Multi-Key Homomophic Encryption from TFHE

Hao Chen[1], Ilaria Chillotti[2][0000−0002−0319−4707], and Yongsoo Song[1][0000−0002−0496−9789]

[1] Microsoft Research, Redmond, USA
{haoche,yongsoo.song}@microsoft.com
[2] imec-COSIC, KU Leuven, Leuven, Belgium
ilaria.chillotti@kuleuven.be

Abstract. In this paper, we propose a Multi-Key Homomorphic Encryption (MKHE) scheme by generalizing the low-latency homomorphic encryption by Chillotti et al. (ASIACRYPT 2016). Our scheme can evaluate a binary gate on ciphertexts encrypted under different keys followed by a bootstrapping.
The biggest challenge to meeting the goal is to design a multiplication between a bootstrapping key of a single party and a multi-key RLWE ciphertext. We propose two different algorithms for this hybrid product. Our first method improves the ciphertext extension by Mukherjee and Wichs (EUROCRYPT 2016) to provide better performance. The other one is a whole new approach which has advantages in storage, complexity, and noise growth.
Compared to previous work, our construction is more efficient 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. We provide experimental results demonstrating the running time of a homomorphic NAND gate with bootstrapping. To the best of our knowledge, this is the first attempt in the literature to implement an MKHE scheme.

Keywords: Multi-key homomorphic encryption · Bootstrapping.

1 Introduction

Cryptographic primitives for secure computation have been actively studied in recent years. Homomorphic Encryption (HE) and Multi-Party Computation (MPC) are the most promising solutions with different models and performance trade-offs. HE is useful for outsourcing 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. In MPC, multiple parties can build an interactive protocol 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. [28] proposed the notion of Multi-Key Homomorphic Encryption (MKHE) which is a variant of HE supporting computation on ciphertexts encrypted under different keys. This attractive primitive can address the aforementioned issues of HE and MPC, and it has many applications such as round-efficient MPC (e.g. [18, 2, 24, 33, 30]) and spooky encryption [19]. There have been several researches (e.g. [17, 30, 7, 31, 10]) on MKHE. However, all the previous works were purely 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.

Chillotti et al. [13] proposed an HE scheme (called TFHE) based on the learning with errors (LWE) [32] assumption and its ring variant (RLWE) [29]. This HE scheme can evaluate an arbitrary binary gate on encrypted bits followed by a bootstrapping. TFHE has advantages in running time and usability compared to other HE schemes. Its bootstrapping has a low-latency and it makes simpler the task of implementing a binary circuit without background knowledge on HE (every gate of the plaintext circuit can be automatically replaced by its bootstrapped homomorphic version).

The TFHE scheme supports an operation called *external product*, which multiplies a Ring GSW (RGSW) ciphertext to an RLWE ciphertext and returns an RLWE ciphertext. A bootstrapping key consists of several RGSW encryptions, and each of them is recursively multiplied to an RLWE ciphertext to refresh it. In the multi-key case, the main difference is that we take a multi-key ciphertext as the input of bootstrapping. Hence we should be able to multiply a bootstrapping key, which is generated by a single party, to a multi-key ciphertext, which is associated with multiple parties.

We propose an RGSW-like cryptosystem and present two methods to multiply a single-key encryption to a multi-key RLWE ciphertext. The first algorithm consists of two phases: generation of a multi-key RGSW ciphertext and multi-key external product. It is similar to previous ciphertext extension method (firstly proposed by Clear and McGoldrick [17] and simplified by Mukherjee and Wichs [30]), but our scheme is simpler, lighter and faster. Our second algorithm for hybrid product is a completely different approach to achieve the same functionality. A single-key ciphertext directly acts on a multi-key RLWE ciphertext without any expensive multi-key RGSW operation. It achieves even better asymptotic complexity and less noise growth, and thereby improves the overall performance.

In summary, 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 comparison). Furthermore, our scheme is easy to implement and compatible with existing techniques for advanced functionalities such as the threshold decryption [26, 3], circuit bootstrapping [14], and plaintext packing [4, 8].

Finally, we provide a proof-of-concept implementation with concrete parameter sets. For example, it took about 0.27, 1.45 and 7.16 seconds to evaluate a bootstrapped NAND gate when the number of parties is 2, 4 and 8, respectively, on a personal computer.

Overview of our scheme. We adapt the formalization of (R)LWE over the real torus $\mathbb{T} = \mathbb{R} \pmod 1$ from Chillotti et al. [13]. We generalize TFHE to support the homomorphic computation on ciphertexts encrypted under independently generated keys. Let $R = \mathbb{Z}[X]/(X^N + 1)$ and $T = \mathbb{T}[X]/(X^N + 1)$ for a power-of-two integer N. We use a gadget vector $\mathbf{g} = (B^{-1}, \ldots, B^{-d}) \in \mathbb{Z}^d$ for some base B and degree d.

Each party (of index i) independently generates the LWE secret $\mathbf{s}_i \in \{0,1\}^n$ and the RLWE secret $z_i \in R$. A multi-key encryption of $m \in \{0,1\}$ is 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 $\mathbf{s}_i = (s_{i,j})_{1 \le j \le n}$ are their LWE secrets. The homomorphic evaluation of a NAND gate consists in an initial linear combination followed by a bootstrapping that takes care of the non-linear part of the gate together with the noise reduction. In particular, the noise reduction is performed by homomorphically computing the decryption formula (on the exponent of X) and by selecting the correct output of the bootstrapping encoded in a fixed test polynomial. The following three steps describe in more detail the NAND evaluation idea: for more details on the original TFHE bootstrapping we refer to [15].

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 $b' + \sum_{i=1}^{k} \langle \mathbf{a}'_i, \mathbf{s}_i \rangle \approx \frac{1}{2}m \pmod 1$. 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 $\overline{\mathbf{c}} = (-\frac{1}{8}X^{\tilde{b}} \cdot h(X), \mathbf{0}) \in T^{k+1}$ for the testing polynomial $h(X) = \sum_{-N/2<d<N/2} X^d$ which is a multi-key RLWE encryption with respect to the concatenated RLWE secret $\overline{\mathbf{z}} = (1, z_1, \ldots, z_k) \in R^{k+1}$. Then, we evaluate Mux gates (data selector) recursively to obtain an RLWE encryption of $-\frac{1}{8}X^{\tilde{b} + \sum_{i=1}^{k} \langle \tilde{\mathbf{a}}_i, \mathbf{s}_i \rangle} \cdot h(X)$ using encryptions of $s_{i,j} \in \{0,1\}$.

Finally, from the output of accumulator, we extract an LWE encryption $\overline{\mathsf{ct}}^* = (b^*, \mathbf{a}^*_1, \ldots, \mathbf{a}^*_k)$ such that $b^* + \sum_{i=1}^{k} \langle \mathbf{a}^*_i, \mathbf{z}^*_i \rangle \approx \frac{1}{4}m \pmod 1$ where $\mathbf{z}^*_i \in \mathbb{Z}^N$ is a permuted coefficient vector of z_i. Finally, we perform the multi-key-switching procedure from $(\mathbf{z}^*_1, \ldots, \mathbf{z}^*_k)$ to $(\mathbf{s}_1, \ldots, \mathbf{s}_k)$ by repeating the ordinary key-switching procedure from $\mathbf{z}^*_i$ to $\mathbf{s}_i$.

The main difference between TFHE and our multi-key variant is in the second step. In the multi-key case, the i-th bootstrapping key (encryptions of $s_{i,j}$ for $1 \le j \le n$) is generated by a single party but we should multiply it to a multi-key RLWE ciphertext. We propose an RLWE-based scheme (called uni-encryption) which supports this hybrid product. In the key generation phase, each party takes a Common Reference String (CRS) $\mathbf{a} \in T^d$ and set a public key $\mathbf{b}_i \approx -z_i \cdot \mathbf{a} \pmod 1$. A party i can uni-encrypt a plaintext $\mu_i \in R$ into a ciphertext $(\mathbf{d}_i, \mathbf{F}_i = [\mathbf{f}_{i,0} | \mathbf{f}_{i,1}]) \in T^d \times T^{d \times 2}$ such that $\mathbf{d}_i \approx r_i \cdot \mathbf{a} + \mu_i \cdot \mathbf{g} \pmod 1$ and $\mathbf{f}_0 + z_i \cdot \mathbf{f}_1 \approx r_i \cdot \mathbf{g} \pmod 1$. Our hybrid product function multiplies a uni-encryption of μ_i to a multi-key RLWE encryption $\overline{\mathbf{c}} \in T^{k+1}$ and returns a multi-key RLWE ciphertext, i.e., the output $\overline{\mathbf{c}}' \in T^{k+1}$ satisfies that $\langle \overline{\mathbf{c}}', \overline{\mathbf{z}} \rangle \approx \mu_i \cdot \langle \overline{\mathbf{c}}, \overline{\mathbf{z}} \rangle \pmod 1$. We propose two different algorithms to achieve this functionality.

Our first hybrid product algorithm is an improvement of the GSW extension algorithm in previous work [17, 30, 7]. It aims to transform a uni-encryption

Scheme	Space		Time		Bootstrap
	Type	Complexity	Type	Complexity	
CZW17 [10]	EvalKey Ciphertext	$\tilde{O}(k^3n)$ $\tilde{O}(kn)$	EvalKey Gen Hom Mult	$\tilde{O}(k^3n)$ $\tilde{O}(k^3n)$	No
PS16 #2 [31]	PK Ciphertext	$\tilde{O}(kn^4)$ $\tilde{O}(k^2n^2)$	Hom Mult	$\tilde{O}(k^{2.37}n^{2.37})$	No
BP16 [7]	PK Ciphertext	$\tilde{O}(kn^3)$ $\tilde{O}(kn)$	Hom NAND	$\mathrm{poly}(k, n)$	Yes
This work (Method 1)	Eval Key Ciphertext	$\tilde{O}(k^2n^2)$ $\tilde{O}(kn)$	Eval Key Gen Hom NAND	$\tilde{O}(k^2n^2)$ $\tilde{O}(k^2n^2)$	Yes
This work (Method 2)	Ciphertext	$\tilde{O}(kn)$	Hom NAND	$\tilde{O}(k^2n^2)$	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.

of μ_i into a multi-key RGSW encryption $\overline{\mathbf{D}}_i \in T^{d(k+1)\times(k+1)}$ of the same message under the concatenated key $\overline{\mathbf{z}} \in R^{k+1}$ satisfying $\overline{\mathbf{D}}_i\overline{\mathbf{z}} \approx \mu_i \cdot (\mathbf{I}_{k+1} \otimes \mathbf{g})$ in $T^{d(k+1)}$. Then, we can perform the multi-key external product between $\overline{\mathbf{c}}$ and $\overline{\mathbf{D}}_i$ to multiply them. Compared to previous algorithm, we reduce the dimension of ciphertexts from $2k$ down to $(k+1)$ by merging duplicated components $(1, z_1, \ldots, 1, z_k)$ into $\overline{\mathbf{z}} = (1, z_1, \ldots, z_k)$. In addition, we observe that the uni-encryption is not used for encrypting real messages, but only for generating a bootstrapping key. Hence we propose a symmetric key encryption to reduce the size of ciphertexts and complexity of extension algorithm. However, the first method does not change the asymptotic complexity $O(kd^2 \cdot N \log N)$ of extension process (see Section 3.2 for details).

We propose a new framework in our second algorithm for hybrid product. The previous GSW extension is done independently from the input multi-key RLWE ciphertext $\overline{\mathbf{c}}$. Instead, we work on $\overline{\mathbf{c}}$ directly to avoid expensive multi-key RGSW operations. There are two main advantages of this approach: its complexity $O(kd \cdot N \log N)$ is asymptotically better and the noise variance is reduced by a factor of $O(d \cdot B^2)$. For these reasons, we used the second algorithm in our implementation.

Related works. López-Alt et al. [28] firstly proposed an MKHE scheme based on the NTRU assumption. Clear and McGoldrick [17] introduced an LWE-based construction, and it was significantly simplified by Mukherjee and Wichs [30]. These schemes are *single-hop* for keys where the list of parties has to be known before the computation starts. This work was improved in concurrent researches by Peikert-Shiehian [31] and Brakerski-Perlman [7] which design multi-hop (dynamic for keys) MKHEs. Chen, Zhang and Wang [10] constructed a scheme which can encrypt a ring element compared to a single bit of prior works. Unfortunately, there have been no research with implementation results because all previous schemes were impractical.

We summarize the performance of relatively efficient MKHE schemes in Table 1. We only consider the second (main) one between two schemes described in [31]. All existing schemes except [7] and [10] use variants of the GSW scheme to encrypt plaintexts. Therefore, the size of ciphertexts grows at least quadractically on the number k of parties in the computation.

Similar to our scheme, [7] encrypts a bit in a single LWE ciphertext. However, they proposed a purely abstract bootstrapping based on the evaluation of a huge branching program of length $L = \text{poly}(k, n)$ representing the NAND gate followed by LWE decryption. A memory-complexity tradeoff was proposed to keep a linear storage requirement on k, but even the asymptotic complexity of bootstrapping is not analyzed in the paper.

The construction of a batched MKHE scheme is an orthogonal research issue. Chen et al. [10] proposed a multi-key variant of BGV [6] with a larger plaintext space. However, it is a leveled scheme so a large constant (depending on the maximum level of a circuit to be evaluated) is hidden in the $\tilde{O}(\cdot)$ notation. Moreover, the space and time complexity of homomorphic multiplication 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.

Since [7] and [10] use the GSW extension to generate evaluation (bootstrapping) keys, our improved (compact and symmetric) method can be directly applied to these schemes for better performance.

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. For a positive integer k, $[k] = \{1, 2, \ldots, k\}$ denotes the index set.

2.2 Multi-key Homomorphic Encryption

A multi-key homomorphic encryption `MKHE` consists of five PPT algorithms `Setup`, `KeyGen`, `Enc`, `Dec`, and `NAND`.

- $pp \leftarrow \texttt{MKHE.Setup}(1^\lambda)$: Given the security parameter λ, returns a public parameter pp.
- $(sk, pk) \leftarrow \texttt{MKHE.KeyGen}(pp)$: Generates its secret and public keys. We assume that each party has its own ID (index) mapped to the keys.
- $\texttt{ct} \leftarrow \texttt{MKHE.Enc}(m; pk)$: Given a bit $m \in \{0,1\}$, returns a ciphertext $\texttt{ct} \in \{0,1\}^*$. We assume that every ciphertext contains IDs of relevant parties.
- $m \leftarrow \texttt{MKHE.Dec}(\overline{\texttt{ct}}; \{sk_i\}_{i\in[k]})$: Given a ciphertext $\overline{\texttt{ct}}$, let $\{sk_i\}_{i\in[k]}$ be the sequence of secret keys of relevant parties. Decrypts the ciphertext into a bit $m \in \{0,1\}$.
- $\overline{\texttt{ct}}' \leftarrow \texttt{MKHE.NAND}(\overline{\texttt{ct}}_1, \overline{\texttt{ct}}_2, \{pk_i\}_{i\in[k]})$: Given ciphertexts $\overline{\texttt{ct}}_1$ and $\overline{\texttt{ct}}_2$, let k be the number of parties relevant to either $\overline{\texttt{ct}}_1$ or $\overline{\texttt{ct}}_2$, and $\{pk_i\}_{i\in[k]}$ be the sequence of their public keys. Evaluates the NAND gate and returns a ciphertext $\overline{\texttt{ct}}'$. The output ciphertext implicitly includes k indices of related parties.

An MKHE scheme is called secure if its encryption is semantically secure. The output $\overline{\texttt{ct}}' \leftarrow \texttt{MKHE.NAND}(\overline{\texttt{ct}}_1, \overline{\texttt{ct}}_2, \{pk_i\}_{i\in[k]})$ of homomorphic NAND should satisfy $\texttt{MKHE.Dec}(\overline{\texttt{ct}}', \{sk_i\}_{i\in[k]}) = m_1 \bar{\wedge} m_2$ with an overwhelming probability if $\overline{\texttt{ct}}_1$ and $\overline{\texttt{ct}}_2$ are encryptions of m_1 and m_2, respectively.

2.3 TLWE and TRLWE

The TFHE scheme, presented for the first time in [13], 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 $\mathbb{Z}[X]/(X^N+1)$ modulo 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 [15].

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 $(b, a) \in T^2$, where a is sampled uniformly from T and $b = a \cdot z + e \pmod 1$ for an error e. The secret key z is an integer polynomial of degree N sampled from a key distribution ψ on R and the error polynomial e is sampled from a Gaussian distribution with standard deviation β. We will set ψ as the uniform distribution on the set of polynomials of R with binary coefficients in $\{0,1\}$. 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), 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).

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.

2.4 TRGSW and External Product

For a base integer $B \geq 2$ and a degree d, we call $\mathbf{g} = (B^{-1}, \ldots, B^{-d})$ 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 $\frac{1}{2}B^{-1}$.

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[3]. For a fixed TRLWE secret $s \in R$, 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

[3] 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 [15].

samples **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 **A** and **B** encrypted with the same secret key can be defined as a list of independent external products between the cipher **A** and the lines composing the cipher **B**.

The scheme TFHE has been implemented and is publicly available at [16]. 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 [32] and RGSW [23, 20] schemes and describe some extended algorithms that will be used in our MKHE scheme.

3.1 Multi-Key-Switching on LWE Ciphertexts

We first describe the standard LWE-based 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. Set the decomposition base B' and degree d' for gadget vector $\mathbf{g}' = (B'^{-1}, \ldots, B'^{-d'})$. Return the public parameter $pp^{\texttt{LWE}} = (n, \chi, \alpha, B', d')$.

An LWE secret $\mathbf{s}$ is sampled from the distribution χ. We use the key-switching gadget vector $\mathbf{g}' = (B'^{-1}, \ldots, B'^{-d'})$. Recall that the base decomposition algorithm with respect to $\mathbf{g}'$ transforms an element $a \in \mathbb{T}$ into the d'-dimensional vector $\mathbf{g}'^{-1}(a)$ with coefficients in $\mathbb{Z}_{B'}$ which minimizes $|a - \langle \mathbf{g}'^{-1}(a), \mathbf{g}' \rangle|$.

We assume that the following LWE algorithms implicitly takes $pp^{\texttt{LWE}}$ as an input.

• $\underline{\texttt{LWE.KeyGen}()}$: Sample the LWE secret $\mathbf{s} \leftarrow \chi$.

• $\underline{\texttt{LWE.Enc}(m, \mathbf{s})}$: 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 $\mathsf{ct} = (b, \mathbf{a}) \in \mathbb{T}^{n+1}$ where $b = -\langle \mathbf{a}, \mathbf{s} \rangle + \frac{1}{4}m + e \pmod 1$.

Note that the scaling factor is 1/4, as in FHEW [20] or TFHE [13]. We described a symmetric encryption for simplicity, but this algorithm can be replaced by any LWE-style encryption schemes such as public key encryption [27]. The only requirement is that the output ciphertext should be a vector $\mathsf{ct} = (b, \mathbf{a}) \in \mathbb{T}^{n+1}$ satisfying $b + \langle \mathbf{a}, \mathbf{s} \rangle \approx \frac{1}{4}m \pmod 1$.

• $\underline{\texttt{LWE.KSGen}(\mathbf{t}, \mathbf{s})}$: Given LWE secrets $\mathbf{t} \in \mathbb{Z}^N$ and $\mathbf{s} \in \mathbb{Z}^n$, it returns the key-switching key $\mathsf{KS} = \{\mathbf{K}_j\}_{j \in [N]} \in (\mathbb{T}^{d' \times (n+1)})^N$ from $\mathbf{t}$ to $\mathbf{s}$. For each $j \in [N]$, the j-th entry is generated by sampling $\mathbf{A}_j \leftarrow U(\mathbb{T}^{d' \times n})$ and $\mathbf{e}_j \leftarrow D_\beta^{d'}$, and returning $\mathbf{K}_j = [\mathbf{b}_j | \mathbf{A}_j]$ where $\mathbf{b}_j = -\mathbf{A}_j \mathbf{s} + \mathbf{e}_j + t_j \cdot \mathbf{g}' \pmod 1$.

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 $\mathsf{KS} \leftarrow \texttt{LWE.KSGen}(\mathbf{t}, \mathbf{s})$.

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

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

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

Security. The j-th component $\mathbf{K}_j$ of a key-switching key $\mathsf{KS} = \{\mathbf{K}_j\}_{j \in [N]}$ from $\mathbf{t} \in \mathbb{Z}^N$ to $\mathbf{s} \in \mathbb{Z}^n$ is generated by adding $t_j \cdot \mathbf{g}'$ to the first column of a matrix in $\mathbb{T}^{d' \times (n+1)}$ whose rows are LWE instances under the secret $\mathbf{s}$. Therefore, $\mathsf{KS} \leftarrow \texttt{LWE.KSGen}(\mathbf{t}, \mathbf{s})$ is computationally indistinguishable from the uniform distribution over $(\mathbb{T}^{d' \times (n+1)})^N$ under the LWE assumption with parameter (n, χ, β) if $\mathbf{s}$ is sampled according to χ.

Correctness. We show that if $\mathsf{ct} = (b, \mathbf{a}_1, \dots, \mathbf{a}_k)$ is an LWE ciphertext encrypted by $\overline{\mathbf{t}} = (\mathbf{t}_1, \dots, \mathbf{t}_k)$ and $\{\mathsf{KS}_i\}_{i \in [k]}$ are key-switching keys from $\mathbf{t}_i \in \mathbb{Z}^N$ to $\mathbf{s}_i \in \mathbb{Z}^n$, then the output ciphertext encrypts the same message under the concatenated secret $\overline{\mathbf{s}} = (\mathbf{s}_1, \dots, \mathbf{s}_k)$. The correctness of this algorithm is simply shown by the following equation:

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

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

3.2 Multi-key RLWE and Hybrid Product

In this section, we present a ring-based scheme supporting two algorithms `UniEnc` and `Prod`. First, `UniEnc` is a *single-key* symmetric encryption which can encrypt a ring element. We can multiply a uni-encryption to a *multi-key* RLWE cihertext using the hybrid product algorithm `Prod`. In fact, we provide two different methods to perform this operation. We will explain their performance and (dis)advantages later.

• $\underline{\texttt{RLWE.Setup}(1^\lambda)}$: 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 for the gadget vector $\mathbf{g} = (B^{-1}, \ldots, B^{-d})$.
4. Generate a random vector $\mathbf{a} \leftarrow U(T^d)$.

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

Our RLWE-based scheme is based on the CRS model since the public parameter $pp^{\texttt{RLWE}}$ contains a CRS $\mathbf{a} \in T^d$. The parameter should be chosen appropriately so that the RLWE problem with parameter (N, ψ, α) achieves at least λ-bit security level. We assume that the following RLWE algorithms implicitly takes $pp^{\texttt{RLWE}}$ as an input.

• $\underline{\texttt{RLWE.KeyGen}()}$: Sample the secret $z \leftarrow \psi$ and set $\mathbf{z} = (1, z)$. Sample an error vector $\mathbf{e} \leftarrow D_\alpha^d$ and set the public key as $\mathbf{b} = -z \cdot \mathbf{a} + \mathbf{e} \pmod 1$. Return $(z, \mathbf{b}) \in R \times T^d$.

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

1. Sample $r \leftarrow \psi$ and an error $\mathbf{e}_1 \leftarrow D_\alpha^d$. Output the vector $\mathbf{d} = r \cdot \mathbf{a} + \mu \cdot \mathbf{g} + \mathbf{e}_1 \in T^d$.
2. Sample $\mathbf{f}_1 \leftarrow U(T^d)$ and $\mathbf{e}_2 \leftarrow D_\alpha^d$. Output the ciphertext $\mathbf{F} = [\mathbf{f}_0 | \mathbf{f}_1] \in T^{d \times 2}$ where $\mathbf{f}_0 = -z \cdot \mathbf{f}_1 + r \cdot \mathbf{g} + \mathbf{e}_2 \pmod 1$.

A uni-encryption consists of three polynomial vectors of dimension d, three-fourth the size of ordinary RGSW ciphertexts in $T^{2d \times 2}$. The first component $\mathbf{d}$ and the CRS $\mathbf{a}$ together form an encryption of μ under the randomness r. We can consider $\mathbf{F}$ as an encryption of r under the secret z. In the following, we describe two different algorithms for hybrid product.

• $\underline{\texttt{RLWE.Prod}(\overline{\mathbf{c}}, (\mathbf{d}_i, \mathbf{F}_i), \{\mathbf{b}_j\}_{j \in [k]})}$: Given a multi-key RLWE ciphertext $\overline{\mathbf{c}} \in T^{k+1}$ and the public keys of k parties associated to $\overline{\mathbf{c}}$, multiply a uni-encryption $(\mathbf{d}_i, \mathbf{F}_i)$ encrypted by the i-th party to $\overline{\mathbf{c}}$ as follows. We use the notation $z_0 = 1$ and $\mathbf{b}_0 = -\mathbf{a}$ in this algorithm (in the context of $\mathbf{b}_j \approx -z_j \cdot \mathbf{a}$).

Method 1. The first method consists of two steps. We first generate an extended RGSW ciphertext $\overline{\mathbf{D}}_i \in T^{d(k+1) \times (k+1)}$ by combining a uni-encryption $(\mathbf{d}_i, \mathbf{F}_i)$ and the set of public keys $\{\mathbf{b}_j\}_{j \in [k]}$, then multiply it to $\overline{\mathbf{c}}$ using the multi-key external product.

Step 1. Ciphertext extension. $\overline{\mathbf{D}}_i \leftarrow \texttt{RLWE.Extend}((\mathbf{d}_i, \mathbf{F}_i), \{\mathbf{b}_j\}_{j \in [k]})$: For $0 \leq j \leq k$, compute the vectors $\mathbf{x}_j, \mathbf{y}_j \in R_Q^d$ by $\mathbf{x}_j[\ell] = \langle \mathbf{g}^{-1}(\mathbf{b}_j[\ell]), \mathbf{f}_0 \rangle$

and $\mathbf{y}_j[\ell] = \langle \mathbf{g}^{-1}(\mathbf{b}_j[\ell]), \mathbf{f}_1 \rangle$ for all $\ell \in [d]$, i.e., $[\mathbf{x}_j | \mathbf{y}_j] = \mathbf{M}_j \mathbf{F}_i \in T^{d \times 2}$ where $\mathbf{M}_j \in R^{d \times d}$ is the matrix of which ℓ-th row vector is $\mathbf{g}^{-1}(\mathbf{b}_j[\ell])$.

Return the expanded ciphertext

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

Step 2. Multi-key GSW external product. $\overline{\mathbf{c}}' \leftarrow \overline{\mathbf{c}} \boxdot \overline{\mathbf{D}}_i$: For a multi-key RLWE ciphertext $\overline{\mathbf{c}} \in T^{k+1}$ and a multi-key RGSW ciphertext $\overline{\mathbf{D}}_i \in T^{d(k+1)\times(k+1)}$, return the ciphertext $\overline{\mathbf{c}}' = \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}) \cdot \overline{\mathbf{D}}_i \pmod 1$.

Method 2. Given a multi-key RLWE ciphertext $\overline{\mathbf{c}} = (c_0, c_1, \ldots, c_k) \in T^{k+1}$, we first compute the following polynomials for all $0 \le j \le k$:

$$\begin{aligned} u_j &= \langle \mathbf{g}^{-1}(c_j), \mathbf{d}_i \rangle, \\ v_j &= \langle \mathbf{g}^{-1}(c_j), \mathbf{b}_j \rangle, \\ w_{j,0} &= \langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,0} \rangle, \\ w_{j,1} &= \langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,1} \rangle. \end{aligned}$$

Then, we return the multi-key RLWE ciphertext $\overline{\mathbf{c}}' = (c_0', \ldots, c_k') \in T^{k+1}$ where $c_0' = u_0 + \sum_{j=0}^{k} w_{j,0} \pmod 1$, $c_i' = u_i + \sum_{j=0}^{k} w_{j,1} \pmod 1$, and $c_j' = u_j$ for $j \in [k] \backslash \{i\}$.

Security. We claim that the distribution

$$\begin{aligned} \mathcal{D}_0 = \{(\mathbf{a}, \mathbf{b}, \mathbf{d}, \mathbf{F}) :\ & pp^{\texttt{RLWE}} \leftarrow \texttt{RLWE.Setup}(1^\lambda), \\ & (z, \mathbf{b}) \leftarrow \texttt{RLWE.KeyGen}(), (\mathbf{d}, \mathbf{F}) \leftarrow \texttt{RLWE.UniEnc}(\mu, z)\} \end{aligned}$$

is computationally indistinguishable from the uniform distribution over $T^{d \times 5}$ for any $\mu \in R$ under the RLWE assumption with parameter (N, ψ, α). We consider the following distributions: First, we can switch $(\mathbf{b}, \mathbf{a})$ and $\mathbf{F} = [\mathbf{f}_0 | \mathbf{f}_1]$ into independent uniform distributions on $T^{d \times 2}$ using the RLWE assumption of the secret z. Hence $\mathcal{D}_0$ is computationally indistinguishable from

$$\begin{aligned} \mathcal{D}_1 = \{(\mathbf{a}, \mathbf{b}, \mathbf{d}, \mathbf{F}) :& \mathbf{a}, \mathbf{b} \leftarrow U(T^d), \mathbf{F} \leftarrow U(T^{d\times 2}), \\ & r \leftarrow \psi, \mathbf{e}_1 \leftarrow D_\alpha^d, \mathbf{d} = r \cdot \mathbf{a} + \mu \cdot \mathbf{g} + \mathbf{e}_1 \pmod 1\}. \end{aligned}$$

Then, $\mathbf{d}$ is changed to a uniform distribution using the RLWE assumption of secret r again. Therefore, $\mathcal{D}_1$ is indistinguishable from the distribution

$$\mathcal{D}_2 = \{(\mathbf{a}, \mathbf{b}, \mathbf{d}, \mathbf{F}) : \mathbf{a}, \mathbf{b}, \mathbf{d} \leftarrow U(T^d), \mathbf{F} \leftarrow U(T^{d\times 2})\}.$$

Since $\mathcal{D}_2$ is independent from μ, our RLWE scheme is semantically secure.

Correctness of method 1. Let $(z_j, \mathbf{b}_j)$ be the RGSW key of the j-th party for $j \in [k]$. Suppose that $(\mathbf{d}_i, \mathbf{F}_i = [\mathbf{f}_{i,0}|\mathbf{f}_{i,1}])$ is a uni-encryption of $\mu_i \in R$ of the i-th party, i.e., $\mathbf{d}_i \approx r_i \cdot \mathbf{a} + \mu_i \cdot \mathbf{g} \pmod 1$ and $\mathbf{f}_{i,0} + z_i \cdot \mathbf{f}_{i,1} \approx r_i \cdot \mathbf{g} \pmod 1$ for some $r_i \leftarrow \psi$.

We call $\overline{\mathbf{D}} \in T^{d(k+1)\times(k+1)}$ a multi-key RGSW encryption of $\mu \in R$ under the concatenated secret $\overline{\mathbf{z}} = (1, z_1, \ldots, z_k) \in R^{k+1}$ if $\overline{\mathbf{D}} \cdot \overline{\mathbf{z}} \approx \mu \cdot \mathbf{G}_{k+1}\overline{\mathbf{z}} \pmod 1$. We first claim that $\overline{\mathbf{D}}_i \leftarrow \texttt{RLWE.Extend}((\mathbf{d}_i, \mathbf{F}_i), \{\mathbf{b}_j\}_{j\in[k]})$ is a valid RGSW encryption of μ_i corresponding to $\overline{\mathbf{z}}$. It suffices to show that $\mathbf{x}_j + z_i \cdot \mathbf{y}_j + z_j \cdot \mathbf{d}_i \approx \mu_i z_j \cdot \mathbf{g} \pmod 1$ for all $0 \le j \le k$. We can combine the following equations to obtain the desired result:

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

The multi-key external product $\boxdot$ is a natural generalization of the ordinary external product between RLWE and RGSW ciphertexts to the multi-key setting [7, 13]. Let us suppose that $\overline{\mathbf{c}} \in T^{k+1}$ is an RLWE ciphertext and $\overline{\mathbf{D}}_i \in T^{d(k+1)\times(k+1)}$ is an RGSW encryption of μ_i with respect to the secret $\overline{\mathbf{z}} \in R^{k+1}$, i.e., $\overline{\mathbf{D}}_i\overline{\mathbf{z}} \approx \mu_i \cdot \mathbf{G}_{k+1}\overline{\mathbf{z}} \pmod 1$. Then their external product $\overline{\mathbf{c}}' = \overline{\mathbf{c}} \boxdot \overline{\mathbf{D}}_i$ satisfies that $\langle \overline{\mathbf{c}}', \overline{\mathbf{z}} \rangle = \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}) \cdot \overline{\mathbf{D}}_i \overline{\mathbf{z}} \approx \mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}}) \cdot \mu_i \mathbf{G}_{k+1}\overline{\mathbf{z}} \approx \mu_i \cdot \langle \overline{\mathbf{c}}, \overline{\mathbf{z}} \rangle \pmod 1$, as desired.

Correctness of method 2. We note that $\overline{\mathbf{c}}'$ is generated by adding $\sum_{j=0}^{k} w_{j,0}$ and $\sum_{j=0}^{k} w_{j,1}$ to the zeroth and i-th components of $(u_0, \ldots, u_k)$. Hence we have $\langle \overline{\mathbf{c}}', \overline{\mathbf{z}} \rangle = \sum_{j=0}^{k} u_j \cdot z_j + \sum_{j=0}^{k} (w_{j,0} + w_{j,1} \cdot z_i) \pmod 1$.

From the definition of u_j, v_j, $w_{j,0}$ and $w_{j,1}$, we obtain

$$\begin{aligned} \sum_{j=0}^{k} u_j \cdot z_j \approx \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(c_j), r_i \cdot \mathbf{a} + \mu_i \cdot \mathbf{g} \rangle \cdot z_j \approx \mu_i \cdot \langle \overline{\mathbf{c}}, \overline{\mathbf{z}} \rangle - r_i \cdot \sum_{j=0}^{k} v_j \pmod 1, \\ \sum_{j=0}^{k} (w_{j,0} + w_{j,1} z_i) = \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,0} + z_i \cdot \mathbf{f}_{i,1} \rangle \approx r_i \cdot \sum_{j=0}^{k} v_j \pmod 1, \end{aligned}$$

and conclude that $\langle \overline{\mathbf{c}}', \overline{\mathbf{z}} \rangle \approx \mu_i \cdot \langle \overline{\mathbf{c}}, \overline{\mathbf{z}} \rangle$, as desired.

Performance. In the first method, the `Extend` algorithm transforms a uni-encryption $(\mathbf{d}_i, \mathbf{F}_i)$ generated by the i-th party into a valid multi-key RGSW ciphertext $\overline{\mathbf{D}}_i$ encrypting the same message under the concatenated secret $\overline{\mathbf{z}} = (1, z_1, \ldots, z_k) \in R^{k+1}$. It can be viewed as a variant of RGSW extension of previous work [17, 30, 10]. However, we improve its performance by reducing the dimension of extended ciphertexts by almost half from $2k$ down to $(k+1)$. Moreover, we proposed a symmetric encryption since uni-encryption is not used for plaintext encryption but only for generating the bootstrapping keys of our MKHE scheme. Therefore, our uni-encryption, extension and thereby external product algorithms are better in terms of size, complexity, and noise growth. Our solution can be directly applied to [10] to improve its key-switching key generation in the same context. We note that `Extend` requires $2(k+1)d^2 = O(kd^2)$ polynomial multiplications to generate $\mathbf{x}_j$, $\mathbf{x}_j$ for $0 \leq j \leq k$ and we can store an extended ciphertext using $(2k+3)d = O(kd)$ polynomials due to its sparsity. In addition, the external product (Step 2) takes $(3k+1)d = O(kd)$ polynomial multiplications since $\overline{\mathbf{D}}_i$ is a sparse matrix generated by the extension algorithm.

Meanwhile, our second method updates the input multi-key RLWE ciphertext $\overline{\mathbf{c}}$ without generating any multi-key GSW ciphertext. It requires only $4(k+1)d = O(kd)$ polynomial multiplications, and also enjoys a comparative advantage in terms of noise growth. Roughly speaking, it computes the same function at the plaintext level, but uses a different circuit representation.[4] The next paragraph will explain more about it in detail.

Comparison. The first method is asymptotically slower than the second method, however, we note that the ciphertext extension depends only on $(\mathbf{d}_i, \mathbf{F}_i)$ and $\{\mathbf{b}_j\}_{j\in[k]}$. The extended ciphertext $\overline{\mathbf{D}}_i$ can be pre-computed and reused in the multiplications with different multi-key RLWE ciphertexts. In particular, Step 2 requires less number of polynomial multiplication than Method 2 (about 3/4 times) so one can take an advantage of complexity from this pre-processing.

On the other hand, we point out that the second method introduces a much smaller noise. We denote by $V_B \approx B^2/12$ the variance of a uniform distribution on $\mathbb{Z}_B$. We show in Appendix A that the first method outputs a ciphertext satisfying

$$\langle \overline{\mathbf{c}}', \overline{\mathbf{z}} \rangle = \mu_i \cdot \langle \overline{\mathbf{c}}, \overline{\mathbf{z}} \rangle + e \pmod 1$$

for some error e of variance $V_1 \approx (k+1)d^2 \cdot N^2 \cdot V_B^2 \cdot \beta^2$ while the second method holds the same equation but with different error variance $V_2 \approx \frac{1}{2}(kd+k+1) \cdot N^2 \cdot V_B \cdot \beta^2 \leq (d \cdot V_B)^{-1} \cdot V_1$.

In summary, the first method with pre-processing can have an advantage in complexity (by a factor of about 3/4) by making a trade-off between storage and complexity. Meanwhile, the second method has a smaller noise growth. In other words, one may use a smaller parameter while achieving the same level of noise. For these reasons, the second method is more practical than the first one in almost all aspects.

The controlled selector gate (called CMux in [15]) is one direct application of hybrid product. It aims to securely choose $\overline{\mathbf{c}}_\mu = (1-\mu) \cdot \overline{\mathbf{c}}_0 + \mu \cdot \overline{\mathbf{c}}_1$ between two multi-key RLWE ciphertexts $\overline{\mathbf{c}}_0$ and $\overline{\mathbf{c}}_1$ using an encrypted bit $\mu \in \{0,1\}$. The CMux gate is a core operation in the bootstrapping of our scheme.

[4] For the reader who is familiar with the GSW scheme, let us cite a similar example. For GSW ciphertexts C_i, we denote by $\boxtimes$ the multiplication between GSW ciphertexts. Both $C_1 \boxtimes (C_2 \boxtimes C_3)$ and $(C_1 \boxtimes C_2) \boxtimes C_3$ are computing the same function (product of three plaintexts) but latter one introduces a much smaller error.

• $\underline{\texttt{RLWE.CMux}(\overline{\mathbf{c}}_0, \overline{\mathbf{c}}_1, (\mathbf{d}_i, \mathbf{F}_i), \{\mathbf{b}_j\}_{j\in[k]})}$: Given multi-key ciphertexts $\overline{\mathbf{c}}_0, \overline{\mathbf{c}}_1 \in T^{k+1}$, a uni-encryption $(d_i, \mathbf{F}_i)$ (encrypting a bit $\mu_i \in \{0,1\}$) and the set of public keys $\{\mathbf{b}_j\}_{1\le j\le k}$, compute and return $\overline{\mathbf{c}}' \leftarrow \overline{\mathbf{c}}_0 + \texttt{RLWE.Prod}(\overline{\mathbf{c}}_1 - \overline{\mathbf{c}}_0, (\mathbf{d}_i, \mathbf{F}_i), \{\mathbf{b}_j\}_{j\in[k]})$.

4 Multi-key Variant of TFHE

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 [13], 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 $pp^{\texttt{LWE}} = (n, \chi, \alpha, B', d')$.
– Run $\texttt{RLWE.Setup}(1^\lambda)$ to generate the parameter $pp^{\texttt{RLWE}} = (N, \psi, \beta, B, d, \mathbf{a})$.
– Return the generated public parameters $pp^{\texttt{MKHE}} = (pp^{\texttt{LWE}}, pp^{\texttt{RLWE}})$.

We assume that all other algorithms of MKHE implicitly take $pp^{\texttt{MKHE}}$ as an input.

• $\underline{\texttt{MKHE.KeyGen}()}$: Each party i independently generates its keys as follows.

– Sample the LWE secret by $\mathbf{s}_i \leftarrow \texttt{LWE.KeyGen}()$.
– Run $(z_i, \mathbf{b}_i) \leftarrow \texttt{RLWE.KeyGen}()$ and set the public key as $\mathsf{PK}_i = \mathbf{b}_i$. We write $\mathbf{z}_i^* = (z_{i,0}, -z_{i,N-1}, \dots, -z_{i,1}) \in \mathbb{Z}^N$ for $z_i = z_{i,0} + z_{i,1}X + \cdots + z_{i,N-1}X^{N-1}$.
– Generate $(\mathbf{d}_{i,j}, \mathbf{F}_{i,j}) \leftarrow \texttt{RLWE.UniEnc}(s_{i,j}, z_i)$ for $j \in [n]$ and set the bootstrapping key as $\mathsf{BK}_i = \{(\mathbf{d}_{i,j}, \mathbf{F}_{i,j})\}_{j\in[n]}$.
– Generate the key-switching key $\mathsf{KS} \leftarrow \texttt{LWE.KSGen}(\mathbf{z}_i^*, \mathbf{s}_i)$.
– Return the secret key $\mathbf{s}_i$. Publish the triple $(\mathsf{PK}_i, \mathsf{BK}_i, \mathsf{KS}_i)$ of public, bootstrapping, and key-switching keys.

We remark that for any $a = a_0 + a_1X + \cdots + a_{N-1}X^{N-1} \in T$ and the vector of its coefficients $(a_0, \dots, a_{N-1}) \in \mathbb{T}^N$, the constant term of $a \cdot z \in T$ is equal to $\langle \mathbf{a}, \mathbf{z}^* \rangle$ 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.

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

• $\underline{\texttt{MKHE.NAND}(\overline{\mathsf{ct}}_1, \overline{\mathsf{ct}}_2, \{(\mathsf{PK}_i, \mathsf{BK}_i, \mathsf{KS}_i)\}_{i\in[k]})}$: Given two LWE ciphertexts $\overline{\mathsf{ct}}_1 \in \mathbb{T}^{k_1 n+1}$ and $\overline{\mathsf{ct}}_2 \in \mathbb{T}^{k_2 n+1}$, let k be the number of parties that are associated with either $\overline{\mathsf{ct}}_1$ or $\overline{\mathsf{ct}}_2$. For $i \in [k]$, $\mathsf{PK}_i = \mathbf{b}_i$, $\mathsf{BK}_i = \{(\mathbf{d}_{i,j}, \mathbf{F}_{i,j})\}_{j\in[n]}$ and KS_i are 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 $m = m_1 \barwedge m_2$ homomorphically on encrypted bits.

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 $\overline{\mathbf{s}} = (\mathbf{s}_1, \ldots, \mathbf{s}_k) \in \mathbb{Z}^{kn}$. 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 $\langle \overline{\mathsf{ct}}_i, (1, \mathbf{s}_{j_1}, \ldots, \mathbf{s}_{j_{k_i}}) \rangle = \langle \overline{\mathsf{ct}}'_i, (1, \overline{\mathbf{s}}) \rangle$ for $\overline{\mathbf{s}} = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$.

If $\langle \overline{\mathsf{ct}}'_i, (1, \overline{\mathbf{s}}) \rangle = \frac{1}{4} m_i + e_i \pmod 1$ for some errors $e_i \in \mathbb{R}$, then the output ciphertext satisfies that $\langle \overline{\mathsf{ct}}', (1, \overline{\mathbf{s}}) \rangle = \frac{1}{2} m + e' \pmod 1$ for $m = m_1 \barwedge m_2$ 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 [20], is to evaluate the decryption circuit of an extended LWE ciphertext using the external product of RGSW scheme for bootstrapping.

2-1. 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 $\overline{\mathbf{c}} = (-\frac{1}{8} h(X) \cdot X^{\tilde{b}}, \mathbf{0}) \in T^{k+1}$ where $h = \sum_{-\frac{N}{2} < j < \frac{N}{2}} X^j = 1 + X + \cdots + X^{\frac{N}{2}-1} - X^{\frac{N}{2}+1} - \cdots - X^{N-1}$.

2-2. Let $\tilde{\mathbf{a}}_i = (\tilde{a}_{i,j})_{j\in[n]}$ for $i \in [k]$. Compute

$$\overline{\mathbf{c}} \leftarrow \mathtt{RLWE.CMux}(\overline{\mathbf{c}}, X^{\tilde{a}_{i,j}} \cdot \overline{\mathbf{c}}, (\mathbf{d}_{i,j}, \mathbf{F}_{i,j}), \{\mathbf{b}_\ell\}_{\ell\in[k]})$$

recursively for all $i \in [k]$ and $j \in [n]$.

2-3. Return $\overline{\mathbf{c}} \leftarrow (\frac{1}{8}, \mathbf{0}) + \overline{\mathbf{c}} \pmod 1$.

The accumulator $\overline{\mathbf{c}}$ is initialized in (2-1) as the trivial RLWE encryption of $-\frac{1}{8} h(X) \cdot X^{\tilde{b}}$. The main computation is done in (2-2) using the Mux gate. In each step, it homomorphically selects one of $\overline{\mathbf{c}}$ and $X^{\tilde{a}_{i,\ell}} \cdot \overline{\mathbf{c}}$ using the encryption $(\mathbf{d}_{i,j}, \mathbf{F}_{i,j})$ of $s_{i,j} \in \{0, 1\}$. The output is a multi-key RLWE ciphertext satisfying

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

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 $\langle\overline{\mathbf{c}},\overline{\mathbf{z}}\rangle$ 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 (2-3).

We stress that we proposed two different algorithms for the underlying hybrid product algorithm of CMux.

3-1. For $\overline{\mathbf{c}}=(c_0,c_1,\ldots,c_k)\in T^{k+1}$, 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 $\overline{\mathsf{ct}}''\leftarrow \texttt{LWE.MKSwitch}(\overline{\mathsf{ct}}^*,\overline{\mathsf{KS}})$.

In the last step, we transform $\overline{\mathbf{c}}$ into an LWE ciphertext and run the multi-key-switching algorithm. As we noted above, $\langle\mathbf{a}_i^*,\mathbf{z}_i^*\rangle \pmod 1$ is equal to the constant term of $c_i\cdot z_i$ for $i\in[k]$. Hence, (3-1) returns an LWE ciphertext $\overline{\mathsf{ct}}^*$ satisfying $\langle\overline{\mathsf{ct}}^*,(1,\overline{\mathbf{z}}^*)\rangle\approx\frac{1}{4}m \pmod 1$ for $\overline{\mathbf{z}}^*=(\mathbf{z}_1^*,\ldots,\mathbf{z}_k^*)$. Finally, (3-2) switches the LWE key into $\overline{\mathbf{s}}$ so the output LWE ciphertext satisfies that $\langle\overline{\mathsf{ct}},(1,\overline{\mathbf{s}})\rangle\approx\frac{1}{4}m$ (mod 1), as desired.

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

We note that each party publishes uni-encryptions of $s_1,\ldots,s_n$ encrypted by z as well as a key-switching key from $\mathbf{z}^*=(z_0,-z_{N-1},\ldots,-z_1)$ to $\mathbf{s}$. Similar to TFHE [13] and all other bootstrappable (fully) HE schemes [22] such as [20, 25, 9, 11], our scheme requires an additional circular security assumption.

Correctness conditions. Our scheme should satisfy the following requirements 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.

Performance. The accumulation step (2-2) is the most expensive part of the whole pipeline and all other algorithms including multi-key-switching are asymptotically faster. We run the CMux algorithm $k\cdot n$ times, each of which has almost the same complexity as the hybrid product `RLWE.Prod` described in Section 3.2. The computing server can choose one of two proposed algorithms and inherit their (dis)advantages. The performance of gate bootstrapping would be $k \cdot n$ times of the chosen hybrid product algorithm.

If the first method is chosen, we can pre-compute the extended RGSW ciphertexts $\overline{\mathbf{D}}_{i,j} \leftarrow \texttt{RLWE.Extend}((\mathbf{d}_{i,j}, \mathbf{F}_{i,j}), \{\mathbf{b}_\ell\}_{\ell \in [k]})$ and set $\overline{\mathsf{BK}} := \{\overline{\mathbf{D}}_{i,j}\}_{i \in [k], j \in [n]}$ as a shared bootstrapping key which can be reused in the evaluation of an arbitrary Boolean gate on the same set of k parties. However, it requires more space ($O(k^2nd)$ polynomials) to store $\overline{\mathsf{BK}}$.

4.2 Discussion

We presented a multi-key variant of the TFHE scheme. However, we can simply design some variants of this basic scheme with better functionality and versatility.

More bootstrapped gates. We described only the Multi-Key bootstrapped NAND gate in the previous section, 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 trade-off. Brakerski and Perlman [7] suggested a method to reduce down the memory requirement by generating a temporary evaluation key in each step. Since our first method generates an expanded bootstrapping key $\overline{\mathsf{BK}}$ whose size grows quadratically with the number of parties, we can apply this idea to have a linearly-growing space complexity. In this case, we lose the reusability of a expanded bootstrapping key which is the only advantage of first method compared to the second solution. Therefore, we do *not* have any motivation to adapt this optimization technique.

Distributed decryption. HE has some attractive applications in the construction of advanced cryptographic primitives such as round-efficient MPC [18, 2, 24, 33, 30]. In particular, the distributed property of threshold HE [26, 3] makes an important role to achieve this functionality. Any secure multi-party protocol can be built between key owners to evaluate the decryption circuit, but we introduce a simple example in this paragraph.

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) [2] can be directly applied to our scheme. 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 [2] 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. [14] 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. [4] (see also [8]) 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 [16]. Our source code is publicly available at `https://github.com/ilachill/MK-TFHE`.

Set	LWE				RLWE (RGSW)			
	n	α	B'	d'	N	β	B	d
I	560	$3.05 \cdot 10^{-5}$	2^2	8	1024	$3.72 \cdot 10^{-9}$	2^9	3
II							2^8	4
III							2^6	5

Table 2. Recommended parameter sets.

In Table 2, we present three candidate parameter sets. We increase the dimensions of LWE and RLWE to have a more conservative parameter.[5] Our parameters achieve at least 110-bit security level according to the LWE Estimator [1], which is a common reference in the domain.[6]

As mentioned before, we used the second hybrid product algorithm in implementation. 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 show in Appendix A that the standard deviation of bootstrapping error grows linearly on the number of parties. Hence the growth of parameter with

[5] In [15], the authors recommend to take more conservative parameters for the original TFHE scheme as well. This new parameter set will affect their gate bootstrapping timing by making it increase of a few milliseconds with respect to the original given execution timing of about 13 ms.

[6] `https://bitbucket.org/malb/lwe-estimator/src/master/`

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 [20, 13] which reduces the complexity of key-switching procedure by publishing all LWE encryptions of $a \cdot B'^i \cdot t_j$ for $i \in [d']$, $j \in [N]$, and $a \in \mathbb{Z}_{B'}$, compared to the encryptions of $B'^i \cdot t_j$ 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	$\overline{\mathsf{ct}}$	NAND
I	1.1 s	0.62 MB	70.1 MB	2.19 KB	2	4.38 KB	0.27 s
II	1.2 s	0.82 MB	70.1 MB	2.19 KB	2	4.38 KB	0.43 s
					4	8.77 KB	1.45 s
III	1.3 s	1.03 MB	70.1 MB	2.19 KB	2	4.38 KB	0.50 s
					4	8.77 KB	1.90 s
					8	17.32 KB	7.16 s

Table 3. Performance of our implementation. k denotes the number of parties in computation.

Our experimental results are summarized in Table 3. All experiments are performed on a Intel Core i7-4910MQ at 2.90GHz laptop, running on a single thread, which takes 13ms to execute a gate bootstrapping of the TFHE library. On the left sides of table, we describe the *local* complexity of our scheme such as key generation timing of each party. This part is independent from k. The other side presents the *global* performance corresponding to the multi-key operation. The parameter sets I, II and III support homomorphic computation on any number of parties up to 2, 4 and 8, respectively. A smaller parameter has a better performance but a larger one makes the scheme more flexible because more parties can join the computation dynamically. We believe that the code has space for optimization. This, with a more accurate choice of the parameters could produce better execution timings.

6 Conclusion

We designed a practical MKHE scheme by generalizing the gate bootstrapping of TFHE to the multi-key case. Our main technical contribution is to establish a new hybrid product between single-key and multi-key ciphertexts which provides better storage, computational cost and noise growth. We implemented our scheme to present its concrete performance.

As we discussed in Section 4.2, one future direction is to implement advanced functionalities of TFHE in the multi-key setting. Another direction is to design a practical MKHE scheme from another HE system (e.g. BFV [5, 21], CKKS [12]) which has advantages in amortized complexity. Finally, one primary open problem in this area is how to construct an MKHE scheme without the CRS model.

7 Acknowledgments

The second author (I.C.) has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT and by the FWO under an Odysseus project GOH9718N. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ERC or FWO.

References

1. M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.
2. 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.
3. 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.
4. 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.
5. Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.
6. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proc. of ITCS*, pages 309–325. ACM, 2012.
7. Z. Brakerski and R. Perlman. Lattice-based fully dynamic multi-key fhe with short ciphertexts. In *Annual Cryptology Conference*, pages 190–213. Springer, 2016.
8. S. Carpov, M. Izabachène, and V. Mollimard. New techniques for multi-value homomorphic evaluation and applications. *IACR Cryptology ePrint Archive*, 2018:622, 2018.
9. 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.
10. 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.
11. 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.

12. J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.
13. 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.
14. 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.
15. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, Apr 2019.
16. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. TFHE: Fast fully homomorphic encryption library. https://tfhe.github.io/tfhe/, August 2016.
17. 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.
18. 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.
19. Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs. Spooky encryption and its applications. In *Annual Cryptology Conference*, pages 93–122. Springer, 2016.
20. 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.
21. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
22. 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.
23. 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.
24. 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.
25. S. Halevi and V. Shoup. Bootstrapping for helib. In *Advances in Cryptology–EUROCRYPT 2015*, pages 641–670. Springer, 2015.
26. 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`.
27. 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.
28. 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.
29. 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.
30. 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.

31. C. Peikert and S. Shiehian. Multi-key fhe from lwe, revisited. In *Theory of Cryptography Conference*, pages 217–238. Springer, 2016.
32. 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.
33. 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 $(-\frac{1}{2}B^{-d}, \frac{1}{2}B^{-d}]$. 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'^2 and $V_{B'}$ based on the parameter B' and d' 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. 10, [15]).

Hybrid product method 1.

Step 1: Ciphertext extension. Let us suppose that

$$\begin{aligned} &\mathbf{b}_j = -z_j \cdot \mathbf{a} + \mathbf{e}_j \pmod 1 \text{ for } j \in [k], \\ &\mathbf{d}_i = r_i \cdot \mathbf{a} + \mu_i \cdot \mathbf{g} + \mathbf{e}_{i,1} \pmod 1, \\ &\mathbf{f}_{i,0} + z_i \cdot \mathbf{f}_{i,1} = r_i \cdot \mathbf{g} + \mathbf{e}_{i,2} \pmod 1 \text{ for some } \mu \in R \text{ and} \\ &\overline{\mathbf{D}}_i \leftarrow \texttt{RLWE.Extend}\left((\mathbf{d}_i, \mathbf{F}_i), \{\mathbf{b}_j\}_{j\in[k]}\right). \end{aligned}$$

We let $\mathbf{e}_0 = \mathbf{0}$ for simplicity. Then for any $0 \le j \le k$, the j-th row of $\overline{\mathbf{D}}_i$ satisfies that

$$\mathbf{x}_j + z_i \cdot \mathbf{y}_j = \mathbf{M}_j \cdot (r_i \cdot \mathbf{g} + \mathbf{e}_{i,2}) = r_i \cdot \mathbf{b}_j + (r_i \cdot \mathbf{e}'_j + \mathbf{M}_j \cdot \mathbf{e}_{i,2}) \pmod 1$$

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

$$\begin{aligned} z_j \cdot \mathbf{d}_i = &\; r_i z_j \cdot \mathbf{a} + \mu_i z_j \cdot \mathbf{g} + z_j \cdot \mathbf{e}_{i,1} \\ = &\; -r_i \cdot \mathbf{b}_j + \mu_i z_j \cdot \mathbf{g} + (r_i \cdot \mathbf{e}_j + z_j \cdot \mathbf{e}_{i,1}) \pmod 1. \end{aligned}$$

Therefore, the j-th row is decrypted into $\mu_i z_j \cdot \mathbf{g} + \mathbf{e}_{i,j}$ for the GSW extension error $\mathbf{e}_{i,j} = r_i \cdot (\mathbf{e}_j + \mathbf{e}'_j) + z_j \cdot \mathbf{e}_{i,1} + \mathbf{M}_j \cdot \mathbf{e}_{i,2}$.

Its variance is bounded by

$$V_{exp} \le (N/2)\epsilon^2 + (1 + d \cdot V_B) \cdot N\beta^2$$

since $\mathsf{Var}(r_i) = \mathsf{Var}(z_j) = 1/2$, $\mathsf{Var}(\mathbf{e}_j) \le \mathsf{Var}(\mathbf{e}_{i,1}) = \mathsf{Var}(\mathbf{e}_{i,2}) = \beta^2$, $\mathsf{Var}(\mathbf{e}'_j) = \epsilon^2$ and $\mathsf{Var}(\mathbf{M}_j \cdot \mathbf{e}_{i,2}) = dN \cdot V_B \cdot \beta^2$.

Step 2: Multi-key GSW external product. Let $\overline{\mathbf{c}}$ and $\overline{\mathbf{D}}$ be multi-key RLWE and RGSW ciphertexts. Suppose that $\overline{\mathbf{D}}$ satisfies $\overline{\mathbf{D}}\overline{\mathbf{z}} = \mu \cdot \mathbf{G}_{k+1}\overline{\mathbf{z}} + \overline{\mathbf{e}}$ for a plaintext $\mu \in R$ and an error vector $\overline{\mathbf{e}}$. We denote by $\mathsf{VarErr}(\overline{\mathbf{D}}) = \mathsf{Var}(\overline{\mathbf{e}})$. The external product outputs an RLWE ciphertext $\overline{\mathbf{c}}'$ satisfying

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

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

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

since $\mathsf{Var}(\overline{\mathbf{e}}') = \epsilon^2$ and $\mathsf{Var}(\mathbf{G}_{k+1}^{-1}(\overline{\mathbf{c}})) = V_B$.

In our case, $\overline{\mathbf{D}} = \overline{\mathbf{D}}_i$ is an extended RGSW ciphertext whose error variance is $V_{exp} \le (N/2)\epsilon^2 + (1 + d \cdot V_B) \cdot N\beta^2$. As a result, our first method returns a ciphertext whose noise variance is

$$V_1 = \mu_i^2 \cdot \epsilon^2 (1 + kN/2) + (k+1)dN \cdot V_B \cdot V_{exp}.$$

In our MKHE scheme, the decomposition error ϵ^2 can be easily controlled. Hence the extension error is mainly dominated by $V_{exp} \approx dN \cdot V_B \cdot \beta^2$. Similarly, the noise of hybrid product is dominated by $V_1 \approx (k+1)dN \cdot V_B \cdot V_{exp} \approx (k+1)d^2 \cdot N^2 \cdot V_B^2 \cdot \beta^2$.

Hybrid product method 2. As shown earlier, the output $\overline{\mathbf{c}}'$ of the second multiplication algorithm satisfies $\langle \overline{\mathbf{c}}', \overline{\mathbf{z}} \rangle = \sum_{j=0}^{k} u_j \cdot z_j + \sum_{j=0}^{k} (w_{j,0} + w_{j,1} \cdot z_i)$. The first term is

$$\sum_{j=0}^{k} u_j \cdot z_j = \sum_{j=0}^{k} \big\langle \mathbf{g}^{-1}(c_j), r_i \cdot \mathbf{a} + \mu_i \cdot \mathbf{g} + \mathbf{e}_{i,1} \big\rangle \cdot z_j \pmod 1$$

$$= \mu_i \cdot \langle \overline{\mathbf{c}}, \overline{\mathbf{z}} \rangle + \mu_i \cdot e' + r_i \cdot \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(c_j), z_j \cdot \mathbf{a} \rangle + \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(c_j), \mathbf{e}_{i,1} \rangle \cdot z_j \pmod 1$$

$$= \mu_i \cdot \langle \overline{\mathbf{c}}, \overline{\mathbf{z}} \rangle - r_i \cdot \sum_{j=0}^{k} v_j + \mu_i \cdot e' + r_i \cdot \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(c_j), \mathbf{e}_j \rangle + \left\langle \sum_{j=0}^{k} z_j \cdot \mathbf{g}^{-1}(c_j), \mathbf{e}_{i,1} \right\rangle$$

for the decomposition error $e' = \sum_{j=0}^{k} \big(\langle \mathbf{g}^{-1}(c_j), \mathbf{g} \rangle - c_j \big) \cdot z_j$, while the second term is

$$\sum_{j=0}^{k} (w_{j,0} + w_{j,1} z_i) = \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), \mathbf{f}_{i,0} + z_i \cdot \mathbf{f}_{i,1} \rangle \pmod 1$$
$$= \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(v_j), r_i \cdot \mathbf{g} + \mathbf{e}_{i,2} \rangle = r_i \cdot \sum_{j=0}^{k} v_j + r_i \cdot e'' + \left\langle \sum_{j=0}^{k} \mathbf{g}^{-1}(v_j), \mathbf{e}_{i,2} \right\rangle \pmod 1$$

for $e'' = \sum_{j=0}^{k} \big(\langle \mathbf{g}^{-1}(v_j), \mathbf{g} \rangle - v_j \big)$. Note that $\mathsf{Var}(e') = \epsilon^2 (1 + kN/2)$ and $\mathsf{Var}(e'') = \epsilon^2 (k+1)$.

Therefore, the noise of $\overline{\mathbf{c}}'$ is

$$\mu_i \cdot e' + r_i \cdot \sum_{j=0}^{k} \langle \mathbf{g}^{-1}(c_j), \mathbf{e}_j \rangle + \left\langle \sum_{j=0}^{k} z_j \cdot \mathbf{g}^{-1}(c_j), \mathbf{e}_{i,1} \right\rangle + r_i \cdot e'' + \left\langle \sum_{j=0}^{k} \mathbf{g}^{-1}(v_j), \mathbf{e}_{i,2} \right\rangle,$$

and its variance

$$\begin{aligned} V_2 = \mu_i^2 N \epsilon^2 (1 + kN/2) + (N^2/2)(k+1) V_B \beta^2 + dN(1 + kN/2) V_B \beta^2 \\ + (N/2)\epsilon^2 (k+1) + (k+1) N V_B \beta^2, \end{aligned}$$

is dominated by $V_2 \approx \frac{1}{2}(kd + k + 1) \cdot N^2 \cdot V_B \cdot \beta^2$.

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 $\frac{1}{12}(1 + kn/2)$.

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 $\overline{\mathbf{c}} = \overline{\mathbf{c}}_0 + \texttt{RLWE.Prod}(\overline{\mathbf{c}}_1 - \overline{\mathbf{c}}_0, \overline{\mathbf{C}})$ to choose $\overline{\mathbf{c}}_\mu$ homomorphically:

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

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

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

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

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

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

$$\mathsf{Var}(e_{ks}) = kN \left(\frac{1}{2} \epsilon'^2 + d' \cdot V_{B'} \cdot \alpha^2 \right). \tag{2}$$

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 $a \cdot B'^{\ell} \cdot t_{i,j}$ for $1 \le j \le N$, $0 \le \ell < d'$ and $a \in \mathbb{Z}_{B'}$ 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 $\mathbf{g}'^{-1}(a_{i,j}) = (a_{i,j,\ell})_{0 \le \ell < d'}$ for each $1 \le i \le k$ and $1 \le j \le N$, and then compute the summation of LWE encryptions of $a_{i,j,\ell} \cdot B'^{\ell} \cdot t_{i,j}$ for $1 \le i \le k$, $1 \le j \le N$ and $0 \le \ell < d'$. Therefore, the output ciphertext $\overline{\mathsf{ct}}'$ satisfies that

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

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

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

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).