Multi-Key Homomophic Encryption from TFHE

Hao Chen[1], Ilaria Chillotti[2], and Yongsoo Song[1]

[1] Microsoft Research, Redmond, USA
{haoche,Yongsoo.Song}@microsoft.com
[2] KU Leuven, Leuven, Belgium
ilaria.chillotti@kuleuven.be

Abstract. In this paper, we propose a Multi-Key Homomorphic Encryption (MKHE) which allows homomorphic evaluation of a binary gate (with bootstrapping) on ciphertexts encrypted under different keys. We generalize a low-latency homomorphic encryption scheme of Chillotti et al. (ASIACRYPT 2016) by exploiting a key-extension approach of Brakerski and Perlman (CRYPTO 2016).
All the prior works on MKHE were too inefficient to be used in practice. Our construction improved the performance in terms of both asymptotic and concrete complexity: the length of ciphertexts and the computational costs of a binary gate grow linearly and quadratically on the number of parties, respectively. Furthermore, our scheme is fully-dynamic so that no information about the involved parties needs to be known before the computation and the resulting ciphertext can be reused in further computation with newly joined parties.
To the best of our knowledge, this is the first work to implement an MKHE scheme. Our implementation takes about 0.15 (resp. 0.72) seconds to perform the gate bootstrapping when the number of involved parties is 2 (resp. 4).

Keywords: Multi-key homomorphic encryption · Bootstrapping.

1 Introduction

Homomorphic Encryption (HE) and Multi-Party Computation (MPC) are well-studied cryptographic primitives for secure computation. We can use an HE scheme to outsource the storage and computation to a public cloud, but all data providers should agree on the same public key generated by a secret key owner. MPC performs an interactive protocol between parties to evaluate a circuit without revealing an auxilarity information beyond the computation result, but it usually suffers from a high communication and round complexity.

López-Alt et al. [25] proposed the notion of Multi-Key Homomorphic Encryption (MKHE) which is a variant of HE supporting the computation on ciphertexts encrypted under different keys. This attractive primitive can address the issues of HE and MPC, and it has many applications such as round-efficient MPC (e.g. [15, 1, 21, 30, 27]) and spooky encryption [16]. There have been several researches (e.g. [14, 27, 5, 28, 8]) on MKHE. However, all the previous works were abstract and far from practical. In particular, the efficiency of MKHE remained an open question for years because there has been no study to implement or compare the MKHE schemes empirically.

In this paper, we present an efficient MKHE scheme based on the learning with errors (LWE) assumption [29] and its ring variant (RLWE) [26]. Our construction can be viewed as a generalization of the TFHE scheme of Chillotti et al. [10], where an LWE-based ciphertext encrypts a single bit and one can evaluate a binary gate on ciphertexts with bootstrapping using the ring GSW (RGSW) scheme [20, 17]. We modify the extension algorithm of Brakerski-Perlman [5] to generate a common bootstrapping key of involved parties from the individual evaluation keys.

This work is no worse than previous ones in terms of both functionality and efficiency. This work is comparale to previous ones in terms of both functionality and efficiency. Our construction is fully-dynamic, meaning that a new party can join the computation at anytime and a resulting ciphertext may be reused for further computation. Meanwhile, the length of ciphertext and the computational costs of a single binary gate grow linearly and quadratically on the number of involved parties, respectively (see Table 1 for a comparison). Furthermore, our scheme is simple to implement and easily compatible with existing techniques for advanced functionalities such as the threshold decryption [23, 2], circuit bootstrapping [11], and a larger plaintext space [3, 6].

Finally, we provide a proof-of-concept implementation with concrete parameter sets. For example, it took 3.4, 10.1, and 56.9 seconds to generate a bootstrapping key, and 0.15, 0.72, and 5.15 seconds for a gate bootstrapping when the number of parties is 2, 4, and 8, respectively, on a personal computer.

Overview of Our Scheme. Chillotti et al. [10] introduced a formalization of (R)LWE over the real torus $\mathbb{T} = \mathbb{R} \pmod 1$ and the set of cyclotomic polynomials $T = \mathbb{T}[X]/(X^N + 1)$, and proposed a FHE scheme (called TFHE) with a low-latency bootstrapping based on the LWE and RGSW schemes. We first describe the TFHE scheme briefly. TFHE generates both LWE secret $\mathbf{s} = (s_i)_{i \in [n]} \in \mathbb{Z}^n$ and RLWE secret $z(X) = \sum_{0 \le i < N} z_i X^i \in R = \mathbb{Z}[X]/(X^N + 1)$. A TFHE encryption of a single bit $m_i \in \{0, 1\}$ is a vector $\mathsf{ct}_i = (b_i, \mathbf{a}_i) \in \mathbb{T}^{n+1}$ satisfying $b_i + \langle \mathbf{a}_i, \mathbf{s} \rangle \approx \frac{1}{4} m_i \pmod 1$. One can homomorphically evaluate a bootstrapped NAND (or any other binary) gate $m = m_1 \bar{\wedge} m_2$ on encrypted bits by evaluating a linear combination followed by a fast bootstrapping. The linear combination outputs a ciphertext $\mathsf{ct}' = (b', \mathbf{a}')$ such that $b' + \langle \mathbf{a}', \mathbf{s} \rangle \approx \frac{1}{2} m \pmod 1$. The fast bootstrapping starts with an accumulation, that exploits the RGSW scheme to evaluate the decryption formula and recover the scaling factor of $\frac{1}{4}$. It requires a bootstrapping key which is a sequence of RGSW encryptions of s_i encrypted by $z(X)$. Finally, we extract an LWE ciphertext $\mathsf{ct}'' = (b'', \mathbf{a}'') \in \mathbb{T}^{N+1}$ such that $b'' + \langle \mathbf{a}'', \mathbf{z} \rangle \approx \frac{1}{4} m \pmod 1$ for $\mathbf{z} = (z_0, \ldots, z_{N-1})$ and run the key-switching process to recover the original LWE secret $\mathbf{s}$.

We generalize TFHE to support the homomorphic computation on ciphertexts encrypted under independently generated keys. Each party independently generates the LWE secret $\mathbf{s}_i \in \mathbb{Z}^n$ and the RLWE secret $z_i(X) \in R$. An encryption of $m \in \{0,1\}$ will be a vector of the form $\overline{\mathsf{ct}} = (b, \mathbf{a}_1, \ldots, \mathbf{a}_k) \in \mathbb{T}^{kn+1}$ such that $b + \langle \mathbf{a}_1, \mathbf{s}_1 \rangle + \cdots + \langle \mathbf{a}_k, \mathbf{s}_k \rangle \approx \frac{1}{4}m \pmod 1$ where k denotes the number of involved parties and $\mathbf{s}_i = (s_{ij})_{j\in[n]}$ are their LWE secrets. The homomorphic evaluation of a NAND gate consists of the following three phases.

First, we evaluate the linear combination for the NAND gate $m = m_1 \barwedge m_2$ on encrypted bits m_1, m_2 and return a ciphertext $\overline{\mathsf{ct}}' = (b', \mathbf{a}'_1, \ldots, \mathbf{a}'_k)$ satisfying $b' + \sum_{i\in[k]} \langle \mathbf{a}'_i, \mathbf{s}_i \rangle \approx \frac{1}{2}m \pmod 1$. The evaluation is done after arranging the entries and extending the dimension of input ciphertexts to share the same secret.

In the second step, we extract the most significant bits $\tilde{b} = \lfloor 2N \cdot b' \rceil$ and $\tilde{\mathbf{a}}_i = \lfloor 2N \cdot \mathbf{a}'_i \rceil$, and initialize the accumulator $\mathsf{ACC} = (-\frac{1}{8}X^{\tilde{b}} \cdot h(X), \mathbf{0}) \in T^{k+1}$ for the testing polynomial $h(X) = \sum_{-N/2<d<N/2} X^d$. Then, we recursively compute the Mux gate (choice function) based on the external product with an RGSW encryption of $X^{s_{ij}}$. The main difference between our MKHE scheme and TFHE is that we have to generate an RLWE encryption of $X^{s_{ij}}$ with respect to the *concatenated* RLWE secret $\bar{\mathbf{z}} = (1, z_1, \ldots, z_k) \in R^{k+1}$. We adapt and modify the extension algorithm in previous works [14, 27, 5] which combines a single-key RGSW encryption with auxiliary information about encryption randomness with public information of other parties and returns an RGSW encryption of the same message under the concatenation of individual secrets. Our algorithm returns a ciphertext of dimension $(k+1)$ which is almost half of the previous one, and it is based on the Common Reference String (CRS) model as in prior works. The output ACC is an RLWE encryption of $\frac{1}{8} - \frac{1}{8}X^{\tilde{b}+\sum_{i=1}^{k}\langle \tilde{\mathbf{a}}_i, \mathbf{s}_i\rangle} \cdot h(X)$ whose constant term is approximately equal to $\frac{1}{4}m$ as desired since $\tilde{b} + \sum_{i=1}^{k} \langle \tilde{\mathbf{a}}_i, \mathbf{s}_i \rangle \approx N \cdot m \pmod{2N}$. The computational costs of our bootstrapping pipeline mainly depends on the second accumulation step whose complexity grows quadratically with the number of involved parties.

Finally, we extract an LWE encryption from ACC and perform the multi-key-switching procedure. An LWE ciphertext $\overline{\mathsf{ct}}'' = (b'', \mathbf{a}''_1, \ldots, \mathbf{a}''_k) \in \mathbb{T}^{1+kN}$ from ACC satisfies $b'' + \sum_{i=1}^{k} \langle \mathbf{a}''_i, \mathbf{z}_i \rangle \approx \frac{1}{4}m \pmod 1$ where $\mathbf{z}_i \in \mathbb{Z}^N$ is the coefficient vector of z_i. The multi-key-switching takes this ciphertext as an input to generate an LWE ciphertext encrypting the same message under $(\mathbf{s}_1, \ldots, \mathbf{s}_k)$ by repeating the ordinary key-switching procedure from $\mathbf{z}_i$ to $\mathbf{s}_i$.

Related Works. López-Alt et al. [25] firstly proposed an MKHE scheme based on the NTRU assumption. Clear and McGoldrick [14] introduced an LWE-based construction, and it was significantly simplified by Mukherjee and Wichs [27]. This scheme is *single-hop* for keys where the list of parties has to be known before the computation. This work was extended in concurrent researches by Peikert-Shiehian [28] and Brakerski-Perlman [5] which design multi-hop (dynamic for keys) MKHEs. Chen, Zhang and Wang [8] constructed a scheme which can encrypt a ring element compared to a single bit of prior works.

We summarize the performance of recent MKHE schemes in Table 1. We only consider the second (main) one between two schemes described in [28]. Both [28] #2 and [8] are leveled so a large constant (depending on the maximum level of a circuit to be evaluated) is hidden in the $\tilde{O}(\cdot)$ notation. The space

Scheme	Space		Time		Bootstrap
	Type	Complexity	Type	Complexity	
CZW17 [8]	EvalKey Ciphertext	$\tilde{O}(k^3 n)$ $\tilde{O}(kn)$	EvalKey Gen Hom Mult	$\tilde{O}(k^4 n)$ $\tilde{O}(k^3 n)$	No
PS16 #2 [28]	PK Ciphertext	$\tilde{O}(kn^4)$ $\tilde{O}(k^2 n^2)$	Hom Mult	$\tilde{O}(k^{2.37} n^{2.37})$	No
BP16 [5]	PK Ciphertext	$\tilde{O}(kn^3)$ $\tilde{O}(kn)$	Hom NAND	$\mathrm{poly}(k, n)$	Yes
This work	Eval Key Ciphertext	$\tilde{O}(k^2 n^2)$ $\tilde{O}(kn)$	Eval Key Gen Hom NAND	$\tilde{O}(k^2 n^2)$ $\tilde{O}(k^2 n^2)$	Yes

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

and time complexity of [8] grow rapidly as the number of parties increases. Its complexity is quasi-linear on the security parameter, however, our scheme can be implemented using a smaller parameter.[3]

[5] is based on an abstract bootstrapping method which evaluates a huge branching program of length $L = \mathrm{poly}(k, n)$ representing the decryption circuit. Its memory requirement grows linearly on the number of parties but it comes from a space-time tradeoff. This idea is easily applicable to our scheme to reduce down the space complexity (see Section 4.2 for details).

2 Background

2.1 Notation

All logarithms are in base two unless otherwise indicated. We denote vectors in bold, e.g. $\mathbf{a}$, and matrices in upper-case bold, e.g. $\mathbf{A}$. We denote by $\langle \cdot, \cdot \rangle$ the usual dot product of two vectors. For a real number r, $\lfloor r \rceil$ denotes the nearest integer to r, rounding upwards in case of a tie. We use $x \leftarrow D$ to denote the sampling x according to distribution D. For a finite set S, $U(S)$ denotes the uniform distribution on S. For a real $\alpha > 0$, D_α denotes the Gaussian distribution of variance α^2. We let λ denote the security parameter throughout the paper: all known valid attacks against the cryptographic scheme under scope should take $\Omega(2^\lambda)$ bit operations.

[3] For the reader who is familiar with previous HE schemes, we note that a gate boostrapping of TFHE [10, 13] takes only 13ms despite its quadratic complexity with the security parameter. A single multiplication of ring-based schemes [4, 18] with a quasi-linear complexity usually takes longer.

2.2 TLWE and TRLWE

The TFHE scheme, presented for the first time in [10], is based on the TLWE (resp. TRLWE) problem, which is the torus variant of the LWE (resp. RLWE) problem. Instead of working over $\mathbb{Z}/q\mathbb{Z}$, or over the ring $\mathbb{Z}[X]/(X^N+1) \bmod q$ in the ring variant, in TFHE we work over the real Torus $\mathbb{T} = \mathbb{R} \bmod 1$ and over $T = \mathbb{T}[X]/(X^N+1)$, the set of cyclotomic polynomials over $\mathbb{T}$ for a power-of-two integer N. In this section and in the following one we present an overview of the TFHE scheme: for more details we refer to [13].

We denote by $R = \mathbb{Z}[X]/(X^N+1)$ the set of cyclotomic polynomials over $\mathbb{Z}$. Then, we observe that $\mathbb{T}$ and T are modules over $\mathbb{Z}$ and R, respectively. This means that they are groups with respect to the addition and they are provided with an external product by an integer or an integer polynomial.

A TLWE sample is a pair $(b, \mathbf{a}) \in \mathbb{T}^{n+1}$, where $\mathbf{a}$ is sampled uniformly over $\mathbb{T}^n$ and $b = \langle \mathbf{a}, \mathbf{s} \rangle + e$. The secret key $\mathbf{s}$ and error e are sampled from a key distribution χ on $\mathbb{Z}^n$ and a Gaussian with standard deviation $\alpha > 0$.

By following the same path, a TRLWE sample is a pair of polynomials $(b(X), a(X)) \in T^2$, where $a(X)$ is sampled uniformly and $b(X) = a(X) \cdot z(X) + e(X)$. The secret key $z(X)$ is an integer polynomial of degree N sampled from a key distribution ψ on R and the error polynomial $e(X)$ is sampled from a Gaussian distribution with standard deviation β.

For $a, b \in \mathbb{R}$ (resp. T), we denote by $a \approx b \pmod 1$ if $a = b + e \pmod 1$ for a small error $e \in \mathbb{R}$ (resp. $\mathbb{R}[X]/(X^N+1)$).

We can then define two problems for both TLWE and TRLWE:

- Decision problem: for a fixed TLWE secret $\mathbf{s}$ (resp. TRLWE secret $z(X)$), distinguish the uniform distribution over $\mathbb{T}^{n+1}$ (resp. T^2) from the TLWE (resp. TRLWE) samples.
- Search problem: given arbitrarily many samples from the TLWE (resp. TRLWE) distribution, find the secret $\mathbf{s}$ (resp. $z(X)$).

TLWE samples can be used to encrypt Torus messages. By fixing the message space as a discrete subset $\mathcal{M} \subseteq \mathbb{T}$, a message $\mu \in \mathcal{M}$ can be encrypted by adding the trivial TLWE sample $(\mu, \mathbf{0})$ to a TLWE sample generated as described in previous paragraphs. Then, the corresponding ciphertext ct is a pair $(b, \mathbf{a}) \in \mathbb{T}^{n+1}$, with $b = -\langle \mathbf{a}, \mathbf{s} \rangle + e + \mu$. In order to decrypt, we compute the phase $\varphi_{\mathbf{s}}$ of the ciphertext ct, which is equal to $\varphi_{\mathbf{s}}(\mathsf{ct}) = b + \langle \mathbf{a}, \mathbf{s} \rangle$, and we approximate it to the nearest message possible in $\mathcal{M}$ to retrieve μ. By following the same footstep, we can use TRLWE samples to encrypt torus polynomial messages in T.

Thanks to the $\mathbb{Z}$-module structure of the torus and to the R-module structure of T, the TLWE and TRLWE samples have additive homomorphic properties. The external integer homomorphic multiplication can be performed thanks to the TRGSW ciphertexts we define in the next section.

2.3 TRGSW and External Product

For a base integer $B \geq 2$ and a degree d, we call $\mathbf{g} = (1/B, \ldots, 1/B^d)$ the *gadget vector*. For an integer $k \geq 1$, the gadget matrix is defined by

$$\mathbf{G}_k = \mathbf{I}_k \otimes \mathbf{g} = \begin{bmatrix} \mathbf{g} & 0 & \ldots & 0 \\ 0 & \mathbf{g} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \mathbf{g} \end{bmatrix} \in \mathbb{T}^{dk \times k}.$$

For any $\mathbf{u} \in \mathbb{T}^k$, we define its base decomposition by a dk-dimensional vector $\mathbf{v} = \mathbf{G}_k^{-1}(\mathbf{u})$ with coefficients in $\mathbb{Z} \cap (-B/2, B/2]$ which minimizes $\|\mathbf{v}^T \cdot \mathbf{G}_k - \mathbf{u}^T\|_\infty$. The decomposition error $\|\mathbf{v}^T \cdot \mathbf{G}_k - \mathbf{u}^T\|_\infty$ is bounded by $1/(2B^{d+1})$.

We identify an arbitrary element of T to the vector of its coefficients in $\mathbb{T}^N$, and naturally extend the base decomposition $\mathbf{G}_k^{-1}(\cdot)$ to a function $T^k \to R^{dk}$ by applying the basic decomposition function coefficient wisely.

Then, we can define the TRGSW samples as the torus variant of RGSW samples, in the same way as we did in previous section[4]. For a fixed TRLWE secret $s(X)$, we define a TRGSW sample as $\mathbf{C} = \mathbf{Z} + \mu \cdot \mathbf{G}_2$, where each line of the matrix $\mathbf{Z} \in T^{d \times 2}$ is a TRLWE encryption of 0, $\mathbf{G}_2$ is the gadget matrix and the message $\mu \in R$ is an integer polynomial.

TRGSW samples are homomorphic with respect to the addition and to an internal multiplication. Furthermore, an external product, noted $\boxdot$, with TRLWE can be defined as $\mathbf{A} \boxdot \mathbf{b} = \mathbf{G}_2^{-1}(\mathbf{b}) \cdot \mathbf{A}$, for all TRLWE samples $\mathbf{b}$ and TRGSW samples $\mathbf{A}$ encrypted with the same secret key. In the following sections, we define a variant of the TRGSW samples and an adapted external product. The internal product between two TRGSW samples $\mathbf{A}$ and $\mathbf{B}$ encrypted with the same secret key can be defined as a list of independent external products between the cipher $\mathbf{A}$ and the lines composing the cipher $\mathbf{B}$.

The scheme TFHE has been implemented and is publicly available at [12]. In Section 5 we present some experimental results we obtained by implementing our Multi-Key scheme on top of the TFHE library.

In the rest of the paper, in order to lighten the notations, we will abandon the 'T' notation in front of LWE, RLWE and RGSW.

3 Basic Schemes

In this section, we present the LWE [29] and RGSW [20, 17] schemes and describe some extended algorithms that will be used in our MKHE scheme.

[4] We define only the Ring version TRGSW, since this is the only sample we need in this paper. TGSW can be defined in the same way. For more details we refer to [13].

3.1 Multi-Key-Switching on LWE Ciphertexts

We first describe the standard LWE scheme and generalize its key-switching algorithm to the multi-key case.

$\underline{\texttt{LWE.Setup}(1^\lambda)}$: It takes the security parameter as input and generates the LWE dimension n, key distribution χ, error parameter α, decomposition base B_{ks} and degree d_{ks}. Return the public parameter $pp^{\texttt{LWE}} = (n, \chi, \alpha, B_{ks}, d_{ks})$.

An LWE secret $\mathbf{s}$ is sampled from the distribution $\leftarrow \chi$.

We use the key-switching gadget vector $\mathbf{g}_{ks} = (1/B_{ks}, \ldots, 1/B_{ks}^d)$. Recall that the base decomposition algorithm with respect to $\mathbf{g}_{ks}$ transforms an element $a \in \mathbb{R}$ into the d_{ks}-dimensional vector $\mathbf{g}_{ks}^{-1}(a)$ with coefficients in $\mathbb{Z} \cap (-B_{ks}/2, B_{ks}/2]$ which minimizes $|a - \langle \mathbf{g}_{ks}^{-1}(a), \mathbf{g}_{ks} \rangle|$.

$\underline{\texttt{LWE.Enc}(m)}$: This is a standard LWE encryption which takes a bit $m \in \{0, 1\}$ as an input. It samples $a \leftarrow U(\mathbb{T}^n)$ and $e \leftarrow D_\alpha$, and returns the ciphertext $\mathsf{ct} = (b, \mathbf{a}) \in \mathbb{T}^{n+1}$ for $b = \frac{1}{4}m - \langle \mathbf{a}, \mathbf{s} \rangle + e \pmod 1$.

Note that the scaling factor is 1/4, as in FHEW [17] or TFHE [10]. We described a symmetric encryption for simplicity, but this algorithm can be replaced by any LWE-style encryption schemes such as public key encryption [24]. The only requirement is that the output ciphertext should be a vector $\mathsf{ct} = (b, \mathbf{a}) \in \mathbb{T}^{n+1}$ satisfying $b + \langle \mathbf{a}, \mathbf{s} \rangle \approx \frac{1}{4}m \pmod 1$.

$\underline{\texttt{LWE.KSGen}(\mathbf{t}, \mathbf{s})}$: For given LWE secrets $\mathbf{t} \in \mathbb{Z}^N$ and $\mathbf{s} \in \mathbb{Z}^n$, it returns the key-switching key $\mathsf{KS} = \{\mathbf{KS}_j\}_{j \in [N]} \in (\mathbb{T}^{d_{ks} \times (n+1)})^N$ from $\mathbf{t}$ to $\mathbf{s}$. For each $j \in [N]$, the j-th component is generated by $\mathbf{KS}_j = [\mathbf{b}_j | \mathbf{A}_j]$ where $\mathbf{A}_j \leftarrow U(\mathbb{T}^{d_{ks} \times n})$, $\mathbf{e}_j \leftarrow D_\beta^{d_{ks}}$ and $\mathbf{b}_j = -\mathbf{A}_j \mathbf{s} + \mathbf{e}_j + t_j \cdot \mathbf{g}_{ks} \pmod 1$.

We can transform an LWE ciphertext corresponding to $\mathbf{t}$ into another LWE encryption of the same message under the secret $\mathbf{s}$ using a key-switching key $\mathbf{KS} \leftarrow \texttt{LWE.KSGen}(\mathbf{t}, \mathbf{s})$.

We consider the notion of extended LWE encryption and the multi-key-switching procedure. For k LWE secrets $\mathbf{s}_1, \ldots, \mathbf{s}_k \in \mathbb{Z}^n$, an extended ciphertext $\overline{\mathsf{ct}} = (b, \mathbf{a}_1, \ldots, \mathbf{a}_k) \in \mathbb{T}^{kn+1}$ will be called an encryption of $m \in \{0, 1\}$ with respect to the concatenated secret $\overline{\mathbf{s}} = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$ if $\langle \overline{\mathsf{ct}}, (1, \overline{\mathbf{s}}) \rangle = b + \sum_{i=1}^k \langle \mathbf{a}_i, \mathbf{s}_i \rangle \approx \frac{1}{2}m \pmod 1$.

$\underline{\texttt{LWE.MKSwitch}(\overline{\mathsf{ct}}, \overline{\mathsf{KS}})}$: For a given ciphertext $\mathsf{ct} = (b, \mathbf{a}_1, \ldots, \mathbf{a}_k) \in \mathbb{T}^{kN+1}$ and a sequence of the key-switching keys $\overline{\mathsf{KS}} = \{\mathsf{KS}_i = \{\mathbf{KS}_{i,j}\}_{j \in [N]}\}_{i \in [k]}$, compute $(b'_i, \mathbf{a}'_i) = \sum_{j=1}^N \mathbf{g}_{ks}^{-1}(a_{i,j}) \cdot \mathbf{KS}_{i,j} \pmod 1$ for all $i \in [k]$ and let $b' = b + \sum_{i=1}^k b'_i \pmod 1$. Return the ciphertext $\overline{\mathsf{ct}}' = (b', \mathbf{a}'_1, \ldots, \mathbf{a}'_k) \in \mathbb{T}^{kn+1}$.

This multi-key-switching algorithm takes as the input an extended ciphertext $\overline{\mathsf{ct}} \in \mathbb{T}^{kN+1}$ corresponding to $\overline{\mathbf{t}} = (\mathbf{t}_1, \ldots, \mathbf{t}_k)$ and a sequence of key-switching keys from $\mathbf{t}_i$ to $\mathbf{s}_i$ and returns an encryption of the same message under $\overline{\mathbf{s}} = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$.

Security. The j-th component $\mathbf{KS}_j$ of a key-switching key $\mathsf{KS} = \{\mathbf{KS}_j\}_{j\in[N]}$ from $\mathbf{t} \in \mathbb{Z}^N$ to $\mathbf{s} \in \mathbb{Z}^n$ is generated by adding $t_j \cdot \mathbf{g}_{ks}$ to the first column of a matrix in $\mathbb{T}^{d_{ks}\times(n+1)}$ whose rows are LWE instances under the secret $\mathbf{s}$. Therefore, $\mathsf{KS} \leftarrow \mathtt{LWE.KSGen}(\mathbf{t}, \mathbf{s})$ is computationally indistinguishable from the uniform distribution over $(\mathbb{T}^{d_{ks}\times(n+1)})^N$ under the LWE assumption with parameter (n, χ, β) if $\mathbf{s}$ is sampled according to χ.

Correctness. We show that if the input $\mathsf{ct} = (b, \mathbf{a}_1, \dots, \mathbf{a}_k)$ is an LWE ciphertext encrypted by $\bar{\mathbf{t}} = (\mathbf{t}_1, \dots, \mathbf{t}_k)$ and KS_i's are key-switching keys from $\mathbf{t}_i \in \mathbb{Z}^N$ to $\mathbf{s}_i \in \mathbb{Z}^n$ for $i \in [k]$, respectively, then the output ciphertext encrypts the same message under the concatenated secret $\bar{\mathbf{s}} = (\mathbf{s}_1, \dots, \mathbf{s}_k)$. The correctness of this algorithm is simply shown by the following equation:

$$\begin{aligned}\langle \overline{\mathsf{ct}}', (1, \bar{\mathbf{s}})\rangle &= b + \sum_{i=1}^{k} (b_i' + \langle \mathbf{a}_i', \mathbf{s}_i\rangle) \\ &\approx b + \sum_{i=1}^{k}\sum_{j=1}^{N} \langle \mathbf{g}_{ks}^{-1}(a_{i,j}), t_{i,j} \cdot \mathbf{g}_{ks}\rangle \approx \langle \overline{\mathsf{ct}}, (1, \bar{\mathbf{t}})\rangle \pmod 1.\end{aligned}$$

Therefore, $\overline{\mathsf{KS}} = \{\mathsf{KS}_i\}_{i\in[k]}$ can be considered as a key-switching key from $\bar{\mathbf{t}} \in \mathbb{Z}^{kN}$ to $\bar{\mathbf{s}} \in \mathbb{Z}^{kn}$.

3.2 Extension of RGSW Ciphertexts and Generalized External Product

The GSW scheme has been widely used in the construction of cryptographic primitives due to its intrinsic characteristics. For example, the ciphertext extension procedure in recent MKHE schemes [5, 28, 8] is based on the GSW homomorphic arithmetic without evaluation key. Meanwhile, HE schemes with low-latency bootstrapping [17, 10] enjoy the compatibility of GSW with classical (R)LWE ciphertexts (e.g. external product, key-switching).

In this section, we present a variant of RGSW which supports a new ciphertext extension and a generalized external product on extended ciphertexts. It is similar to the existing techniques in previous work on MKHE, but our solution achieves a better performance by reducing the dimension of ciphertexts and the complexity of operations.

$\underline{\mathtt{RGSW.Setup}(1^\lambda)}$: It takes as input the secret parameter λ.

1. Set the RLWE dimension N which is a power of two.
2. Set the key distribution ψ over R and choose the error parameter α.
3. Set the base integer $B \geq 2$ and the decomposition degree d.
4. Generate a random vector $\mathbf{a} \leftarrow U(T^d)$.

Return the public parameter $pp^{\mathtt{RGSW}} = (N, \psi, \alpha, B, d, \mathbf{a})$.

Our scheme is based on the CRS model since it requires all participants to share a randomly generated vector $\mathbf{a}$. The RGSW parameter should be chosen appropriately so that the RLWE problem with parameter (N, ψ, α) achieves at least λ-bit security level.

$\underline{\texttt{RGSW.KeyGen}(pp^{\texttt{RGSW}})}$: Sample $z \leftarrow \psi$ and return the secret key $\mathsf{SK} \leftarrow z$. We write $\mathbf{z} = (1, z)$. Sample an error vector $\mathbf{e} \leftarrow D_\alpha^d$ and set the public key as $\mathsf{PK} \leftarrow \mathbf{P} = [\mathbf{b}|\mathbf{a}] \in T^{d\times 2}$ where $\mathbf{b} = -\mathbf{a} \cdot z + \mathbf{e} \pmod 1$. Return $(z, \mathbf{P})$.

$\underline{\texttt{RGSW.UniEnc}(\mu, z, \mathbf{P})}$: For an input plaintext $\mu \in R$, a secret key $\mathbf{z}$ and a public key $\mathbf{P}$, it generates and returns the ciphertexts $(\mathbf{C}, \mathbf{D}, \mathbf{F}) \in (T^{d\times 2})^3$ as follows:

1. Sample $\mathbf{c}_1 \leftarrow U(T^d)$ and $\mathbf{e}_c \leftarrow D_\alpha^d$. Output the ciphertext $\mathbf{C} = [\mathbf{c}_0|\mathbf{c}_1] \in T^{d\times 2}$ where $\mathbf{c}_0 = -z \cdot \mathbf{c}_1 + \mathbf{e}_c + \mu \cdot \mathbf{g} \pmod 1$.
2. Sample a randomness $r \leftarrow \psi$ and an error matrix $\mathbf{E} \leftarrow D_\alpha^{d\times 2}$. Output the ciphertext $\mathbf{D} = r \cdot \mathbf{P} + \mathbf{E} + [\mathbf{0}|\mu \cdot \mathbf{g}] \pmod 1$ in $T^{d\times 2}$.
3. Sample $\mathbf{f}_1 \leftarrow U(T^d)$ and $\mathbf{e}_f \leftarrow D_\alpha^d$. Output the ciphertext $\mathbf{F} = [\mathbf{f}_0|\mathbf{f}_1] \in T^{d\times 2}$ where $\mathbf{f}_0 = -z \cdot \mathbf{f}_1 + \mathbf{e}_f + r \cdot \mathbf{g} \pmod 1$.

Our RGSW scheme takes as an input the pair of secret and public keys. The third component $\mathbf{F}$ encrypts the randomness $r \in R$ used in the generation of $\mathbf{D}$. Note that an uni-encryption $(\mathbf{C}, \mathbf{D}, \mathbf{F})$ is 1.5 times the size of ordinary RGSW ciphertexts in $T^{2d\times 2}$.

The following algorithm transforms an uni-encryption $(\mathbf{C}, \mathbf{D}, \mathbf{F})$ encrypted by the i-th party into an RGSW ciphertext corresponding to the set of k parties based on their public key $\{\mathbf{P}_j\}_{j\in[k]}$.

$\underline{\texttt{RGSW.Expand}((\mathbf{C}, \mathbf{D}, \mathbf{F}), i, \{\mathbf{P}_j\}_{j\in[k]})}$: It takes as the input a ciphertext $(\mathbf{C}, \mathbf{D}, \mathbf{F}) \in (T^{d\times 2})^3$, an index $i \in [k]$, and a sequence of public keys $\{\mathbf{P}_j\}_{j\in[k]}$. For each $j \in [k]$, compute the vectors $\mathbf{x}_j, \mathbf{y}_j \in R_Q^d$ by $\mathbf{x}_j[\ell] = \langle \mathbf{g}^{-1}(\mathbf{b}_j[\ell] - \mathbf{b}_i[\ell]), \mathbf{f}_0\rangle$ and $\mathbf{y}_j[\ell] = \langle \mathbf{g}^{-1}(\mathbf{b}_j[\ell] - \mathbf{b}_i[\ell]), \mathbf{f}_1\rangle$ for all $\ell \in [d]$, i.e., $[\mathbf{x}_j|\mathbf{y}_j] = \mathbf{M}_j\mathbf{F} \in T^{d\times 2}$ where $\mathbf{M}_j \in R^{d\times d}$ is the matrix of which ℓ-th row vector is $\mathbf{g}^{-1}(\mathbf{b}_j[\ell] - \mathbf{b}_i[\ell])$. Return the expanded ciphertext

$$\overline{\mathbf{C}} = \begin{bmatrix} \mathbf{c}_0 & \mathbf{0} & \cdots & \mathbf{c}_1 & \cdots & \mathbf{0} \\ \mathbf{d}_0 + \mathbf{x}_1 & \mathbf{d}_1 & \cdots & \mathbf{y}_1 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{d}_0 & \mathbf{0} & \cdots & \mathbf{d}_1 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{d}_0 + \mathbf{x}_k & \mathbf{0} & \cdots & \mathbf{y}_k & \cdots & \mathbf{d}_1 \end{bmatrix} \in T^{d(k+1)\times(k+1)}.$$

$\underline{\texttt{RGSW.ExtProd}(\overline{\mathbf{c}}, \overline{\mathbf{C}})}$: For an RLWE ciphertext $\overline{\mathbf{c}} \in T^{k+1}$ and an RGSW ciphertext $\overline{\mathbf{C}} \in T^{d(k+1)\times(k+1)}$, compute and return the ciphertext $\overline{\mathbf{c}}' = \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}) \cdot \overline{\mathbf{C}} \pmod 1$ in T^{k+1}.

A ciphertext $\overline{\mathbf{C}} \in T^{d(k+1)\times(k+1)}$ is called an extended RGSW encryption of $\mu \in R$ corresponding to a secret $\overline{\mathbf{z}} = (1, z_1, \ldots, z_k) \in R^{k+1}$ if $\overline{\mathbf{C}}\overline{\mathbf{z}} \approx \mu \cdot \mathbf{G}_{k+1}\overline{\mathbf{z}} \pmod 1$. Based on the public information $\mathbf{P}_j$, our ciphertext extension algorithm transforms an uni-encryption $(\mathbf{C}, \mathbf{D}, \mathbf{F})$ corresponding to the i-th secret $(\mathsf{SK}_i, \mathsf{PK}_i) = (z_i, \mathbf{P}_i)$ into an extended RGSW ciphertext $\overline{\mathbf{C}}$ which encrypts the same message under the concatenated secret $\overline{\mathbf{z}} = (1, z_1, \ldots, z_k) \in R^{k+1}$. It requires $O(kd^2)$ polynomial operations, and we need only $(2k+2)d$ polynomials to represent $\overline{\mathbf{C}}$ due to its sparsity.

Our scheme also provides the external product algorithm (though our scheme has no RLWE encryption) which multiplies an RGSW ciphertext to an RLWE ciphertext where both are encrypted by the same key $\overline{\mathbf{z}}$. If $\overline{\mathbf{C}}$ is an output of the RGSW expansion algorithm (so is sparse), then the external product takes $O(kd)$ polynomial operations.

Security. We claim that the distribution

$$\begin{aligned}\mathcal{D}_0 = \{(\mathbf{P}, \mathbf{C}, \mathbf{D}, \mathbf{F}) :& (z, \mathbf{P}) \leftarrow \texttt{RGSW.KeyGen}(pp^{\texttt{RGSW}}),\\ & (\mathbf{C}, \mathbf{D}, \mathbf{F}) \leftarrow \texttt{RGSW.UniEnc}(\mu, z, \mathbf{P})\}\end{aligned}$$

is computationally indistinguishable from the uniform distribution over $(T^{d\times 2})^4$ for any $\mu \in R$ under the RLWE assumption with parameter (N, ψ, α). We consider the following distributions:

$$\begin{aligned}\mathcal{D}_1 = \{(\mathbf{P}, \mathbf{C}, \mathbf{D}, \mathbf{F}) :& \mathbf{P}, \mathbf{C}, \mathbf{F} \leftarrow U(T^{d\times 2}), r \leftarrow \psi, \mathbf{E} \leftarrow D_\alpha^{d\times 2},\\ & \mathbf{D} = r \cdot \mathbf{P} + \mathbf{E} + [\mathbf{0}|\mu \cdot \mathbf{g}] \pmod 1.\},\\ \mathcal{D}_2 = \{(\mathbf{P}, \mathbf{C}, \mathbf{D}, \mathbf{F}) :& \mathbf{P}, \mathbf{C}, \mathbf{D}, \mathbf{F} \leftarrow U(T^{d\times 2})\}.\end{aligned}$$

First, we can change the RLWE samples $\mathbf{P}$, $\mathbf{C}$, and $\mathbf{F}$ of the secret z to random matrices. Then, $\mathbf{D}$ is changed to a uniform distribution using the RLWE assumption of secret r again. Since $\mathcal{D}_2$ is independent from μ, our RGSW scheme is IND-CPA secure.

Correctness. We briefly show the correctness of our ciphertext expansion and external product algorithms. We refer the reader to Appendix A for a detailed noise analysis.

Let $(z_j, \mathbf{P}_j) \leftarrow \texttt{RGSW.KeyGen}(pp^{\texttt{RGSW}})$ be independently generated RGSW keys for $j \in [k]$ and let $(\mathbf{C}, \mathbf{D}, \mathbf{F}) \leftarrow \texttt{RGSW.UniEnc}(\mu, z_i, \mathbf{P}_i)$ be an uni-encryption of μ under the i-th key. We claim that $\overline{\mathbf{C}} \leftarrow \texttt{RGSW.Expand}((\mathbf{C}, \mathbf{D}, \mathbf{F}), i, \{\mathbf{P}_j\}_{j\in[k]})$ is an RGSW encryption of μ corresponding to the concatenated secret $\overline{\mathbf{z}} = (1, z_1, \ldots, z_k)$, i.e., $\overline{\mathbf{C}}\overline{\mathbf{z}} \approx \mu \cdot \mathbf{G}_{k+1}\overline{\mathbf{z}} \pmod 1$. Since $\mathbf{c}_0 + z_i \cdot \mathbf{c}_1 = \mathbf{C}\mathbf{z}_i \approx \mu \cdot \mathbf{g} \pmod 1$, it suffices to show that $(\mathbf{d}_0 + \mathbf{x}_j) + z_j \cdot \mathbf{d}_1 + z_i \cdot \mathbf{y}_j \approx \mu z_j \cdot \mathbf{g} \pmod 1$ for all $j \in [k]$. It is derived from the following equations:

$$\begin{aligned}&\mathbf{d}_0 + z_j \cdot \mathbf{d}_1 \approx \mu z_j \cdot \mathbf{g} + r \cdot \mathbf{P}_i \mathbf{z}_j \approx \mu z_j \cdot \mathbf{g} + r \cdot (\mathbf{b}_i - \mathbf{b}_j) \pmod 1,\\ &\mathbf{x}_j + z_i \cdot \mathbf{y}_j = \mathbf{M}_j \mathbf{F} \mathbf{z}_i \approx r \cdot \mathbf{M}_j \mathbf{g} \approx r \cdot (\mathbf{b}_j - \mathbf{b}_i) \pmod 1.\end{aligned}$$

Now let us suppose that $\overline{\mathbf{c}} \in T^{k+1}$ is an RLWE ciphertext and $\overline{\mathbf{C}} \in T^{d(k+1)\times(k+1)}$ is an RGSW encryption of μ with respect to the secret $\overline{\mathbf{z}} \in R^{k+1}$, i.e., $\overline{\mathbf{C}}\overline{\mathbf{z}} \approx$

$\mu \cdot \mathbf{G}_{k+1}\overline{\mathbf{z}} \pmod 1$. Then their external product $\overline{\mathbf{c}}' \leftarrow \texttt{RGSW.ExtProd}(\overline{\mathbf{c}}, \overline{\mathbf{C}})$ satisfies that $\langle \overline{\mathbf{c}}', \overline{\mathbf{z}} \rangle = \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}) \cdot \overline{\mathbf{C}}\overline{\mathbf{z}} \approx \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}) \cdot \mu\mathbf{G}_{k+1}\overline{\mathbf{z}} \approx \mu \cdot \langle \overline{\mathbf{c}}, \overline{\mathbf{z}} \rangle \pmod 1$, as desired.

4 Our MKHE Scheme

4.1 Description

In this section, we explicitly describe an MKHE scheme based on the LWE and RGSW schemes. Our scheme can bootstrap a ciphertext after the evaluation of a binary gate as in TFHE [10], but it requires to pre-compute the bootstrapping key corresponding to the set of parties involved in a computation.

$\underline{\texttt{MKHE.Setup}(1^\lambda)}$:

- Run $\texttt{LWE.Setup}(1^\lambda)$ to generate the parameter $pp^{\texttt{LWE}} = (n, \chi, \alpha, B_{ks}, d_{ks})$.
- Run $\texttt{RGSW.Setup}(1^\lambda)$ to generate the parameter $pp^{\texttt{RGSW}} = (N, \psi, \beta, B, d, \mathbf{a})$.
- Return the generated public parameters $pp = (pp^{\texttt{RGSW}}, pp^{\texttt{LWE}})$.

$\underline{\texttt{MKHE.KeyGen}(pp)}$:

- Run $(z, \mathbf{P}) \leftarrow \texttt{RGSW.KeyGen}(pp^{\texttt{RGSW}})$ and set the public key as $\mathsf{PK} = \mathbf{P}$. We write $\mathbf{t} = (z_0, -z_{N-1}, \ldots, -z_1) \in \mathbb{Z}^N$ for $z(X) = z_0 + z_1 X + \cdots + z_{N-1}X^{N-1}$.
- Sample the LWE secret $\mathbf{s} = (s_1, \ldots, s_n) \leftarrow \psi$.
- Generate $(\mathbf{C}_\ell, \mathbf{D}_\ell, \mathbf{F}_\ell) \leftarrow \texttt{RGSW.UniEnc}(s_\ell, z, \mathbf{P})$ for $\ell \in [n]$ and set the bootstrapping key as $\mathsf{BK} = \{(\mathbf{C}_\ell, \mathbf{D}_\ell, \mathbf{F}_\ell)\}_{\ell \in [n]}$.
- Generate the key-switching key $\mathsf{KS} \leftarrow \texttt{LWE.KSGen}(\mathbf{t}, \mathbf{s})$.
- Return the secret key $\mathbf{s}$. Publish the triple $(\mathsf{PK}, \mathsf{BK}, \mathsf{KS})$ of public, bootstrapping, and key-switching keys.

We remark that for any $a(X) = a_0 + a_1 X + \cdots + a_{N-1}X^{N-1} \in T$ and the vector of its coefficients $\mathbf{a} = (a_0, \ldots, a_{N-1}) \in \mathbb{T}^N$, the constant term of $a(X) \cdot z(X) \in T$ is equal to $\langle \mathbf{a}, \mathbf{t} \rangle$ modulo 1.

$\underline{\texttt{MKHE.Enc}(m)}$: For an input bit $m \in \{0, 1\}$, run $\texttt{LWE.Enc}(m)$ and return an LWE encryption with the scaling factor $1/4$. The output ciphertext $\mathsf{ct} = (b, \mathbf{a}) \in \mathbb{T}^{n+1}$ satisfies $b + \langle \mathbf{a}, \mathbf{s} \rangle \approx \frac{1}{4}m \pmod 1$.

The dimension of a ciphertext increases after homomorphic computations. The indices of related parties should be stored together with a ciphertext for the correct decryption and homomorphic operations.

$\underline{\texttt{MKHE.Dec}(\overline{\mathsf{ct}}, \{\mathbf{s}_j\}_{j \in [k]})}$: For a ciphertext $\overline{\mathsf{ct}} = (b, \mathbf{a}_1, \ldots, \mathbf{a}_k) \in \mathbb{T}^{kn+1}$ and a tuple of secrets $(\mathbf{s}_1, \ldots, \mathbf{s}_k)$, return the bit $m \in \{0, 1\}$ which minimizes $|b + \sum_{j=1}^k \langle \mathbf{a}_j, \mathbf{s}_j \rangle - \frac{1}{4}m|$.

$\underline{\texttt{MKHE.NAND}(\overline{\mathsf{ct}}_1, \overline{\mathsf{ct}}_2, \{(\mathsf{PK}_j, \mathsf{BK}_j, \mathsf{KS}_j)\}_{j \in [k]})}$: It takes as input two LWE ciphertexts $\overline{\mathsf{ct}}_1 \in \mathbb{T}^{k_1 n+1}$ and $\overline{\mathsf{ct}}_2 \in \mathbb{T}^{k_2 n+1}$, where $[k]$ is the set of indices of the parties that are involved in either $\overline{\mathsf{ct}}_1$ or $\overline{\mathsf{ct}}_2$. For $j \in [k]$, we denote by $\mathsf{PK}_j = \mathbf{P}_j$,

$\mathsf{BK}_j = \{(\mathbf{C}_{j,\ell}, \mathbf{D}_{j,\ell}, \mathbf{F}_{j,\ell})\}_{\ell\in[n]}$ and KS_j the public key, bootstrapping key and key-switching key of the j-th party, respectively.

This algorithm consists of three phases. The first step expands the input LWE ciphertexts and evaluate the NAND gate homomorphically over encrypted plaintexts.

1-1. Extend $\overline{\mathsf{ct}}_1$ and $\overline{\mathsf{ct}}_2$ to the ciphertexts $\overline{\mathsf{ct}}'_1, \overline{\mathsf{ct}}'_2 \in \mathbb{T}^{kn+1}$ which encrypt the same messages under the concatenated secret key $\bar{\mathbf{s}} = (\mathbf{s}_1, \ldots, \mathbf{s}_k) \in \mathbb{Z}^{kn}$. It is simply done by rearranging the components and putting zeros in the empty slots.
1-2. Compute $\overline{\mathsf{ct}}' = (\frac{5}{8}, \mathbf{0}, \ldots, \mathbf{0}) - \overline{\mathsf{ct}}'_1 - \overline{\mathsf{ct}}'_2 \pmod 1$.

To be precise, if an input ciphertext $\overline{\mathsf{ct}}_i = (b_i, \mathbf{a}_{i,1}, \ldots, \mathbf{a}_{i,k_i})$ is an encryption corresponding to a tuple $(j_1, \ldots, j_{k_i}) \in [k]^{k_1}$ of indices, then (1-1) returns $\overline{\mathsf{ct}}'_i = (b_i, \mathbf{a}'_{i,1}, \ldots, \mathbf{a}'_{i,k})$ where $\mathbf{a}'_{i,j} = \begin{cases} \mathbf{a}_{i,\ell} & \text{if } j = j_\ell \text{ for some } \ell \in [k_i], \\ \mathbf{0} & \text{otherwise;} \end{cases}$ for $j \in [k]$. It is clear from the definition that $\langle \overline{\mathsf{ct}}_i, (1, \mathbf{s}_{j_1}, \ldots, \mathbf{s}_{j_{k_i}})\rangle = \langle \overline{\mathsf{ct}}'_i, (1, \bar{\mathbf{s}})\rangle$ for $\bar{\mathbf{s}} = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$.

If $\langle \overline{\mathsf{ct}}'_i, (1, \bar{\mathbf{s}})\rangle = \frac{1}{4}m_i + e_i \pmod 1$ for some errors $e_i \in \mathbb{R}$, then the output ciphertext satisfies that $\langle \overline{\mathsf{ct}}', (1, \bar{\mathbf{s}})\rangle = \frac{1}{2}m + e' \pmod 1$ for $m = NAND(m_1, m_2)$ and $e' = \pm\frac{1}{8} - e_1 - e_2$ which is bounded by $\frac{1}{4}$ when $|e_i| \le \frac{1}{16}$. The next step, called homomorphic *accumulator* [17], is to evaluate the decryption circuit of an extended LWE ciphertext using the external product of RGSW scheme for bootstrapping.

2-1. For $i \in [k]$ and $\ell \in [n]$, run $\overline{\mathbf{C}}_{i,\ell} \leftarrow \texttt{RGSW.Expand}((\mathbf{C}_{i,\ell}, \mathbf{D}_{i,\ell}, \mathbf{F}_{i,\ell}), \{\mathbf{P}_j\}_{j\in[k]})$ to generate an RGSW encryption of $s_{i,\ell}$ under the secret $\bar{\mathbf{z}} = (1, z_1, \ldots, z_k) \in R^{k+1}$. Return the shared bootstrapping key $\overline{\mathsf{BK}} := \{\overline{\mathbf{C}}_{i,\ell}\}_{i\in[k],\ell\in[n]}$.
2-2. Let $\overline{\mathsf{ct}}' = (b', \mathbf{a}'_1, \ldots, \mathbf{a}'_k) \in \mathbb{T}^{kn+1}$. Compute $\tilde{b} = \lfloor 2N \cdot b' \rceil$ and $\tilde{\mathbf{a}}_i = \lfloor 2N \cdot \mathbf{a}'_i \rceil$. Initialize the RLWE ciphertext as $\mathsf{ACC} = (-\frac{1}{8}h(X) \cdot X^{\tilde{b}}, \mathbf{0}) \in T^{k+1}$ where $h(X) = \sum_{-\frac{N}{2}<j<\frac{N}{2}} X^j = 1 + X + \cdots + X^{\frac{N}{2}-1} - X^{\frac{N}{2}+1} - \cdots - X^{N-1}$.
2-3. Let $\tilde{\mathbf{a}}_i = (\tilde{a}_{i,\ell})_{\ell\in[n]}$ for $i \in [k]$. Compute
$$\mathsf{ACC} \leftarrow \mathsf{ACC} + \texttt{RGSW.ExtProd}((X^{\tilde{a}_{i,\ell}} - 1) \cdot \mathsf{ACC}, \overline{\mathbf{C}}_{i,\ell})$$
recursively for all $i \in [k]$ and $\ell \in [n]$.
2-4. Return $\mathsf{ACC} \leftarrow (\frac{1}{8}, \mathbf{0}) + \mathsf{ACC} \pmod 1$.

In (2-1), we generate the bootstrapping key corresponding to the k parties involved in this computation. The accumulator ACC is initialized in (2-2) as a trivial RLWE encryption of $-\frac{1}{8}h(X) \cdot X^{\tilde{b}}$. The main computation is done in (2-3) based on the Mux gate [10]. In each step, it homomorphically selects one of ACC and $X^{\tilde{a}_{i,\ell}} \cdot \mathsf{ACC}$ using the encryption $\overline{\mathbf{C}}_{i,\ell}$ of $s_{i,\ell} \in \{0, 1\}$. The output is an RLWE ciphertext satisfying

$$\begin{aligned}\langle \mathsf{ACC}, \overline{\mathbf{z}} \rangle &\approx -\frac{1}{8} h(X) \cdot X^{\tilde{b} + \sum_{i=1}^{k} \langle \tilde{\mathbf{a}}_i, \mathbf{s}_i \rangle} \\ &= -\frac{1}{8} \left(\sum_{-\frac{N}{2} < j < \frac{N}{2}} X^j \right) \cdot X^{\tilde{b} + \sum_{i=1}^{k} \langle \tilde{\mathbf{a}}_i, \mathbf{s}_i \rangle} \pmod 1.\end{aligned}$$

Since $\tilde{b} + \sum_{i=1}^{k} \langle \tilde{\mathbf{a}}_i, \mathbf{s}_i \rangle \approx (2N) \cdot \langle \overline{\mathsf{ct}}', (1, \overline{\mathbf{s}}) \rangle \approx N \cdot m \pmod{2N}$, the constant term of $\langle \mathsf{ACC}, \overline{\mathbf{z}} \rangle$ is approximately equal to either $-\frac{1}{8}$ (if $m = 0$) or $\frac{1}{8}$ (otherwise; $m = 1$), which is $\frac{1}{4}m - \frac{1}{8}$. Finally, the term $\frac{1}{8}$ is cancelled out in (2-4).

In the last step, we transform ACC into an LWE ciphertext and run the multi-key-switching algorithm as follows.

3-1. For $\mathsf{ACC} = (c_0, c_1, \ldots, c_k) \in T^{k+1}$, let b'' be the constant term of c_0 and $\mathbf{a}_i''$ be the coefficient vector of c_i for $i \in [k]$. Construct the LWE ciphertext $\overline{\mathsf{ct}}'' = (b'', \mathbf{a}_1'', \ldots, \mathbf{a}_k'') \in \mathbb{T}^{kN+1}$.

3-2. Let $\overline{\mathsf{KS}} = \{\mathsf{KS}_i\}_{i \in [k]}$. Run the multi-key-switching algorithm and return the ciphertext $\overline{\mathsf{ct}} \leftarrow \mathtt{LWE.MKSwitch}(\overline{\mathsf{ct}}'', \overline{\mathsf{KS}})$.

As we noted above, $\langle \mathbf{a}_i'', \mathbf{t}_i \rangle \pmod 1$ is equal to the constant term of $c_i z_i$ for $i \in [k]$. Hence, (3-1) returns an LWE ciphertext $\overline{\mathsf{ct}}''$ satisfying $\langle \overline{\mathsf{ct}}'', (1, \overline{\mathbf{t}}) \rangle \approx \frac{1}{4}m \pmod 1$. The next step (3-2) switches the LWE key and outputs an LWE ciphertext $\overline{\mathsf{ct}} \in \mathbb{T}^{kn+1}$ such that $\langle \overline{\mathsf{ct}}, (1, \overline{\mathbf{s}}) \rangle \approx \frac{1}{4}m \pmod 1$, as desired.

Security. Our scheme is semantically secure under the (R)LWE assumptions described in the previous section, so the parameters $pp^{\mathtt{LWE}}$ and $pp^{\mathtt{RGSW}}$ should be chosen properly to achieve at least λ-bit of security level.

It also requires a circular security assumption because each party publishes uni-encryptions of $s_1, \ldots, s_n$ encrypted by z, and a key-switching key from $\mathbf{t} = (z_0, -z_{N-1}, \ldots, -z_1)$ to $\mathbf{s}$. However, our circular security assumption is exactly the same as one in TFHE [10], and it is no stronger than the circular security assumptions in *bootstrappable* HE schemes [19] such as [17, 22, 7, 9].

Correctness. Our scheme should satisfy the following conditions to guarantee its correctness:

– In (2-1), the quantized ciphertext $(\tilde{b}, \tilde{\mathbf{a}}_1, \ldots, \tilde{\mathbf{a}}_k) \in \mathbb{Z}_{2N}^{kn+1}$ should satisfy $\tilde{b} + \sum_{j=1}^{k} \langle \tilde{\mathbf{a}}_j, \mathbf{s}_j \rangle = N \cdot m + \tilde{e}$ for some $\tilde{e} \in \mathbb{Z}$ with $|\tilde{e}| < N/2$. This noise $\tilde{e}$ consists of two parts $\tilde{e} = 2N \cdot e' + e''$ for $e' = \pm\frac{1}{8} - e_1 - e_2$ from the step (1-2) and a rounding error $e'' = (\tilde{b} - 2N \cdot b') + \sum_{j=1}^{k} \langle \tilde{\mathbf{a}}_j - 2N \cdot \mathbf{a}_j', \mathbf{s}_j \rangle$.

– The error $e \in \mathbb{R}$ of an output LWE ciphertext $\overline{\mathsf{ct}}$ should be small enough for the correct decryption and further computations. It is the sum of the constant term of an RLWE error which is accumulated from the external products during (2-3), and the multi-key-switching error from (3-2).

We provide a rigorous noise estimation in Appendix A. We refer the reader to Section 5 for a recommended parameter set.

Type	Space
$\overline{\mathsf{ct}}$	$kn+1$
$\overline{\mathsf{BK}}$	$(2k+2)k \cdot nN \cdot d$
$\overline{\mathsf{KS}}$	$k \cdot (n+1)N \cdot d_{ks} \cdot B_{ks}$

Type	Time
$\overline{\mathsf{BK}}$ Generation	$O(k^2 \cdot nN \log N \cdot d^2)$
Accumulator	$O(k^2 \cdot nN \log N \cdot d)$
Multi-key-switching	$O(k \cdot nN \cdot d_{ks})$

Table 2. Space (number of torus elements) and time (number of scalar operations) complexity of our scheme.

Performance. We run the expand algorithm $k \cdot n$ times for the generation of bootstrapping key (2-1), and we recall that a single expansion algorithm takes $O(kd^2)$ polynomial operations. The total complexity is $O(k^2 N \log N n d^2) = \tilde{O}(k^2 N n)$ scalar operations since one polynomial operation takes $O(N \log N)$ scalar operations. The size of bootstrapping key grows quadratically with the number of parties since it consists of $k \cdot n$ expanded RGSW ciphertexts each of which can be represented by $(2k+2)d$ elements in T. Hence the size of $\overline{\mathsf{BK}}$ is $(2k+2)kNnd = \tilde{O}(k^2 Nn)$ torus elements.

We stress that the bootstrapping key $\overline{\mathsf{BK}}$ depends only on the public information (public/bootstrapping key) of involved parties and so it can be used repeatedly for any computations related to the same set of parties. In addition, the bootstrapping key can be reused when a new party joins: we only need to compute one additional row from the previously generated bootstrapping keys $\overline{\mathbf{C}}_{i,\ell}$ instead of generating them from the beginning again.

The accumulator (2-3) recursively evaluates the external product between extended RLWE and RGSW ciphertexts $k \cdot n$ times. As explained in the previous section, a single external product requires $O(kd)$ polynomial operations. Hence the total complexity is $O(k^2 nN \log N d) = \tilde{O}(k^2 nN)$ scalar operations.

Finally, $\overline{\mathsf{KS}}$ is the concatenation of k key-switching keys and the multi-key-switching algorithm has k times the complexity of the ordinary key-switching algorithm. The time and space complexity of our MKHE scheme is summarized in Table 2.

4.2 Discussion

In the previous section, we introduced an MKHE scheme which supports the evaluation of a NAND gate. However, we can simply design some variants of this basic scheme with better functionality and versatility.

More bootstrapped gates. In this paper, we described only the Multi-Key bootstrapped NAND gate, but any arbitrary binary bootstrapped gate (such as AND, OR, XOR, etc.) can be evaluated in the same way, as it is done in TFHE: it is sufficient to modify the initial linear combination before bootstrapping.

Time-Space Tradeoff. We stress again that the size of an expanded bootstrapping key $\overline{\mathsf{BK}}$ grows quadradically with the number of parties, which is larger than the linear size of a simple concatenation of individual bootstrapping keys $\{\mathsf{BK}_j\}_{j\in[k]}$. Brakerski and Perlman [5] suggested a method to reduce down the memory requirement by generating a temporary evaluation key in each step. This idea can be applied to our scheme to have a linearly-growing space complexity, however, we lose the reusability of a expanded bootstrapping key and the total complexity grows.

Threshold Decryption. HE has some attractive applications in the construction of advanced cryptographic primitives such as round-efficient MPC [15, 1, 21, 30, 27]. In particular, the distributed property of threshold HE [23, 2] makes an important role to achieve this functionality.

Since our MKHE scheme is based on the standard LWE encryption, the techniques for threshold decryption such as *noise smudging* (a.k.a. noise flooding) [1] can be directly applied to our scheme as follows. The noise distribution, parametrized by a constant $\gamma > 0$, should have a *medium size* which is smaller than 1 but sufficiently larger than the error of an input ciphertext to prevent the leakage of extra information beyond the decrypted value. See [1] for parameter choice and security proof.

$\underline{\texttt{MKHE.PartDec}(\overline{\texttt{ct}}, \mathbf{s}_i)}$: For a ciphertext $\overline{\texttt{ct}} = (b, \mathbf{a}_1, \ldots, \mathbf{a}_k) \in \mathbb{T}^{kn+1}$ and the i-th secret $\mathbf{s}_i$, sample an error $e_i \leftarrow D_\gamma$ and return the value $p_i = \langle \mathbf{a}_i, \mathbf{s}_i \rangle + e_i \pmod 1$.

$\underline{\texttt{MKHE.Merge}(b, \{p_j\}_{j\in[k]})}$: For the first entry b of an input ciphertext and the partial decryptions $\{p_j\}_{j\in[k]}$, output the bit $m \in \{0,1\}$ which minimizes $|b + \sum_{j=1}^{k} p_j - \frac{1}{4}m|$.

Faster Evaluation of a Look-Up Table (LUT). There have been some progresses in TFHE-type schemes to accelerate the evaluation of a LUT. For example, Chillotti et al. [11] suggested a *vertical* packing method for TRLWE combined with a circuit bootstrapping algorithm which gives a speed-up compared to the gate-by-gate bootstrapping, while Bonnoron et al. [3] (see also [6]) suggested a method to encrypt more than one bit in a single ciphertext. It is easy to see that these techniques are directly applicable to our MKHE scheme.

5 Experimental Results

We present a proof-of-concept implementation to convince the reader that our scheme is practical. The implementation took a few days of coding and it is based on the TFHE library [12], which takes 13ms to execute a gate bootstrapping. All experiments are performed on a Intel Core i7-4910MQ at 2.90GHz laptop, running on a single thread. Our source code is publicly available at `https://github.com/ilachill/MK-TFHE`.

In Table 3, we present three candidate parameter sets which guarantee both the security and correctness. We set the LWE/RLWE secret distributions χ and ψ as the uniform distributions over the set of binary vectors in $\mathbb{Z}^n$ and over the polynomials in R with binary coefficients, respectively. We adapt the same LWE

Set	LWE				RLWE (RGSW)			
	n	α	B_{ks}	d_{ks}	N	β	B	d
I	500	$2.43 \cdot 10^{-5}$	2^2	8	1024	$3.29 \cdot 10^{-10}$	2^7	4
II							2^6	5
III							2^4	8

Table 3. Recommended parameter sets.

and RLWE parameters as in the TFHE implementation which achieve 152-bit security level based on the security analysis in [13]. We show in Appendix A that the standard deviation of bootstrapping error grows linearly on the number of parties. Hence the growth of parameter with respect to the maximal number of involved parties is very slow. We control the noise by changing the decomposition degree and exponent which do not affect the security level.

We adapt a space-time trade-off technique in [17, 10] which reduces the complexity of key-switching procedure by publishing all LWE encryptions of $a \cdot B_{ks}^i \cdot t_j$ for $i \in [d_{ks}]$, $j \in [N]$, and $a \in \mathbb{Z}_{B_{ks}}$, compared to the encryptions of $B_{ks}^i \cdot t_j$ in the scheme description. Hence our implementation of multi-key-switching is purely represented by a summation of LWE vectors. It does not make any change in asymptotic complexity.

Set	KG	BK	KS	ct	k	$\overline{\mathsf{BK}}$ gen	$\overline{\mathsf{BK}}$	$\overline{\mathsf{KS}}$	$\overline{\mathsf{ct}}$	NAND
I	1.5s	46.9MB	62.6MB	2.0KB	2	0.4s	93.8MB	125.3MB	3.9KB	0.15s
II	1.6s	58.6MB	62.6MB	2.0KB	2	0.6s	117.2MB	125.3MB	3.9KB	0.19s
					4	3.0s	390.6MB	250.5MB	7.8KB	0.72s
III	2.2s	93.8MB	62.6MB	2.0KB	2	1.2s	187.5MB	125.3MB	3.9KB	0.30s
					4	6.9s	625.0MB	250.5MB	7.8KB	1.27s
					8	35.8s	2.20GB	501.0MB	15.6KB	5.15s

Table 4. Performance of our implementation.

Our experimental results are summarized in Table 4. On the left side of the table, we describe the *local* complexity of our scheme such as key generation timing of each party. This part is independent from k. The rest of the table presents the *global* performance of our scheme corresponding to the multi-key operation. The parameter sets I, II and III support homomorphic computation for any number of parties up to 2, 4 and 8, respectively. A smaller parameter has a better peformance but a larger parameter makes the scheme more flexible. For example, III takes twice as long compared to I in the two-party case, but it allows more parties to join the computation dynamically. We observe that the size of ciphertexts and bootstrapping timing grow linearly and quadratically with k, as expected.

References

1. G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 483–501. Springer, 2012.
2. D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. Rasmussen, and A. Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *Annual International Cryptology Conference*, pages 565–596. Springer, 2018.
3. G. Bonnoron, L. Ducas, and M. Fillinger. Large fhe gates from tensored homomorphic accumulator. In *International Conference on Cryptology in Africa*, pages 217–251. Springer, 2018.
4. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proc. of ITCS*, pages 309–325. ACM, 2012.
5. Z. Brakerski and R. Perlman. Lattice-based fully dynamic multi-key fhe with short ciphertexts. In *Annual Cryptology Conference*, pages 190–213. Springer, 2016.
6. S. Carpov, M. Izabachène, and V. Mollimard. New techniques for multi-value homomorphic evaluation and applications. *IACR Cryptology ePrint Archive*, 2018:622, 2018.
7. H. Chen and K. Han. Homomorphic lower digits removal and improved fhe bootstrapping. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 315–337. Springer, 2018.
8. L. Chen, Z. Zhang, and X. Wang. Batched multi-hop multi-key fhe from ring-lwe with compact ciphertext extension. In *Theory of Cryptography Conference*, pages 597–627. Springer, 2017.
9. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–384. Springer, 2018.
10. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33. Springer, 2016.
11. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe. In *Advances in Cryptology – ASIACRYPT 2017*, pages 377–408. Springer, 2017.
12. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast fully homomorphic encryption library. https://tfhe.github.io/tfhe/, August 2016.
13. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachne. Tfhe: Fast fully homomorphic encryption over the torus. Cryptology ePrint Archive, Report 2018/421, 2018. `https://eprint.iacr.org/2018/421`.
14. M. Clear and C. McGoldrick. Multi-identity and multi-key leveled fhe from learning with errors. In *Annual Cryptology Conference*, pages 630–656. Springer, 2015.
15. R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 280–300. Springer, 2001.

16. Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs. Spooky encryption and its applications. In *Annual Cryptology Conference*, pages 93–122. Springer, 2016.
17. L. Ducas and D. Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology–EUROCRYPT 2015*, pages 617–640. Springer, 2015.
18. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
19. C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178. ACM, 2009.
20. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013*, pages 75–92. Springer, 2013.
21. S. D. Gordon, F.-H. Liu, and E. Shi. Constant-round mpc with fairness and guarantee of output delivery. In *Annual Cryptology Conference*, pages 63–82. Springer, 2015.
22. S. Halevi and V. Shoup. Bootstrapping for helib. In *Advances in Cryptology–EUROCRYPT 2015*, pages 641–670. Springer, 2015.
23. A. Jain, P. M. R. Rasmussen, and A. Sahai. Threshold fully homomorphic encryption. Cryptology ePrint Archive, Report 2017/257, 2017. `https://eprint.iacr.org/2017/257`.
24. R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In *Topics in Cryptology–CT-RSA 2011*, pages 319–339. Springer, 2011.
25. A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM, 2012.
26. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology - EUROCRYPT 2010*, pages 1–23, 2010.
27. P. Mukherjee and D. Wichs. Two round multiparty computation via multi-key fhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 735–763. Springer, 2016.
28. C. Peikert and S. Shiehian. Multi-key fhe from lwe, revisited. In *Theory of Cryptography Conference*, pages 217–238. Springer, 2016.
29. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.
30. B. Schoenmakers and M. Veeningen. Universally verifiable multiparty computation from threshold homomorphic cryptosystems. In *International Conference on Applied Cryptography and Network Security*, pages 3–22. Springer, 2015.

A Noise Estimation

For the decomposition base B and degree d, let $\epsilon^2 = 1/(12B^{2d})$ be the variance of uniform distribution over the interval $(-1/2B^d, 1/2B^d]$. We denote by $V_B = \begin{cases} \frac{1}{12}(B^2-1) & \text{if } B \text{ is odd,} \\ \frac{1}{12}(B^2+2) & \text{if } B \text{ is even;} \end{cases}$ the mean square of a uniform distribution over $\mathbb{Z} \cap$

$(-B/2, B/2]$. We similarly define ϵ_{ks}^2 and $V_{B_{ks}}$ based on the parameter B_{ks} and d_{ks} for the key-switching algorithm. We set the RGSW and LWE secret distributions χ, ψ as uniform distributions over $\{0,1\}^N$ and $\{0,1\}^n$, respectively.

The variance of a random variable e over $\mathbb{R}$ is denoted by $\mathsf{Var}(e)$. For a random variable e over $\mathbb{R}[X]/(X^N+1)$, it denotes the variance of a coefficient when all coefficients have the same variance. If $\mathbf{e}$ is a vector of random variables, $\mathsf{Var}(\mathbf{e})$ denotes the maximum of its entries' variances.

We mainly compute the variance of a noise. Our average-case analysis is based on the heuristic assumption that a noise behaves like a Gaussian distribution, which has been empirically shown in the previous work (Fig. 11, [13]).

Ciphertext Expansion. Let us suppose that

$$\begin{aligned} (z_j, \mathbf{P}_j) &\leftarrow \texttt{RGSW.KeyGen}(pp^{\texttt{RGSW}}) \text{ for } j \in [k], \\ (\mathbf{C}, \mathbf{D}, \mathbf{F}) &\leftarrow \texttt{RGSW.UniEnc}(\mu, z_i, \mathbf{P}_i) \text{ for } \mu \in R, \text{ and} \\ \overline{\mathbf{C}} &\leftarrow \texttt{RGSW.Expand}(\mathbf{C}, \mathbf{D}, \mathbf{F}, \{\mathbf{P}_j\}_{j\in[k]}). \end{aligned}$$

Then, we have

$$\begin{aligned} \mathbf{c}_0 + z_i \cdot \mathbf{c}_1 = {} & \mathbf{C}\mathbf{z}_i = \mu \cdot \mathbf{g} + \mathbf{e}_c \pmod 1, \\ \mathbf{d}_0 + z_i \cdot \mathbf{d}_1 = {} & \mathbf{D}\mathbf{z}_i = \mu z_i \cdot \mathbf{g} + r \cdot \mathbf{e}_i + \mathbf{E}_i \mathbf{z}_i \pmod 1, \end{aligned}$$

and

$$\begin{aligned} & (\mathbf{d}_0 + \mathbf{x}_j) + z_i \cdot \mathbf{y}_j + z_j \cdot \mathbf{d}_1 \\ = {} & (\mathbf{d}_0 + z_j \cdot \mathbf{d}_1) + (\mathbf{x}_j + z_i \cdot \mathbf{y}_j) \pmod 1 \\ = {} & ([\mathbf{0}|\mu \cdot \mathbf{g}] + r \cdot \mathbf{P}_i + \mathbf{E}_i)\, \mathbf{z}_j + \mathbf{M}_j \mathbf{F} \mathbf{z}_i \pmod 1 \\ = {} & \mu z_j \cdot \mathbf{g} + r \cdot (\mathbf{b}_i + z_j \cdot \mathbf{a}) + \mathbf{E}_i \mathbf{z}_j + \mathbf{M}_j (r \cdot \mathbf{g} + \mathbf{e}_f) \pmod 1 \\ = {} & \mu z_j \cdot \mathbf{g} + r \cdot (\mathbf{b}_i - \mathbf{b}_j) + r \cdot \mathbf{e}_j + \mathbf{E}_i \mathbf{z}_j + r \cdot (\mathbf{b}_j - \mathbf{b}_i + \mathbf{e}') + \mathbf{M}_j \mathbf{e}_f \pmod 1 \\ = {} & \mu z_j \cdot \mathbf{g} + (r \cdot (\mathbf{e}_j + \mathbf{e}') + \mathbf{E}_i \mathbf{z}_j + \mathbf{M}_j \mathbf{e}_f) \pmod 1, \end{aligned}$$

where $\mathbf{e}' \in \mathbb{R}^d$ is the error vector such that $\mathbf{e}'[\ell] = \langle \mathbf{g}^{-1}(\mathbf{b}_j[\ell] - \mathbf{b}_i[\ell]), \mathbf{g} \rangle - (\mathbf{b}_j[\ell] - \mathbf{b}_i[\ell])$ for $\ell \in [d]$.

A noise in the third formula has the largest variance. It is computed by

$$\mathsf{Var}(r \cdot (\mathbf{e}_j + \mathbf{e}') + \mathbf{E}_i \mathbf{z}_j + \mathbf{M}_j \mathbf{e}_f) = (N/2)\epsilon^2 + (1 + N + dNV_B)\beta^2$$

since $\mathsf{Var}(r) = 1/2$, $\mathsf{Var}(\mathbf{e}_j) = \beta^2$, $\mathsf{Var}(\mathbf{e}') = \epsilon^2$, $\mathsf{Var}(\mathbf{E}_i \mathbf{z}_j) = (1 + N/2)\beta^2$, and $\mathsf{Var}(\mathbf{M}_j \mathbf{e}_f) = dN \cdot V_B \cdot \beta^2$.

Rounding Error. In (2-2), we compute $\tilde{b} = \lfloor 2N \cdot b' \rceil$ and $\tilde{\mathbf{a}}_i = \lfloor 2N \cdot \mathbf{a}'_i \rceil$. We assume that each of the rounding errors behaves like a uniform random variable on the interval $\mathbb{R} \pmod 1 = (-0.5, 0.5]$. Therefore, the total rounding error $(\tilde{b} - \lfloor 2N \cdot b' \rceil) + \sum_{j=1}^{k} \langle \tilde{\mathbf{a}}_j - \lfloor 2N \cdot \mathbf{a}'_j \rceil, \mathbf{s}_j \rangle$ has the variance of $\frac{1}{12}(1 + kn/2)$.

External Product. We consider the external product between an (extended) RLWE and RGSW ciphertexts $\overline{\mathbf{c}}$ and $\overline{\mathbf{C}}$. The RGSW ciphertext will satisfy $\overline{\mathbf{C}}\overline{\mathbf{z}} =$

$\mu \cdot \mathbf{G}_{k+1}\overline{\mathbf{z}} + \overline{\mathbf{e}}$ for a plaintext μ and an error vector $\overline{\mathbf{e}}$. We denote by $\mathsf{VarErr}(\overline{\mathbf{C}}) = \mathsf{Var}(\overline{\mathbf{e}})$. The external product outputs an RLWE ciphertext $\overline{\mathbf{c}}'$ satisfying

$$\begin{aligned}\langle \overline{\mathbf{c}}', \overline{\mathbf{z}} \rangle &= \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}) \cdot \overline{\mathbf{C}}\overline{\mathbf{z}} \pmod 1\\ &= \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}) \cdot (\mu \cdot \mathbf{G}_{k+1}\overline{\mathbf{z}} + \overline{\mathbf{e}}) \pmod 1\\ &= \mu \cdot \langle \overline{\mathbf{c}}, \overline{\mathbf{z}} \rangle + \left(\mu \cdot \langle \overline{\mathbf{e}}', \overline{\mathbf{z}} \rangle + \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}) \cdot \overline{\mathbf{e}}\right) \pmod 1\end{aligned}$$

for the decomposition error $\overline{\mathbf{e}}' = \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}) \cdot \mathbf{G}_{k+1} - \overline{\mathbf{c}}$. Therefore, the variance of external product error $e_{ep} = \mu \cdot \langle \overline{\mathbf{e}}', \overline{\mathbf{z}} \rangle + \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}) \cdot \overline{\mathbf{e}}$ is

$$\mathsf{Var}(e_{ep}) = \mu^2 \cdot \epsilon^2(1 + kN/2) + (k+1)dN \cdot V_B \cdot \mathsf{VarErr}(\overline{\mathbf{C}})$$

since $\mathsf{Var}(\overline{\mathbf{e}}') = \epsilon^2$ and $\mathsf{Var}(\mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}})) = V_B$. Note that we do not include the error in the phase $\langle \overline{\mathbf{c}}, \overline{\mathbf{z}} \rangle$ of the input RLWE ciphertext for simpler analysis of Mux gate.

Mux Gate. Suppose that $\overline{\mathbf{c}}_0, \overline{\mathbf{c}}_1$ are RLWE ciphertexts and $\overline{\mathbf{C}}$ is an RGSW encryption of $\mu \in \{0, 1\}$ with error $\overline{\mathbf{e}}$. The mux gate is to compute $\overline{\mathbf{c}} = \overline{\mathbf{c}}_0 + \texttt{RGSW.ExtProd}(\overline{\mathbf{c}}_1 - \overline{\mathbf{c}}_0, \overline{\mathbf{C}})$ to choose $\overline{\mathbf{c}}_\mu$ homomorphically:

$$\begin{aligned}\langle \overline{\mathbf{c}}, \overline{\mathbf{z}} \rangle &= \langle \overline{\mathbf{c}}_0, \overline{\mathbf{z}} \rangle + \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}_1 - \overline{\mathbf{c}}_0) \cdot (\mu \cdot \mathbf{G}_{k+1}\overline{\mathbf{z}} + \overline{\mathbf{e}}) \pmod 1\\ &= (1-\mu) \cdot \langle \overline{\mathbf{c}}_0, \overline{\mathbf{z}} \rangle + \mu \cdot \langle \overline{\mathbf{c}}_1, \overline{\mathbf{z}} \rangle + \left(\mu \cdot \langle \overline{\mathbf{e}}', \overline{\mathbf{z}} \rangle + \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}_1 - \overline{\mathbf{c}}_0) \cdot \overline{\mathbf{e}}\right) \pmod 1,\end{aligned}$$

for the decomposition error $\overline{\mathbf{e}}' = \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}_1 - \overline{\mathbf{c}}_0) \cdot \mathbf{G}_{k+1} - (\overline{\mathbf{c}}_1 - \overline{\mathbf{c}}_0)$. The noise has the variance of $\mu^2 \cdot \epsilon^2(1 + kN/2) + (k+1)dN \cdot V_B \cdot \mathsf{VarErr}(\overline{\mathbf{C}})$, exactly the same as external product.

Accumulation. The initial RLWE ciphertext has no noise. All bootstrapping keys $\overline{\mathbf{C}}_{i,\ell}$ have the same variance of noise $\mathsf{VarErr}(\overline{\mathbf{C}}_{i,\ell}) = (N/2)\epsilon^2 + (1 + N + dNV_B)\beta^2$ from the expansion algorithm. We recursively evaluate the mux gate $k \cdot n$ times and an encrypted secret $s_{i,\ell}$ is sampled uniformly from $\{0, 1\}$. Therefore, the output of accumulator has an error of variance

$$\frac{1}{2}kn \cdot \epsilon^2(1 + kN/2) + (k+1)kdnN \cdot V_B \cdot \left((N/2)\epsilon^2 + (1 + N + dNV_B)\beta^2\right). \quad (1)$$

Multi-Key Switching. Let $\overline{\mathsf{ct}} = (b, \mathbf{a}_1, \ldots, \mathbf{a}_k)$ be an input LWE ciphertext and $\overline{\mathsf{ct}}' = (b', \mathbf{a}'_1, \ldots, \mathbf{a}'_k)$ be the output of multi-key-switching algorithm. Then, we have

$$\begin{aligned}\langle \overline{\mathsf{ct}}', (1, \overline{\mathbf{s}}) \rangle &= b + \sum_{i=1}^{k} (b'_i + \langle \mathbf{a}'_i, \mathbf{s}_i \rangle) \pmod 1\\ &= b + \sum_{i=1}^{k} \sum_{j=1}^{N} \langle \mathbf{g}_{ks}^{-1}(a_{i,j}), t_{i,j} \cdot \mathbf{g}_{ks} + \mathbf{e}_{i,j} \rangle \pmod 1\\ &= \langle \overline{\mathsf{ct}}, (1, \overline{\mathbf{t}}) \rangle + \sum_{i=1}^{k} \sum_{j=1}^{N} \left(t_{i,j} \cdot e'_{i,j} + \langle \mathbf{g}_{ks}^{-1}(a_{i,j}), \mathbf{e}_{i,j} \rangle\right) \pmod 1\end{aligned}$$

for the decomposition error $e'_{i,j} = \langle \mathbf{g}_{ks}^{-1}(a_{i,j}), \mathbf{g}\rangle - a_{i,j}$. As a result, the variance of a multi-key-switching error $e_{ks} = \sum_{i=1}^{k}\sum_{j=1}^{N}\left(t_{i,j}\cdot e'_{i,j} + \langle \mathbf{g}_{ks}^{-1}(a_{i,j}), \mathbf{e}_{i,j}\rangle\right)$ is obtained by

$$\mathsf{Var}(e_{ks}) = kN\left(\frac{1}{2}\epsilon_{ks}^2 + d_{ks}V_{B_{ks}}\alpha^2\right). \tag{2}$$

We note that this term does not include the error of input LWE ciphertext. If $\langle \mathsf{ct}', (1,\bar{\mathbf{t}})\rangle = \frac{1}{4}m + e \pmod 1$ for a bit $m \in \{0,1\}$ and an error $e \in \mathbb{R}$, then ct' will be an encryption of the same message m with error $e' = e + e_{ks}$.

Multi-Key Switching (modified). Different from the previous algorithm, the key-switching key of the i-th party consists of LWE encryptions of $a \cdot B_{ks}^{\ell}\cdot t_{i,j}$ for $1 \le j \le N$, $0 \le \ell < d_{ks}$ and $a \in \mathbb{Z}_{B_{ks}}$ encrypted under the secret $\mathbf{s}_i$. For an input LWE ciphertext $\overline{\mathsf{ct}} = (b, \mathbf{a}_1, \ldots, \mathbf{a}_k)$, the (modified) multi-key switching algorithm computes $\mathbf{g}^{-1}(a_{i,j}) = (a_{i,j,\ell})_{0\le \ell < d_{ks}}$ for each $1 \le i \le k$ and $1 \le j \le N$, and then compute the summation of LWE encryptions of $a_{i,j,\ell}\cdot B_{ks}^{\ell}\cdot t_{i,j}$ for $1 \le i \le k$, $1 \le j \le N$ and $0 \le \ell < d_{ks}$. Therefore, the output ciphertext $\overline{\mathsf{ct}}'$ satisfies that

$$\begin{aligned}
\langle \overline{\mathsf{ct}}', (1,\bar{\mathbf{s}})\rangle &= b + \sum_{i=1}^{k}\sum_{j=1}^{N}\sum_{\ell=0}^{d_{ks}-1} \mathbf{g}_{ks}^{-1}(a_{i,j})[\ell]\cdot B_{ks}^{\ell}\cdot t_{i,j} + e_{i,j,a_{i,j,\ell}} \pmod 1\\
&= b + \sum_{i=1}^{k}\sum_{j=1}^{N}(a_{i,j} + e'_{i,j})\cdot t_{i,j} + \sum_{i=1}^{k}\sum_{j=1}^{N}\sum_{\ell=0}^{d_{ks}-1} e_{i,j,a_{i,j,\ell}} \pmod 1\\
&= \langle \overline{\mathsf{ct}}, (1,\bar{\mathbf{t}})\rangle + \left(\sum_{i=1}^{k}\sum_{j=1}^{N} t_{i,j}\cdot e'_{i,j} + \sum_{i=1}^{k}\sum_{j=1}^{N}\sum_{\ell=0}^{d_{ks}-1} e_{i,j,a_{i,j,\ell}}\right) \pmod 1,
\end{aligned}$$

for the decomposition error $e'_{i,j} = \langle \mathbf{g}_{ks}^{-1}(a_{i,j}), \mathbf{g}\rangle - a_{i,j}$. As a result, the variance of a multi-key-switching error $e_{ks} = \sum_{i=1}^{k}\sum_{j=1}^{N} t_{i,j}\cdot e'_{i,j} + \sum_{i=1}^{k}\sum_{j=1}^{N}\sum_{\ell=0}^{d_{ks}-1} e_{i,j,a_{i,j,\ell}}$ is obtained by

$$\mathsf{Var}(e_{ks}) = kN\left(\frac{1}{2}\epsilon_{ks}^2 + d_{ks}\alpha^2\right), \tag{3}$$

which is smaller than that of standard key-switching error (2).

Bootstrapping. The bootstrapping noise is simply the sum of the accumulation and multi-key-switching errors so that it has the variance of (1) + (3).