325. ACM (Jan 2012). <https://doi.org/10.1145/2090236.2090262>
13. Brakerski, Z., Perlman, R.: Lattice-based fully dynamic multi-key FHE with short ciphertexts. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 190–213. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_8
14. Chen, H., Chillotti, I., Song, Y.: Multi-key homomorphic encryption from TFHE. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part II. LNCS, vol. 11922, pp. 446–472. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-34621-8_16
15. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 395–412. ACM Press (Nov 2019). <https://doi.org/10.1145/3319535.3363207>
16. Chen, L., Zhang, Z., Wang, X.: Batched multi-hop multi-key FHE from ring-LWE with compact ciphertext extension. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part II. LNCS, vol. 10678, pp. 597–627. Springer, Cham (Nov 2017). https://doi.org/10.1007/978-3-319-70503-3_20
17. Cheon, J.H., Choe, H., Kim, S., Yeo, Y.: Multi-party homomorphic encryption with dynamicity and ciphertext reusability. IACR Communications in Cryptology **3**(1) (2026). <https://doi.org/10.62056/a0qj89krz>
18. Cheon, J.H., Hanrot, G., Kim, J., Stehlé, D.: SHIP: A shallow and highly parallelizable CKKS bootstrapping algorithm. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part III. LNCS, vol. 15603, pp. 398–428. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91131-6_14
19. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 409–437. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_15
20. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 3–33. Springer, Berlin, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6_1
21. Clear, M., McGoldrick, C.: Multi-identity and multi-key leveled FHE from learning with errors. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 630–656. Springer, Berlin, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-48000-7_31
22. Dahl, M., Demmler, D., El Kazdadi, S., Meyre, A., Orfila, J.B., Rotaru, D., Smart, N.P., Tap, S., Walter, M.: Noah’s ark: Efficient threshold-fhe using noise flooding. In: Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. p. 35–46. WAHC ’23, Association for Computing Machinery, New York, NY, USA (2023). <https://doi.org/10.1145/3605759.3625259>
23. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 617–640. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_24
24. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), <https://eprint.iacr.org/2012/144>

25. Gentry, C.: Computing on encrypted data (invited talk). In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 09. LNCS, vol. 5888, p. 477. Springer, Berlin, Heidelberg (Dec 2009). https://doi.org/10.1007/978-3-642-10433-6_32
26. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Berlin, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_5
27. Hwang, I., Lee, H., Seo, J., Song, Y.: Practical zero-knowledge PIOP for maliciously secure multiparty homomorphic encryption. In: Huang, C.Y., Chen, J.C., Shieh, S.P., Lie, D., Cortier, V. (eds.) ACM CCS 2025. pp. 4049–4063. ACM Press (Oct 2025). <https://doi.org/10.1145/3719027.3765229>
28. Hwang, I., Min, S.: HIENAA. Online: <https://github.com/hienaa-org/hienaa> (2026)
29. Hwang, I., Min, S., Song, Y.: Carousel: Fully homomorphic encryption with bootstrapping over automorphism group. In: Hanaoka, G., Yang, B.Y. (eds.) ASIACRYPT 2025, Part VII. LNCS, vol. 16251, pp. 131–164. Springer, Singapore (Dec 2025). https://doi.org/10.1007/978-981-95-5122-4_5
30. Joye, M., Paillier, P.: Blind rotation in fully homomorphic encryption with extended keys. In: International symposium on cyber security, cryptology, and machine learning. pp. 1–18. Springer (2022)
31. Kim, T., Kwak, H., Lee, D., Seo, J., Song, Y.: Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 726–740. ACM Press (Nov 2023). <https://doi.org/10.1145/3576915.3623176>
32. Kraitsberg, M., Lindell, Y., Osheter, V., Smart, N.P., Talibi Alaoui, Y.: Adding distributed decryption and key generation to a ring-LWE based CCA encryption scheme. In: Jang-Jaccard, J., Guo, F. (eds.) ACISP 19. LNCS, vol. 11547, pp. 192–210. Springer, Cham (Jul 2019). https://doi.org/10.1007/978-3-030-21548-4_11
33. Kwak, H., Lee, D., Song, Y., Wagh, S.: A general framework of homomorphic encryption for multiple parties with non-interactive key-aggregation. In: Pöpper, C., Batina, L. (eds.) ACNS 2024, Part II. LNCS, vol. 14584, pp. 403–430. Springer, Cham (Mar 2024). https://doi.org/10.1007/978-3-031-54773-7_16
34. Kwak, H., Min, S., Song, Y.: Towards practical multi-key TFHE: Parallelizable, key-compatible, quasi-linear complexity. In: Tang, Q., Teague, V. (eds.) PKC 2024, Part II. LNCS, vol. 14604, pp. 354–385. Springer, Cham (Apr 2024). https://doi.org/10.1007/978-3-031-57728-4_12
35. Lee, Y.: Lmkcdey revisited: Speeding up blind rotation with signed evaluation keys. *Mathematics* **12**(18) (2024). <https://doi.org/10.3390/math12182909>, <https://www.mdpi.com/2227-7390/12/18/2909>
36. Lee, Y., Micciancio, D., Kim, A., Choi, R., Deryabin, M., Eom, J., Yoo, D.: Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part III. LNCS, vol. 14006, pp. 227–256. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30620-4_8
37. Li, N., Zhou, T., Yang, X., Han, Y., Liu, W., Tu, G.: Efficient multi-key fhe with short extended ciphertexts and directed decryption protocol. *IEEE Access* **7**, 56724–56732 (2019)
38. Li, Y., Wen, J., Liang, S., Kong, F., Huang, B.: Multi-group fully homomorphic encryption scheme based on lwe and ntru. *Electronics* **15**(5) (2026). <https://doi.org/10.3390/e15050511>

- [//doi.org/10.3390/electronics15050940](https://doi.org/10.3390/electronics15050940), <https://www.mdpi.com/2079-9292/15/5/940>
39. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Karloff, H.J., Pitassi, T. (eds.) 44th ACM STOC. pp. 1219–1234. ACM Press (May 2012). <https://doi.org/10.1145/2213977.2214086>
 40. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Berlin, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_1
 41. Min, S., Park, J., Song, Y.: Multi-key fully homomorphic encryption with non-interactive setup in the plain model. In: Heninger, N., Rosulek, M. (eds.) Advances in Cryptology – CRYPTO 2026. pp. 331–362. Springer Nature Switzerland, Cham (2026)
 42. Mouchet, C., Bertrand, E., Hubaux, J.P.: An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption. *J. Cryptol.* **36**(2) (Mar 2023). <https://doi.org/10.1007/s00145-023-09452-8>, <https://doi.org/10.1007/s00145-023-09452-8>
 43. Mouchet, C., Troncoso-Pastoriza, J.R., Bossuat, J.P., Hubaux, J.P.: Multiparty homomorphic encryption from ring-learning-with-errors. *PoPETs* **2021**(4), 291–311 (Oct 2021). <https://doi.org/10.2478/popets-2021-0071>
 44. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 735–763. Springer, Berlin, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_26
 45. Okada, H., Takagi, T.: Low communication threshold FHE from standard (module-)LWE. In: Hanaoka, G., Yang, B.Y. (eds.) ASIACRYPT 2025, Part VII. LNCS, vol. 16251, pp. 165–198. Springer, Singapore (Dec 2025). https://doi.org/10.1007/978-981-95-5122-4_6
 46. Park, J., Rovira, S.: Efficient tfhe bootstrapping in the multiparty setting. *IEEE Access* **11**, 118625–118638 (2023). <https://doi.org/10.1109/ACCESS.2023.3326237>
 47. Park, J., Van Leeuwen, B., Zajonc, O.: FINALLY: A multi-key FHE scheme based on NTRU and LWE. *CiC* **1**(3), 15 (2024). <https://doi.org/10.62056/aebn-4c2h>
 48. Passelègue, A., Stehlé, D.: Low communication threshold fully homomorphic encryption. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part I. LNCS, vol. 15484, pp. 297–329. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0875-1_10
 49. Peikert, C., Shiehian, S.: Multi-key FHE from LWE, revisited. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 217–238. Springer, Berlin, Heidelberg (Oct / Nov 2016). https://doi.org/10.1007/978-3-662-53644-5_9
 50. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005). <https://doi.org/10.1145/1060590.1060603>
 51. Rotaru, D., Smart, N.P., Tanguy, T., Vercauteren, F., Wood, T.: Actively secure setup for SPDZ. *Journal of Cryptology* **35**(1), 5 (Jan 2022). <https://doi.org/10.1007/s00145-021-09416-w>
 52. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient public key encryption based on ideal lattices. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 617–635. Springer, Berlin, Heidelberg (Dec 2009). https://doi.org/10.1007/978-3-642-10366-7_36

53. Xiang, B., Min, S., Hwang, I., Wang, Z., He, H., Wei, Y., Yang, K., Zhang, J., Deng, Y., Yu, Y.: HERDS: Multi-key fully homomorphic encryption with sublinear bootstrapping. In: Daemen, J., Thomé, E. (eds.) EUROCRYPT 2026, Part IV. LNCS, vol. 16545, pp. 214–242. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-25327-9_8
54. Xiang, B., Zhang, J., Wang, K., Deng, Y., Feng, D.: NTRU-based bootstrapping for MK-FHEs without using overstretched parameters. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part I. LNCS, vol. 15484, pp. 241–270. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0875-1_8
55. Zyskind, G., Zarchy, D., Leibovich, M., Peikert, C.: High-throughput universally composable threshold FHE decryption. In: Huang, C.Y., Chen, J.C., Shieh, S.P., Lie, D., Cortier, V. (eds.) ACM CCS 2025. pp. 2339–2353. ACM Press (Oct 2025). <https://doi.org/10.1145/3719027.3744884>

A Additional Preliminary

A.1 Fully Homomorphic Encryption (FHE) And Its Variants

Definition 4 (Fully Homomorphic Encryption). A fully homomorphic encryption (FHE) scheme is a family of PPT algorithms $\mathcal{E} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Eval}, \text{Dec})$, defined as follows.

- $pp \leftarrow \text{Setup}(1^\lambda)$: Given a security parameter λ , outputs a public parameter pp .
- $(sk, evk, pk) \leftarrow \text{KeyGen}(pp)$: Given a public parameter pp , outputs a secret key, (public) evaluation key, and public key pair (sk, evk, pk) .
- $ct \leftarrow \text{Enc}(pk; m)$: Given a public key pk and message m , outputs a ciphertext ct .
- $ct \leftarrow \text{Eval}(C, evk, pk, \{ct_i\}_{1 \leq i \leq \ell})$: Given a circuit C , evaluation key evk , public key pk and ℓ ciphertexts ct_i , outputs a ciphertext ct .
- $m \leftarrow \text{Dec}(sk; ct)$: Given a secret key sk and a ciphertext ct , outputs a message m .

We also require the following properties.

- **Correctness:** For $\text{Dec}(sk, \text{Eval}(C, \{ct_i\}_{1 \leq i \leq \ell}, evk)) = C(\{m_i\}_{1 \leq i \leq \ell})$ with an overwhelming probability.
- **Compactness:** For $ct \leftarrow \text{Eval}(C, \{ct_i\}_{1 \leq i \leq \ell}, evk)$, $|ct| = \text{poly}(\lambda)$.

For multi-key fully homomorphic encryption scheme, we follow the definition from [39].

Definition 5 (Multi-Key Fully Homomorphic Encryption). A multi-key fully homomorphic encryption (MKFHE) scheme is a family of PPT algorithms $\mathcal{E} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Eval}, \text{Dec})$, defined as follows.

- $pp \leftarrow \text{Setup}(1^\lambda, 1^K)$: Given a security parameter λ and the number of parties $K > 0$, outputs a public parameter pp .
- $(sk, evk, pk) \leftarrow \text{KeyGen}(pp)$: Given a public parameter pp , outputs a secret key, (public) evaluation key, and public key pair (sk, evk, pk) .
- $ct \leftarrow \text{Enc}(pk; m)$: Given a public key pk and message m , outputs a ciphertext ct .
- $ct \leftarrow \text{Eval}(C, \{(evk_i, pk_i, ct_i)\}_{1 \leq i \leq \ell})$: Given a circuit C , ℓ tuples (evk_i, pk_i, ct_i) , each consists of evaluation key evk_i , public key pk_i and ciphertext ct_i , outputs a ciphertext ct .
- $m \leftarrow \text{Dec}(\{sk_i\}_{1 \leq i \leq K}; ct)$: Given secret keys $\{sk_i\}_{1 \leq i \leq K}$ and a ciphertext ct , outputs a message m .

We also require the following properties.

- **Correctness:** For $\text{Dec}(sk, \text{Eval}(C, \{(ct_i, evk_i)\}_{1 \leq i \leq \ell})) = C(\{m_i\}_{1 \leq i \leq \ell})$ with overwhelming probability.
- **Compactness:** For $ct \leftarrow \text{Eval}(C, \{(ct_i, evk_i)\}_{1 \leq i \leq \ell}, evk)$, $|ct| = \text{poly}(\lambda, K)$.

A.2 Useful Lemmas

Lemma 6. *Let $M = 2^{\lambda-1}K(K-1)$. Then, the collision probability of K uniformly sampled integers from $[M]$ is less than $2^{-\lambda}$.*

Proof. Let X_i be the random variable sampled from $[M]$, and let $E_{i,j}$ be the event that $X_i = X_j$ for $1 \leq i < j \leq K$. Then, $\Pr[E_{i,j}] = 1/M$ for any i, j . Therefore, the collision probability is bounded by

$$\Pr \left[\bigcup_{i,j} E_{i,j} \right] \leq \sum_{i,j} \Pr[E_{i,j}] = \frac{K(K-1)}{2M} = 2^{-\lambda}.$$

□

Lemma 7. *If $\alpha \leq x \leq \beta$ for some $0 < \alpha, \beta < 1$, the following inequality holds.*

$$e^{x \log(1-\beta)/\beta} \leq 1 - x \leq e^{x \log(1-\alpha)/\alpha}$$

Proof. Consider a function $e^{-cx} + x - 1$. Assume that it intersects with the x-axis at the point $x = t$. Then, we obtain $c = -\log(1-t)/t$. Since it is convex and passes through the origin, $e^{-cx} + x - 1 \leq 0$ for all $0 \leq x \leq t$, and $e^{-cx} + x - 1 \geq 0$ for all $t \leq x$. By substituting α and β into t , we obtain the inequalities

$$e^{\log(1-\beta)x/\beta} + x - 1 \leq 0 \quad \text{and} \quad e^{\log(1-\alpha)x/\alpha} + x - 1 \geq 0$$

given that $\alpha \leq x \leq \beta$. By adding $1 - x$ to both sides of the inequalities, we can prove the claim. □

Lemma 8. *For any $1 < x$, the following inequality holds.*

$$\frac{1}{x} \leq -\log \left(1 - \frac{1}{x} \right) \leq \frac{1}{x-1}$$

Proof. For a fixed x , we have

$$1 \leq \frac{1}{1-t} \leq \frac{1}{1-1/x} = \frac{x}{x-1}$$

for all $1 \leq t \leq 1/x$. Therefore, by integrating each side from 0 to $1/x$, we obtain the desired inequality

$$\frac{1}{x} = \int_0^{1/x} 1 \cdot dt \leq -\log \left(1 - \frac{1}{x} \right) = \int_0^{1/x} \frac{1}{1-t} dt \leq \frac{1}{x-1} = \int_0^{1/x} \frac{1}{1-1/x} dt.$$

□

B Deferred Proofs

B.1 Proof of Lem. 1

Proof. For a fixed party i , the probability that its indices collide with other parties' indices for all repetitions is given by

$$\left(1 - \left(1 - \frac{1}{M}\right)^{K-1}\right)^A,$$

since the probability that the i -th party's index does not collide per repetition is $\left(\frac{M-1}{M}\right)^{K-1}$, given that the parties choose the indices uniformly at random. Therefore, by applying the union bound for all $1 \leq i \leq K$, we obtain the claimed failure probability. $\square$

B.2 Proof of Lem. 2

Proof. We first assume that $M \geq K$, which is a natural choice of M , as at most K parties choose random indices between 1 and M . Of course, the protocol could succeed even when M is smaller than K , as long as it is larger than 2, however, it would amplify the value of A by a huge factor. Then, the failure probability from Lem. 1 can be bounded as follows:

$$\begin{aligned} & K \cdot \left(1 - \left(1 - \frac{1}{M}\right)^{K-1}\right)^A \\ & \leq K \cdot \left(1 - \exp\left(\frac{1}{M} \cdot K \log\left(1 - \frac{1}{K}\right)(K-1)\right)\right)^A \\ & \leq K \cdot \exp\left(e \log\left(1 - \frac{1}{e}\right) \cdot \exp\left(\frac{1}{M} \cdot K \log\left(1 - \frac{1}{K}\right)(K-1)\right) \cdot A\right) \end{aligned}$$

where the first inequality is obtained by applying Lem. 7 with $\beta = 1/K$, since $M \geq K \geq 2$, and the second inequality is obtained by applying Lem. 7 again with $\alpha = e^{-1}$, since

$$\alpha = e^{-1} \leq e^{-K/M} \leq \exp\left(\frac{1}{M} \cdot K \log\left(1 - \frac{1}{K}\right)(K-1)\right)$$

by Lem. 8. Therefore, to bound the failure probability below $2^{-\kappa}$, we can simply set

$$K \cdot \exp\left(e \log\left(1 - \frac{1}{e}\right) \cdot \exp\left(\frac{1}{M} \cdot K \log\left(1 - \frac{1}{K}\right)(K-1)\right) \cdot A\right) = 2^{-\kappa}.$$

Consequently, we obtain

$$AM = C_1(K) \cdot M \cdot e^{C_2(K)/M}$$

where

$$C_1(K) = (\log K + \kappa \log 2) / \left(-e \log(1 - \frac{1}{e}) \right),$$

$$C_2(K) = -K(K-1) \log \left(1 - \frac{1}{K} \right).$$

Given that $C_1(K)$ and $C_2(K)$ are positive for all K , $C_1(K) \cdot M \cdot e^{C_2(K)/M}$ is a monotonically increasing function for M , we set $M = K$ to minimize AM . By substituting this value into the equation above, we obtain the following explicit formulae for A and M .

$$A = \left\lceil \frac{\log K + \kappa \log 2}{-e \log(1 - 1/e)} \cdot e^{-(K-1) \log(1-1/K)} \right\rceil, \quad M = K. \quad (4)$$

The asymptotics of A is indeed $O(\log K + \kappa)$ given that $C_2(K) = O(K)$ by Lem. 8, and the asymptotics of M is, trivially, $O(K)$. As a result, we have $AM = O(K(\log K + \kappa))$. $\square$

B.3 Proof of Lem. 4

Proof. To avoid the collision of indices with a probability $2^{-\kappa}$, the index pool size M needs to be set to at least $M = K(K-1)2^{\kappa-1}$ by Lem. 3. Now, recall that each party needs to output $B \log_B M$ key-switching keys, which can be re-written as

$$B \log_B M = B \log_B (K(K-1)2^{\kappa-1}) = \frac{B}{\log B} \cdot (\kappa - 1 + \log(K^2 - K)),$$

which is minimized when $B = 3$. Nonetheless, each party needs to output $O\left(\frac{B}{\log B} \cdot (\kappa + \log K)\right)$ key-switching keys asymptotically. With constantly small B (for example, $B = 3$), this is reduced to $O(\kappa + \log K)$. $\square$

B.4 Proof of Lem. 5

Proof. Let $\bar{r} = \sum_{1 \leq i \leq k} r_i$. Then, by definition,

$$\begin{aligned} \mathbf{b} &= \mathbf{a}_{\text{rlk}}^0 \cdot \bar{s} + \bar{\mathbf{e}}_b \pmod{Q} \\ \mathbf{d} &= -\mathbf{a}_{\text{rlk}}^0 \cdot \bar{r} + \vec{g} \cdot \bar{s} + \bar{\mathbf{e}}_d \pmod{Q} \\ \mathbf{f} &= -\mathbf{a}_{\text{rlk}}^1 \cdot \bar{s} + \vec{g} \cdot \bar{r} + \bar{\mathbf{e}}_f \pmod{Q} \end{aligned}$$

for some small noise vector $\bar{\mathbf{e}}_b, \bar{\mathbf{e}}_d, \bar{\mathbf{e}}_f \in R^d$ where $\|\bar{r}\|_\infty \leq B_\chi$ and $\|\bar{\mathbf{e}}_b\|_\infty, \|\bar{\mathbf{e}}_d\|_\infty, \|\bar{\mathbf{e}}_f\|_\infty \leq B_\psi$. By substituting this into main formula, we obtain

$$\begin{aligned} b' + a' \cdot \bar{s} &= \bar{s} \cdot (b + a\bar{s}) + \bar{\mathbf{e}}_d^\top \cdot (g^{-1}(b) + g^{-1}(a) \cdot \bar{s}) \\ &\quad + \bar{r} \cdot \bar{\mathbf{e}}_b^\top \cdot g^{-1}(a) + \mathbf{e}_f \cdot g^{-1} \left(\mathbf{a}_{\text{rlk}}^0{}^\top \cdot g^{-1}(b) + \mathbf{b}^\top \cdot g^{-1}(a) \right). \end{aligned}$$

By definition, we have

$$\begin{aligned}\|\bar{\mathbf{e}}_d^\top \cdot (g^{-1}(b) + g^{-1}(a) \cdot \bar{s})\|_\infty &\leq dKB_\psi N(1 + KNB_\chi), \\ \|\bar{r} \cdot \bar{\mathbf{e}}_b^\top \cdot g^{-1}(a)\|_\infty &\leq dK^2BB_\chi B_\psi N^2, \\ \|\mathbf{e}_f \cdot g^{-1}(\mathbf{a}_{\text{rlk}}^0{}^\top \cdot g^{-1}(b) + \mathbf{b}^\top \cdot g^{-1}(a))\|_\infty &\leq dKB_\psi N\end{aligned}$$

Therefore, we obtain the bound of the infinity norm of the final error e by summing them all, *i.e.*,

$$\|e\|_\infty \leq 2dKB_\psi N(1 + KNB_\chi).$$

□

C Noise Analysis of Key-switching Methods

Now we provide a noise analysis for our novel key-switching methods. Below, we use B as the bound of gadget decomposition, d as the length of the gadget vector, and B_χ, B_ψ as the probabilistic upper bounds of the key and noise distributions χ and ψ .

We begin by providing a lemma on the noise growth of the key-switching operation.

Lemma 9 (Key-Switching). *Given an RLWE ciphertext $\text{ct} \in R_Q^2$ and a key-switching key ksk from a secret s to a secret t with noise bound B_{ksk} , *i.e.*,*

$$[1 \ t] \cdot \text{ksk} = \bar{g}^\top \cdot s + \mathbf{e}^\top \pmod{Q}, \quad \text{and} \quad \|\mathbf{e}\|_\infty \leq B_{\text{ksk}},$$

it satisfies that

$$[1 \ t] \cdot \text{ct}' = [1 \ s] \cdot \text{ct} + e \pmod{Q}$$

where $\text{ct}' \leftarrow \text{KeySwitch}(\text{ksk}; \text{ct})$ for some $e \in R$ such that $\|e\|_\infty \leq dBB_{\text{ksk}}N$.

Proof. Let us parse $\text{ct} = (b, a)$ and $\text{ct}' = (b', a')$. Then, by definition, we have

$$[1 \ t] \cdot \text{ct}' = [1 \ t] \cdot \left(\begin{bmatrix} b \\ 0 \end{bmatrix} + \text{ksk} \cdot g^{-1}(a) \right) = b + as + \mathbf{e}^\top \cdot g^{-1}(a) \pmod{Q}.$$

Therefore, we can set $e = \mathbf{e}^\top \cdot g^{-1}(a)$ and its infinity norm is bounded by $dBB_{\text{ksk}}N$ since $\|\mathbf{e}\|_\infty \leq B_{\text{ksk}}$ and $\|g^{-1}(a)\|_\infty \leq B$ by definition. □

Now we analyze the noise growth from each key-switching method from Sec. 4.1 and Sec. 4.2.

Lemma 10. *Given the joint key-switching key $\overline{\text{ksk}}$ obtained with Algorithm 1 and a ciphertext $\text{ct} = (b, a) \in R_Q^2$ of the i -th party, let $\text{ct}' = (b', a') \leftarrow \text{JointKeySwitch}(\overline{\text{ksk}}; \text{ct})$. Then, it satisfies that*

$$[1 \ \bar{s}] \cdot \text{ct}' = [1 \ s_i] \cdot \text{ct} + e \pmod{Q}$$

for some small $e \in R$ where $\|e\|_\infty \leq dBB_\psi KN$

Proof. By definition, the key-switching key $\text{ksk}_{s_i \rightarrow \bar{s}}$ satisfies

$$[1 \ \bar{s}] \cdot \text{ksk}_{s_i \rightarrow \bar{s}} = [1 \ \bar{s}] \cdot \left[\sum_{1 \leq j \leq k} \begin{bmatrix} \mathbf{b}_{\alpha, \text{id}x_{\alpha}^i}^j \\ \mathbf{a}_{\alpha, \text{id}x_{\alpha}^i} \end{bmatrix} \right] = \sum_{1 \leq j \leq k} \mathbf{e}_{\alpha, \text{id}x_{\alpha}^i}^j \pmod{Q}$$

for some $1 \leq \alpha \leq A$ and $1 \leq \text{id}x_{\alpha}^i \leq M$. Since each $\mathbf{e}_{\alpha, \text{id}x_{\alpha}^i}^j$ is drawn from the noise distribution ψ , the noise term of the key-switching key is bounded by $kB_{\psi} \leq KB_{\psi}$. By substituting this bound into Lem. 9 directly, we can prove the claim. $\square$

Lemma 11. *Given the joint key-switching key $\overline{\text{ksk}}$ obtained from Algorithm 2 and a ciphertext $\text{ct} = (b, a) \in R_Q^2$ under the i -th party's secret key s_i , let $\text{ct}' = (b', a') \leftarrow \text{JointKeySwitch}(\overline{\text{ksk}}; \text{ct})$. Then, it satisfies that*

$$b' + a'\bar{s} = b + as + e \pmod{Q}$$

for some small $e \in R$ where $\|e\|_{\infty} \leq dBB_{\psi}K(K-1)N$

Proof. By construction, each $\text{ksk}_{s_{\mathcal{J}_{\text{in}}} \rightarrow s_{\mathcal{J}_{\text{out}}}}$ has the noise bound of Ke_{ψ} , similar to the proof of Lem. 10. Therefore, a single key-switching inside the algorithm (Algorithm 3) introduces an error whose infinity bound is $dBB_{\psi}KN$ by Lem. 9. Recall that the number of unit key-switchings is bounded by $K-1$, the noise increase from the entire key-switching key algorithm is bounded by $dBB_{\psi}K(K-1)N$. $\square$

In usual parameters, the gadget dimension $d = O(\log Q)$ and the bound $B = O(1)$. The noise distribution ψ is usually chosen as a constant-sized distribution, with $B_{\psi} = O(1)$. Therefore, the key-switching algorithm introduces noise of size only $O(K^2 \log QN)$ in either method.

D Additional Discussions On Key-switching

D.1 Comparison

We now investigate the differences between two proposed key-switching algorithms. The first method is computation-friendly with $O(1)$ key-switching cost, at the expense of quasi-linear large communication and setup costs with respect to the maximum number of parties K . On the other hand, the second method is communication-friendly with $O(\log K + \kappa)$ communication cost and $O(K(\log K + \kappa))$ setup cost, while the key-switching cost is $O(K)$. We summarize the above in Table 3.

It is worth noting that these two methods do not contradict each other and can, therefore, be combined for a more flexible choice of parameters. The computation-friendly method essentially repeats the whole process with smaller M in order to amplify the success probability to an overwhelming level. In a similar manner, the communication-friendly method can be repeated with polynomially large M . This, in fact, may increase the communication cost but help remove the random oracle model by keeping the entire CRS size polynomially large with respect to K and κ , and help reduce the key-switching cost.

Method	Comm. Cost	Setup Cost	Key-Switching Cost	Security Model
Algorithm 1	$O(K(\log K + \kappa))$	$O(K^2)$	$O(1)$	CRS
Algorithm 2	$O(\log K + \kappa)$	$O(K(\kappa + \log K))$	$O(K)$	CRS + ROM

Table 3. Comparison of different key-switching algorithms. K , κ , and B denote the (maximum) number of parties, a parameter for failure probability, and the base for the B -tree construction.

D.2 LWE-based construction

We also note that our key-switching key generation protocol can be extended to the LWE setting. In the LWE setting, each party's secret $\vec{s}_i$ is given as a (short) vector in $\mathbb{Z}^n$ for the LWE dimension n , and the joint secret is given as $\vec{s} = \sum_{1 \leq i \leq k} \vec{s}_i$. The key-switching key from $\vec{s}_i$ to $\vec{s}$ is a stack of LWE encryptions of each entry of $\vec{g} \otimes s_i$. The main idea remains the same, while the CRS for a single key-switching key is now a matrix of $\mathbb{Z}_Q$ elements of size $n \times nd$. Given AM (or M , when using the communication-friendly method), each i -th party picks a random index (or indices) and outputs an encryption of $\vec{g} \otimes s_i$ at that exact index and zero encryption otherwise using the CRS corresponding to the indices. Then, they can be aggregated on-the-fly to generate a joint key-switching key. The space complexity of the LWE-based protocol is now quadratic with respect to the ciphertext dimension $n(= N)$, unlike the RLWE-based protocol, which has linear space complexity with respect to N .

E Correctness Proof of Multi-Key GSW

Lemma 12 (Encryption). For $ct \leftarrow \text{MKGSW.Enc}(pk_i; m)$ for some $m \in R_Q$,

$$[1 \ s_i] \cdot \overline{ct} = m \cdot \vec{g}^\top + \mathbf{e}^\top$$

for some $\mathbf{e} \in R^d$ where $\|\mathbf{e}\|_\infty \leq 2B_\chi B_\psi N + B_\psi$.

Proof. See Sec. B.

Proof. By definition,

$$[1 \ s_i] \cdot ct = e_{pk}^i \cdot \mathbf{r}^\top + s_i \cdot \mathbf{e}_1^\top + \mathbf{e}_0^\top + \vec{g}^\top \cdot m \pmod{Q}.$$

The infinity norm of each term $e_{pk}^i \cdot \mathbf{r}^\top$, $s_i \cdot \mathbf{e}_1^\top$, and $\mathbf{e}_0^\top$ is bounded by $B_\chi B_\psi N$, $B_\chi B_\psi N$, and B_ψ . Therefore, we obtain the claimed bound by summing them all. $\square$

Lemma 13 (Extend). Given the joint relinearization and key-switching keys $\overline{rlk}$ and $\overline{ksk}$, and a fresh *RLWE* encryption ct of $m \in R_Q$ from the i -th party, let $\overline{ct} \leftarrow \text{MKGSW.Extend}(\overline{rlk}, \overline{ksk}; ct)$. Then,

$$[1 \ \vec{s}] \cdot \overline{ct} = m \cdot \vec{g}^\top \otimes [1 \ \vec{s}] + \mathbf{e} \pmod{Q}$$

for some $\mathbf{e} \in R^{2d}$, where $\|\mathbf{e}\|_\infty \leq 3dBB_\chi^2 B_\psi K^3 N^2$.

Proof. For simplicity, let us define $\mathbf{e} = (e_1, \dots, e_{2d})$ as the ‘noise’ of the RGSW ciphertext. Let us parse $\overline{\mathbf{ct}} = [\overline{\mathbf{ct}}_1 \mid \dots \mid \overline{\mathbf{ct}}_{2d}]$ and see each $\overline{\mathbf{ct}}_i$ as an RLWE ciphertext. Then, for any $1 \leq j \leq d$, the noise of $\overline{\mathbf{ct}}_j$, i.e., $[1 \ \overline{s}] \cdot \overline{\mathbf{ct}}_j - m \cdot g_j \bmod Q$ has infinite norm bounded by

$$2B_\chi B_\psi N + B_\psi + dBB_\psi K(K-1)N \leq dBB_\chi B_\psi K^2 N \quad (5)$$

through combining Lems. 10 to 12. Therefore, the noise of $\overline{\mathbf{ct}}_{j+d}$, i.e., $[1 \ \overline{s}] \cdot \overline{\mathbf{ct}}_{j+d} - m \cdot g_j \overline{s} \bmod Q$ has infinite norm bounded by

$$dBB_\chi^2 B_\psi K^3 N^2 + 2dKBB_\psi N(1 + KNB_\chi)$$

by combining Eq. (5) and Lem. 5. This can be naively bounded with $3dBB_\chi^2 B_\psi K^3 N^2$. $\square$

Lemma 14 (Multiplication). *Given (multi-key) RGSW encryptions $\overline{\mathbf{ct}}$ and $\overline{\mathbf{ct}}'$ of m and m' with noise vector $\mathbf{e} \in R_Q^{2d}$ and $\mathbf{e}' \in R_Q^{2d}$, i.e.,*

$$[1 \ \overline{s}] \cdot \overline{\mathbf{ct}} = m \cdot \vec{g} \otimes [1 \ \overline{s}] + \mathbf{e}^\top, \quad [1 \ \overline{s}] \cdot \overline{\mathbf{ct}}' = m' \cdot \vec{g} \otimes [1 \ \overline{s}] + \mathbf{e}'^\top \pmod{Q},$$

let $\overline{\mathbf{ct}}'' = \text{MKGSW.Mult}(\overline{rlk}; \overline{\mathbf{ct}}, \overline{\mathbf{ct}}')$. Then it follows that

$$[1 \ \overline{s}] \cdot \overline{\mathbf{ct}}'' = mm' \cdot \vec{g} \otimes [1 \ \overline{s}] + \mathbf{e}''^\top \pmod{Q}$$

for some $\mathbf{e}'' \in R^{2d}$ where $\|\mathbf{e}''\|_\infty \leq \|m\|_1 \cdot \|\mathbf{e}'\|_\infty + 2dBN \cdot \|\mathbf{e}\|_\infty$.

Proof. By definition, we have

$$\begin{aligned} [1 \ \overline{s}] \cdot \overline{\mathbf{ct}}'' &= [1 \ \overline{s}] \cdot \overline{\mathbf{ct}} \cdot g^{-1}(\mathbf{ct}') \\ &= (m \cdot \vec{g} \otimes [1 \ \overline{s}] + \mathbf{e}^\top) \cdot g^{-1}(\mathbf{ct}') \\ &= m \cdot [1 \ \overline{s}] \cdot \mathbf{ct}' + \mathbf{e}^\top \cdot g^{-1}(\mathbf{ct}') \\ &= mm' \cdot \vec{g} \otimes [1 \ \overline{s}] + m \cdot \mathbf{e}'^\top + \mathbf{e}^\top \cdot g^{-1}(\mathbf{ct}') \pmod{Q}. \end{aligned}$$

Since $\|m \cdot \mathbf{e}'^\top\|_\infty \leq \|m\|_1 \cdot \|\mathbf{e}'\|_\infty$ and $\|\mathbf{e}^\top \cdot g^{-1}(\mathbf{ct}')\|_\infty \leq 2dBN\|\mathbf{e}\|_\infty$, the infinity norm of the noise $\mathbf{e}'' = m \cdot \mathbf{e}'^\top + \mathbf{e}^\top \cdot g^{-1}(\mathbf{ct}')$ is bounded by $\|m\|_1 \cdot \|\mathbf{e}'\|_\infty + 2dBN\|\mathbf{e}\|_\infty$. $\square$

F Correctness Proof of Multi-Key FV

Lemma 15 (Multiplication). *Given two multi-key FV encryptions $\overline{\mathbf{ct}}$ and $\overline{\mathbf{ct}}'$ of m and m' , i.e.,*

$$b + a\overline{s} = \frac{Q}{p} \cdot m + e, \quad b' + a'\overline{s} = \frac{Q}{p} \cdot m' + e' \pmod{Q}$$

for some noise e and e' , let $\overline{ct''} \leftarrow \text{MKFV.Mul}(\overline{rlk}; \overline{ct}, \overline{ct}')$. Then it follows that

$$[1 \ \overline{s}] \cdot \overline{ct''} = \frac{Q}{p} \cdot mm' + e'' \pmod{Q}$$

for some $e'' \in R$ where $\|e''\|_\infty = O(pKN^2(\|e\|_\infty + \|e'\|_\infty))$.

Proof. Let $\overline{ct} = (b, a)$ and $\overline{ct}' = (b', a')$ be the FV encryptions of the messages m and m' . For the ‘liftings’ $(\hat{b}, \hat{a})$ and $(\hat{b}', \hat{a}')$ of the ciphertexts, assuming that the secret key is small enough, we have

$$\hat{b} + \hat{a}\overline{s} = \frac{Q}{p} \cdot m + QI + e, \quad \hat{b}' + \hat{a}'\overline{s} = \frac{Q}{p} \cdot m' + QI' + e'$$

for small polynomials I and I' . Specifically, they are bounded by $0.5\|\overline{s}\|_1 \leq 0.5KB_\chi N$. Therefore, we obtain the following equation by tensoring these ciphertexts modulo Q^2/p :

$$\begin{aligned} & \hat{b}\hat{b}' + (\hat{a}\hat{b}' + \hat{a}'\hat{b})\overline{s} + \hat{a}\hat{a}'(\overline{s})^2 \\ &= \frac{Q^2}{p^2} \cdot mm' + Q \cdot (Ie' + e'I) + \frac{Q}{p} \cdot (me' + m'e) + ee' \\ &= \frac{Q^2}{p^2} \cdot mm' + e_{\text{mul}} \pmod{\frac{Q^2}{p}}. \end{aligned}$$

By definition, we have

$$e_{\text{mul}} \leq \frac{Q}{2}KB_\chi N^2(\|e\|_\infty + \|e'\|_\infty) + \frac{Q}{2}N(\|e\|_\infty + \|e'\|_\infty) + N\|e\|_\infty\|e'\|_\infty.$$

Therefore, after we ‘scale’ the tensored ciphertext $(\hat{b}\hat{b}', \hat{a}\hat{b}' + \hat{a}'\hat{b}, \hat{a}\hat{a}')$ into the modulus Q by multiplying p/Q and rounding, its phase is approximately $Q/p \cdot mm'$ under the secret $(1, \overline{s}, (\overline{s})^2)$. This rounding noise is bounded by

$$0.5(1 + KB_\chi N + K^2B_\chi^2N^2).$$

Now the secret is then changed back to $(1, \overline{s})$ using RLEV encryption of $(\overline{s})^2$ under $\overline{s}$ using the joint key multiplication algorithm. This incurs noise of the size only $2dKBB_\psi N(1 + KNB_\chi)$ by Lem. 5. As a result, we obtain the following bound of the multiplication noise:

$$\begin{aligned} e'' &\leq \frac{p}{2}(1 + KB_\chi N)N(\|e\|_\infty + \|e'\|_\infty) + \frac{p}{Q}N\|e\|_\infty\|e'\|_\infty \\ &\quad + \frac{1}{2}(1 + KB_\chi N + K^2B_\chi^2N^2) + 2dKBB_\psi N(1 + KNB_\chi). \end{aligned}$$

□

Lemma 16 (Automorphism). *Given an FV ciphertext $\overline{ct}$, let $\overline{ct}' \leftarrow \text{MKFV.Aut}(\overline{\text{atk}}_i; \overline{ct})$. Then it follows that*

$$[1 \ \overline{s}] \cdot \overline{ct}' = \Psi([1 \ \overline{s}] \cdot \overline{ct}) + e \pmod{Q}$$

for some $e \in R$, where $\Psi = X \rightarrow X^i$ and $\|e\|_\infty \leq dKBB_\psi N$.

Proof. We can understand the automorphism operation as a key-switching of the ‘rotated’ ciphertext $(b(X^i), a(X^i))$ from the secret $\bar{s}(X^i)$ to $\bar{s}$ where $\text{ct} = (b, a)$. Then, by Lem. 9, we can directly verify the claim. $\square$

Theorem 3 (Correctness). *A circuit $\mathcal{C}$ with depth $d_{\mathcal{C}}$ can be evaluated using the proposed multi-key FV scheme if $Q = \tilde{O}(p^{d_{\mathcal{C}}+1} K^{d_{\mathcal{C}}+2} N^{2d_{\mathcal{C}}+1})$.*

Proof. By the discussion in Sec. C, a multi-key FV ciphertext has the noise asymptotics $O(K^2 \log Q N)$ right after key-switching, assuming that the key and noise distributions χ and ψ are constantly small. Note that the addition increases the noise only additively, and the multiplication amplifies the noise by a factor of pKN^2 . The final noise after the evaluation of $\mathcal{C}$ is asymptotically $O(p^{d_{\mathcal{C}}} K^{d_{\mathcal{C}}+2} N^{2d_{\mathcal{C}}+1} \log Q)$. To allow for correct decryption, $Q = O(p^{d_{\mathcal{C}}+1} K^{d_{\mathcal{C}}+2} N^{2d_{\mathcal{C}}+1} \log Q)$. $\square$

Note that the proposed scheme incurs a KN extra factor for the multiplication compared to a single-key scheme. Since the decryption circuit has the complexity $O(\log N + \log \log Q)$, our construction contributes an extra $O((KN)^{\log N + \log \log Q})$ factor to the final modulus. This results in an increase in the ciphertext dimension by a factor of $\log K \cdot \log N$, making the ciphertext dimension $N = O(\lambda + \log \lambda \cdot \log K)$ to allow for bootstrapping.

G MK Variant of AP-like Schemes

AP-like FHE schemes [4, 23, 20, 36, 29] target low-latency circuit evaluation for LWE encryption of small integers. This family of FHE schemes supports *functional bootstrapping*, which can evaluate any univariate function while bootstrapping. This functionality can be obtained with a homomorphic lookup table (LUT) evaluation called the *blind rotation*. First, the input LWE ciphertext $(b, \vec{a}) \in \mathbb{Z}_Q^{n+1}$ is compressed with a compact ciphertext modulus $q \ll Q$, as long as it successfully decrypts. Let us write this compressed ciphertext $(b, \hat{a}') \in \mathbb{Z}_q^{n+1}$ where

$$b' + \langle \vec{a}', \vec{t} \rangle = \left\lfloor \frac{q}{p} \right\rfloor \cdot m + e \pmod{q}$$

for some small $|e| < q/2p$, where $\vec{t} = (t_0, \dots, t_{n-1})$ is the secret of the LWE ciphertext. Then, based on the evaluated function $f : \mathbb{Z}_p \rightarrow \mathbb{Z}_p$, an LUT $\mathcal{F} : \mathbb{Z}_q \rightarrow \mathbb{Z}_t$ is constructed so that $\mathcal{F}(Q/p \cdot m + e) = f(m)$ for all combinations of m and e . Note that the homomorphic evaluation of $\mathcal{F}([b' + \langle \vec{a}', \vec{t} \rangle]_q)$ outputs $f(m)$ by definition while removing the input noise e . At the core of blind rotation lies the evaluation of this equation, by physically (left) circular-shifting the LUT vector $(\mathcal{F}(0), \dots, \mathcal{F}(q-1))$ by the index $b' + \langle \vec{a}', \vec{t} \rangle$ rather than directly computing the modulo function $[\cdot]_q$. The first index then becomes $\mathcal{F}([b' + \langle \vec{a}', \vec{t} \rangle]_q)$, and we can simply extract the value to complete the bootstrapping.

This algorithm can be instantiated in various ways. Let us parse $a' = (a'_0, \dots, a'_{n-1}) \in \mathbb{Z}_q^n$. In the seminal work by Alperin-Sheriff and Peikert [4], the LWE ciphertexts of $\mathcal{F}(0)$ to $\mathcal{F}(q-1)$ were physically rotated by multiplying an encrypted

permutation matrix that rotates the ciphertexts by the index $a'_i t_i$ iteratively for $0 \leq i < n$. They also mentioned that it can be adopted into the ring setting using the monomial group. For simplicity, we consider $q = 2N$. Over the power-of-two cyclotomic ring $R = \mathbb{Z}[X]/(X^N + 1)$, consider a polynomial $tv = \sum_{0 \leq i < N} \mathcal{F}(i)X^i \in R_p$. Then, the shifting by $a'_i t_i$ can be done simply by multiplying a monomial $X^{-a'_i t_i}$ to the polynomial. The blind rotation can be performed by multiplying X^{-b} and $X^{-a'_i t_i}$ iteratively, and then the constant term becomes the desired value. One can then extract the constant term into an LWE encryption under the vectorized RLWE secret and switch the key to the original LWE secret. Note that this approach can encode only half of the LUT. interchangeably, it only accepts negacyclic LUTs where the last half is of the first half of the LUT.

This idea was later adopted and optimized in the FHEW homomorphic encryption scheme [23], where binary gates can be efficiently computed with a linear sum of input ciphertexts and a blind rotation. In this work, we focus only on the NAND gate for simplicity. In FHEW, a bit is embedded into $\mathbb{Z}_4$ and then encoded in the MSB, as in FV [11, 24]. In other words, an encryption $\mathbf{c} \in \mathbb{Z}_Q^{n+1}$ of a bit b satisfies $[1 \ \mathbf{t}^\top] \cdot \mathbf{c} \approx Q/4 \cdot b \bmod Q$. Let us assume $8 \mid Q$. To evaluate a NAND gate given input ciphertexts $\mathbf{c}, \mathbf{c}' \in \mathbb{Z}_Q^{n+1}$ of bits b and b' , we simply compute 1. $\mathbf{c}'' = \mathbf{c} + \mathbf{c}' + (Q/8, 0) \bmod Q$, 2. run blind rotation on a ‘scaled’ ciphertext $\hat{\mathbf{c}} = \lfloor 2N/Q \cdot \mathbf{c}'' \rfloor$ with the ‘test vector’ $tv = Q/8 \cdot \sum_{0 \leq i < N} X^i$ to obtain $\mathbf{c}'''$, and 3. output $\mathbf{c}''' + (Q/8, \vec{0}) \bmod Q$. This algorithm indeed outputs the NAND gate evaluation of the inputs since the encoding of message $b + b'$ is approximately $N/4$ if $b = b' = 0$, $5N/4$ if $b = b' = 1$, and $3N/4$ otherwise. Then, with the right parameterization, the blind rotation outputs $-Q/8$ if and only if $b = b' = 1$, and $Q/8$ otherwise. Hence, after the addition by $Q/8$, the final encoding is 0 if and only if $b = b' = 1$, and $Q/8$ otherwise, which indeed emulates the NAND gate evaluation.

There has been a series of works that optimize this blueprint [20, 36] or extend its functionality. As different directions, the Carousel scheme utilizes the automorphism group instead [29] and the SHIP scheme utilizes the root of unity over the complex plane [18]. Nevertheless, these schemes can be classified into two groups, namely FHEW-like schemes [23, 36] and CGGI-like schemes [20, 18, 29]. The main difference between these two schemes lies in how to instantiate the monomial multiplication. The FHEW-like schemes directly multiply the monomials $X^{-a'_i t_i}$ whereas the CGGI-like schemes leverage a homomorphic MUX gate to ‘select’ between the candidates of rotated lookup tables.

In this section, we construct two AP-style MKFHE schemes with succinct ciphertexts, where one is FHEW-like and the other is CGGI-like. There are a number of existing approaches that can potentially be combined with our novel key-switching method to build an MKFHE scheme. However, we build our schemes on top of two of these approaches, which achieve poly-logarithmic time complexity with respect to the number of participants. In our constructions, we set the LWE dimension $n = N$ for simplicity, thereby removing the need for key-switching. Naturally, the LWE secret is $\vec{t} = (\bar{s}_0, \bar{s}_1, \dots, \bar{s}_{N-1})$, where

the joint key is $\bar{s} = \sum_{0 \leq i < N} \bar{s}_i$. Finally, we use RLWE encryption as the base encryption, unlike the conventional AP-like schemes that use LWE encryption, in order to avoid the description of the LWE version of the key-switching algorithm. Nevertheless, this does not incur any functional difference in the final scheme.

G.1 FHEW

For FHEW-like bootstrapping, we leverage Lee's variant of the LMKCDEY bootstrapping [35], which was specifically designed for the multiparty setting. We first classify the indices of input ciphertexts as $\mathcal{I}_i = \{j \mid a'_j = i \bmod 2N\}$. Note that the multiplicative group $\mathbb{Z}_{2N}^\times$ is generated by 5 and -1 and is a set of odd numbers between 0 and $2N - 1$. Therefore, all the numbers inside $\mathbb{Z}_{2N}$ fall into these four categories: 5^i , $5^i + 1$, -5^i , and $-5^i + 1$. The core idea of LMKCDEY bootstrapping is to express the phase $b' + \langle \vec{a}', \vec{t} \rangle$ of the input ciphertext as follows:

$$\begin{aligned} b' + \sum_{0 \leq i < N} a'_i t_i &= b' + \sum_{0 \leq i < 2N} \sum_{j \in \mathcal{I}_i} i \cdot t_j \\ &= b' + \sum_{1 \leq i \leq N/2} \left(\sum_{j \in \mathcal{I}_{5^i}} 5^i t_j + \sum_{j \in \mathcal{I}_{5^i+1}} (5^i + 1) t_j + \sum_{j \in \mathcal{I}_{-5^i}} -5^i t_j + \sum_{j \in \mathcal{I}_{-5^i+1}} (-5^i + 1) t_j \right) \\ &= b' + \sum_{i \text{ is even}} t_i + \sum_{1 \leq i \leq N/2} \left(\sum_{\substack{j \in \mathcal{I}_{5^i} \\ \text{or} \\ j \in \mathcal{I}_{5^i+1}}} t_j - \sum_{\substack{j \in \mathcal{I}_{-5^i} \\ \text{or} \\ j \in \mathcal{I}_{-5^i+1}}} t_j \right) \cdot 5^i \pmod{2N}. \end{aligned}$$

We understand this as the evaluation of a degree- $N/2$ polynomial at 5, where the coefficients are the addition and subtraction of secret key components t_j . We evaluate this polynomial using Horner's method over the exponent of a monomial. Over the exponent, multiplication by 5 naturally translates to the automorphism $X \mapsto X^5$, and addition translates to multiplication by monomials. Therefore, its homomorphic evaluation is feasible with RGSW encryptions of $X^{\pm t_i}$ and the automorphism key for $X \mapsto X^5$. We note that $X^{\pm t_j} = \pm \sum_{1 \leq i \leq k} s_{i,j}$ can be computed from $X^{\pm s_{i,j}}$, j -th coefficient of the i -th party's secret s_i , *i.e.*, $s_i = \sum_{0 \leq j < N} s_{i,j} X^{j-1}$.

Specifically, the function f_{evk} and the circuit C_{evk} are given as follows:

$$\begin{aligned} - f_{\text{evk}}(s_i) &= (X^{\pm s_{i,0}}, \dots, X^{\pm s_{i,N-1}}) \\ - C_{\text{evk}}(f_{\text{evk}}(s_1), \dots, f_{\text{evk}}(s_k)) &= (\prod_{1 \leq i \leq k} X^{\pm s_{i,0}}, \dots, \prod_{1 \leq i \leq k} X^{\pm s_{i,N-1}}), \end{aligned}$$

and the circuit C_{evk} has a multiplicative depth k . After homomorphic evaluation of C_{evk} using the multi-key RGSW FHE scheme, we obtain a set of blind rotation keys $\overline{\text{brk}} = \{(\overline{\text{brk}}_j^+, \overline{\text{brk}}_j^-)\}_{0 \leq j < N}$ where each $\overline{\text{brk}}_j^+$ is an RGSW encryption of X^{t_j} and $\overline{\text{brk}}_j^-$ is an RGSW encryption of X^{-t_j} .

In principle, the automorphism key for $X \mapsto X^5$ needs to be generated via multi-key RGSW. However, we remark that it can be generated more efficiently using the same methodology as the multi-key FV scheme Sec. 5.3.

The concrete blind rotation algorithm based on these evaluation keys is given in Algorithm 4.

Algorithm 4 NAND evaluation using [35]

Input: FHEW ciphertexts $(b = \sum_{0 \leq i < N} b_i X^i, a = a_0 - \sum_{1 \leq i < N} a_i X^{N-i}), (b' = \sum_{0 \leq i < N} b'_i X^i, a' = a'_0 - \sum_{1 \leq i < N} a'_i X^{N-i}) \in R_Q^2$, the blind rotation key $\overline{\text{brk}} = \{(\overline{\text{brk}}_j^+, \overline{\text{brk}}_j^-)\}_{0 \leq j < N}$, and the automorphism key $\overline{\text{atk}} = \overline{\text{atk}}_5$ for $X \mapsto X^5$.

- 1: $\hat{b} = \left\lfloor \frac{2N}{Q} \cdot (b_0 + b'_0 + Q/8 \bmod Q) \right\rfloor \bmod 2N$
- 2: $\mathcal{I}_i \leftarrow \{j \mid \lfloor 2N/Q \cdot (a_j + a'_j \bmod Q) \rfloor = i \bmod 2N\}$ for $0 \leq i < N$
- 3: $\text{acc} \leftarrow (Q/8 \cdot X^{-\hat{b}} \cdot \sum_{0 \leq i < N} X^i, 0) \in R_Q^2$
- 4: **for** $i = N/2; i \geq 1; i = i - 1$ **do**
- 5: **for** $j \in \mathcal{I}_{5^i} \cup \mathcal{I}_{5^i+1}$ **do**
- 6: $\text{acc} \leftarrow \overline{\text{brk}}_j^- \cdot g^{-1}(\text{acc}) \bmod Q$
- 7: **end for**
- 8: **for** $j \in \mathcal{I}_{-5^i} \cup \mathcal{I}_{-5^i+1}$ **do**
- 9: $\text{acc} \leftarrow \overline{\text{brk}}_j^+ \cdot g^{-1}(\text{acc}) \bmod Q$
- 10: **end for**
- 11: $\text{acc} \leftarrow \text{Aut}(\text{atk}_5; \text{acc})$
- 12: **end for**
- 13: **for** $1 \leq i \leq N/2$ **do**
- 14: **for** $j \in \mathcal{I}_{5^i+1} \cup \mathcal{I}_{-5^i+1}$ **do**
- 15: $\text{acc} \leftarrow \overline{\text{brk}}_j^- \cdot g^{-1}(\text{acc}) \bmod Q$
- 16: **end for**
- 17: **end for**

Output: $\mathbf{c} = \text{acc} + (Q/8, 0) \bmod Q$

G.2 CGGI

The conventional CGGI style blind rotation relies on the binary structure of the secret key. However, even if each party's secret is sampled from a binary distribution, the joint secret may not be a binary secret. Therefore, the existing multi-party variants [2, 46] modified the blind rotation formula. In this work, we rely on the construction from [46], which leverages the generalized CGGI bootstrapping due to Joye and Paillier [30]. At the heart of their bootstrapping lies a special bootstrapping key structure. Assuming that each party's secret is binary, each entry of the coefficient vector of the joint key is confined to an interval $[0, k]$. The blind rotation key is then given by Nk RGSW encryptions $\overline{\mathbf{C}}_{i,j}$ which indicate whether the i -th coefficient of the vectorized joint key $\bar{t}_i$ is j or not, for $0 \leq i < N$ and $1 \leq j \leq k$. In other words, $\overline{\mathbf{C}}_{i,j}$ is an encryption of one if and only if $\bar{t}_i = j$, and is a zero encryption otherwise. Then, homomorphically multiplying $X^{-a_i \bar{t}_i}$ to an RLWE encryption ct can be performed by computing

$$\text{ct} + \sum_{1 \leq j \leq k} \overline{\mathbf{C}}_{i,j} \cdot g^{-1}(X^{-a_i j} \cdot \text{ct} - \text{ct}),$$

interchangeably,

$$\text{ct} + \left(\sum_{1 \leq j \leq k} (X^{-a_{ij}} - 1) \cdot \overline{\mathbf{C}}_{i,j} \right) \cdot g^{-1}(\text{ct}).$$

It is easy to check that this formula indeed outputs the RLWE encryption whose phase is approximately $X^{-a_i \bar{t}_i} \cdot [1 \ \bar{s}] \text{ct}$ by the definition of the blind rotation keys $\overline{\mathbf{C}}_{i,j}$. Essentially, the blind rotation can be performed by 1. initializing the accumulator and 2. sequentially multiplying the term $X^{-a_i \bar{t}_i}$ to the accumulator using the above formula.

The main challenge in realizing this is generating the blind rotation keys from each party's inputs. Park and Rovira [46] resolve this problem by introducing the 'homomorphic indicator'. The main idea is to generate the blind rotation key when $k = \ell + 1$, from the blind rotation key when $k = \ell$ and the input from the $\ell + 1$ -th party for $1 \leq \ell < k$ inductively. For simplicity, let us consider the algorithm inside the plaintext domain rather than the ciphertext domain. Let $b_{i,j}^\ell$ for $1 \leq j \leq \ell$ be the plaintext of the blind rotation keys when $k = \ell$, and $b_{i,j}^{\ell+1}$ for $1 \leq j \leq \ell + 1$ be the blind rotation keys when $k = \ell + 1$, for some $0 \leq i < N$. Our algorithm is to (homomorphically) compute $b_{i,j}^{\ell+1}$ given the i -th entry of the ℓ -th party's secret $s_{\ell,i}$. If $s_{\ell,i}$ is zero, then it is obvious that $b_{i,j}^{\ell+1} = b_{i,j}^\ell$ for $1 \leq j \leq \ell$, and $b_{i,\ell+1}^{\ell+1} = 0$. And if $s_{\ell,i}$ is one, we have $b_{i,j+1}^{\ell+1} = b_{i,j}^\ell$ for $1 \leq j \leq \ell$, and $b_{i,1}^{\ell+1} = 1 - \sum_{1 \leq j \leq \ell} b_{i,j}^\ell$ since it is one if and only if all $b_{i,j}^\ell$ are zero, and zero otherwise. Using the base MUX gate $x + b \cdot (y - x)$ which outputs x if and only if the selecting bit $b = 0$ and outputs y if and only if $b = 1$, each $b_{i,j}^{\ell+1}$ can be expressed as the following linear relation with respect to the selecting bit $s_{\ell,i}$:

$$b_{i,j}^{\ell+1} = \begin{cases} b_{i,1}^\ell + s_{\ell,i} \cdot (1 - \sum_{1 \leq \iota \leq \ell} b_{i,\iota}^\ell - b_{i,1}^\ell) & \text{if } j = 1 \\ b_{i,j}^\ell + s_{\ell,i} \cdot (b_{i,j-1}^\ell - b_{i,j}^\ell) & \text{if } 2 \leq j \leq \ell \\ s_{\ell,i} \cdot b_{i,\ell}^\ell & \text{if } j = \ell + 1 \end{cases}.$$

By evaluating the circuit above using the multi-key GSW scheme, we can obtain the blind rotation key set for the CGGI blind rotation. When $\ell = 1$, we simply initialize $b_{i,1}^1 = s_{1,i}$. Then, the above equation is homomorphically evaluated for each $\ell = 2$ to $\ell = k$ inductively based on each party's input $s_{\ell,i}$.

Remark 1. It is worth noting that the multi-key variant of the Carousel scheme is almost directly derived from the CGGI construction. The Carousel scheme has a similar key structure to the CGGI scheme, where the only difference is the automorphism key sets. Since the automorphism key can also be non-interactively generated using the same idea as the multi-key FV scheme Sec. 5.3, the key generation for the Carousel scheme can be instantiated too. Furthermore, the Carousel bootstrapping circuit includes only the extra automorphism operation compared to the CGGI scheme.

Algorithm 5 NAND evaluation using [46]

Input: CGGI ciphertexts $(b = \sum_{0 \leq i < N} b_i X^i, a = a_0 - \sum_{0 \leq i < N} a_i X^{N-i}), (b' = \sum_{0 \leq i < N} b'_i X^i, a' = a_0 - \sum_{1 \leq i < N} a'_i X^{N-i}) \in R_Q^2$, the blind rotation key $\overline{\text{brk}} = \{\overline{\text{brk}}_{i,j}\}_{0 \leq i < N, 1 \leq j \leq k}$.

- 1: $\hat{b} = \lfloor 2N/Q \cdot (b_0 + b'_0 + Q/8 \bmod Q) \rfloor \bmod 2N$
- 2: $\text{acc} \leftarrow (Q/8 \cdot X^{-\hat{b}} \cdot \sum_{0 \leq i < N} X^i, 0) \in R_Q^2$
- 3: **for** $0 \leq i < N$ **do**
- 4: $\hat{a}_i \leftarrow \lfloor 2N/Q \cdot (a_i + a'_i \bmod Q) \rfloor \bmod 2N$
- 5: $\text{acc} \leftarrow \text{acc} + \left(\sum_{1 \leq j \leq k} \left(X^{-a_i j - 1} \cdot \overline{\text{brk}}_{i,j} \right) \right) \cdot g^{-1}(\text{acc})$
- 6: **end for**

Output: $\mathbf{c} = \text{acc} + (Q/8, 0) \bmod Q$
