Multi-Key Homomorphic Encryption with Threshold Re-Encryption

Akira Nakashima[1], Yukimasa Sugizaki[1], Hikaru Tsuchida**[2], Takuya Hayashi[1], Koji Nuida[3], Kengo Mori[1], and Toshiyuki Isshiki[1]

[1]NEC Corporation, Kanagawa, Japan
{akira-nakashima,yukimasa-sugizaki,takuya-hayashi,
ke-mori.bx,toshiyuki-isshiki}@nec.com
[2]Saitama Institute of Technology, Saitama, Japan
h_tsuchida@sit.ac.jp
[3]Kyushu University, Fukuoka, Japan
nuida@imi.kyushu-u.ac.jp

Abstract. Fully homomorphic encryption (FHE) is a cryptographic scheme that allows users to perform arbitrary arithmetic operations over plaintexts by operations (called homomorphic operations) on ciphertexts without decryption. A multi-key FHE (MK-FHE) can perform homomorphic operations on ciphertexts encrypted with different encryption keys.

In MK-FHE schemes, to decrypt a ciphertext encrypted with different users' keys, users having the corresponding decryption keys run a threshold decryption, which is a combination of each user's partial decryption and merging of their results. However, it has a drawback that the merging process requires communication and hence these users must be online during the process. Moreover, the computation and communication costs grow when the number of involved users increases. There is a previous work to overcome this issue by applying the idea of proxy re-encryption (PRE), where a proxy can convert a multi-key ciphertext, using re-encryption keys given by the key holders, into a ciphertext decryptable by a single receiver's decryption key. However, a collusion of only an adversarial receiver and the single proxy can reveal the original user's decryption key.

To resolve the issue, we propose a new framework of MK-FHE with threshold PRE. Here we introduce N proxies performing re-encryption in threshold manner; now the adversarial receiver needs to collude with all of the N proxies, which becomes more difficult than the previous single-proxy case. We also propose an instantiation based on the BFV scheme and prove its security. In addition, we implement our scheme and measure the running time of its algorithms.

Keywords: Privacy-enhancing technologies · Multi-key homomorphic encryption · Threshold proxy re-encryption · Secret sharing scheme.

** This work was done when this author worked at NEC Corporation.

1 Introduction

1.1 Backgrounds

Fully homomorphic encryption (FHE), one of the privacy-enhancing technologies [21], is a cryptographic scheme that allows users to perform arbitrary arithmetic operations over plaintexts by operations on ciphertexts without decryption. Such operations on ciphertexts are called homomorphic operations. FHE has attracted worldwide attention, and various communities have carried out standardization activities in recent years [15, 22, 36]. Mature schemes that are sometimes mentioned as candidates for standardization include BGV [6], BFV [18], CGGI [14], and CKKS [13].

Most existing FHE schemes can only perform homomorphic operations on ciphertexts encrypted with the same encryption key. Such an FHE scheme is called single-key FHE (SK-FHE). An FHE that can perform homomorphic operations on ciphertexts encrypted with different encryption keys is called multi-key FHE (MK-FHE), which is useful for secure cooperative computation between users, e.g., cross-organizational data analysis. In MK-FHE, each user encrypts its input using its encryption key and sends it to a computing server. Then, the computing server evaluates the function with the users' encrypted inputs by homomorphic operations and sends the encrypted evaluation result to the users. Note that the evaluation result is encrypted by the encryption keys involved in the encryption of the input. Hence, decrypting it requires all of the involved users' decryption keys.

To decrypt the encrypted evaluation result, the users run a threshold decryption. This procedure allows users to decrypt the ciphertext without revealing users' decryption keys by partial decryption, i.e., each user operates on a part of the ciphertext by using its decryption key and merges partial decryption results by communicating with each other. Most recently, the National Institute of Standards and Technology (NIST) has initiated standardization activities on the threshold decryption in FHE [30].

The threshold decryption requires all the owners of the decryption keys to be online, which is sometimes difficult to guarantee depending on the communication environment, each user's device, and application. It is also undesirable when a number of users are involved in the computation, in which case more computations and communications are required for threshold decryption.

To overcome the issue, Yasuda et al. proposed MK-FHE with proxy re-encryption [37]. They introduced re-encryption keys that allow delegation of decryption authority between users without revealing the decryption key. The delegator of the decryption authority creates a re-encryption key for the delegatee and sends it to a proxy. The proxy re-encrypts the evaluation results' ciphertext encrypted with the users' encryption keys, including the delegators, into ciphertext encrypted with the encryption key of the delegatee, without decrypting it. The delegatee can obtain the evaluation result by decrypting the re-encrypted evaluation result with its decryption key without running the threshold decryption.

However, in the scheme of [37], if the delegatee colludes with the proxy who has the re-encryption key, the decryption key of the delegator may be compromised. If the delegator's decryption key is compromised, it is possible for the adversarial delegatee and proxy to perform all the operations that cannot be performed without using the delegatee's decryption key, e.g., delegating the delegator's decryption authority to a third party without the delegator's permission.

1.2 Our Contributions

Table 1. Comparison of Proposed Scheme with Related Works Which Support Proxy Re-Encryption (#Corruptions: number of corrupted parties from proxies and one receiver by adversary required for delegator's decryption key compromise from re-encryption key. The symbol " - " means that an adversary cannot obtain the delegator's decryption key from the re-encryption key even if the adversary corrupts any number of parties other than the delegator.)

Schemes	Is it an MK-FHE scheme?	#Corruptions
[34, 12]	No (Not FHE scheme)	-
[26, 16]	No (SK-FHE)	Two parties from one proxy and one receiver
[37]	Yes (MK-FHE [33] based on GSW [19])	Two parties from one proxy and one receiver
MK-BFV-TRE (Ours)	Yes (MK-FHE [10] based on BFV [18])	$N + 1$ parties from N proxies and one receiver

In this paper, we propose a new framework of MK-FHE with *threshold proxy re-encryption* (MK-FHE-TRE). In contrast to MK-FHE with a single proxy in [37], now a re-encryption key is distributed to multiple proxy servers, and the re-encryption is jointly performed without reconstructing the re-encryption key.

We also propose an instantiation of MK-FHE-TRE based on the multi-key variant of BFV [18] (MK-BFV-TRE). It has the following two features. See Table 1 for a comparison of previous studies and MK-BFV-TRE.

1. MK-BFV-TRE is based on the multi-key BFV (MK-BFV) proposed by Chen et al. [10], which is a multi-key variant of BFV [18]. BFV is one of the candidates for a standardized FHE scheme. On the other hand, the scheme of Yasuda et al. [37] is based on MK-FHE proposed by Peikert and Shiehian [33], which is a multi-key variant of GSW [19]. GSW is as mature as BFV, but it has a larger ciphertext size than BFV. Also, while BFV is implemented in well-known open-source softwares of FHE, e.g., OpenFHE [4] and Microsoft

SEAL [35], GSW is not implemented in them. From these viewpoints, being based on BFV instead of GSW is expected to be practically advantageous.
2. In MK-BFV-TRE, the decryption key of the delegator is not compromised unless the delegatee colludes with all the proxies. As the number of proxies increases, the adversary's attack cost to obtain the delegator's decryption key increases. Therefore, MK-BFV-TRE can manage re-encryption keys more securely than the scheme of Yasuda et al. [37].

We also prove the security of MK-BFV-TRE under the same security assumption for MK-BFV and the underlying secret sharing scheme.

In addition, we implement MK-BFV-TRE for experiments to demonstrate a relationship between computation and communication costs. By using our implementation, we measure the sizes of keys and ciphertexts and the computation time for each algorithm of MK-BFV-TRE while varying the number of users and proxies. For processes involving communication, we also estimate the execution time by making assumptions about the communication environment while varying the number of users and proxies. From these experimental results, we believe that our decryption after our threshold re-encryption is superior to threshold decryption of [10] when the number of users is much larger than the number of proxies.

1.3 Technical Overview

We extend MK-BFV by adding proxy re-encryption to it. Next, we realize MK-BFV-TRE by distributing the re-encryption key among multiple proxies and modifying the extended MK-BFV with proxy re-encryption so that the re-encryption is performed without reconstructing the re-encryption key.

How to construct proxy re-encryption. A straightforward method of re-encryption in FHE is the *key-switching* technique, which is also employed in the scheme [37] proposed by Yasuda et al. However, the key-switching technique accumulates noise in the ciphertext, which is undesirable when complex computations are performed by homomorphic operations or when many users are involved in computations.

To avoid accumulations of noises by re-encryption via the key-switching technique, we introduce a novel proxy re-encryption scheme for MK-BFV. In our proxy re-encryption, apart from the user's decryption key, we introduce a *masking key* to decrypt the re-encrypted ciphertext. The masking key is created by the delegatee at the time of the re-encryption key creation event for each delegator. The delegator adds the delegatee's masking key to the delegator's decryption key and regards it as the re-encryption key.

Since the masking key is used to decrypt the re-encrypted ciphertext, the delegatee needs to manage the masking key in addition to the decryption key which decrypts the ciphertext before re-encryption. That is, we avoid noise accumulation in the ciphertext by the key-switching technique in exchange for an increase in the cost of key management by the delegatee.

How to distribute re-encryption key. To distribute the re-encryption key among multiple proxies, we use an additive secret sharing (ASS) over the same polynomial ring as the ciphertexts modulus of our MK-BFV-TRE. We call the distributed values among multiple proxies using secret sharing as *share*. Loosely speaking, since the decryption procedure of MK-BFV [10] is a simple inner product of keys and ciphertexts, only addition between shares and multiplication of shares and constants can re-encrypt the ciphertext without reconstructing the re-encryption key.

How to avoid adversarial collusions of proxies and a receiver. Our MK-BFV-TRE introduces N proxies and employs N-out-of-N ASS ((N, N)-ASS). Even if $N-1$ proxies collude, the re-encryption key cannot be reconstructed by $N-1$ proxies.

If the N proxies collude, the re-encryption key is reconstructed. However, since the delegator's decryption key is concealed by the added masking key, which is a random value, the reconstructed re-encryption key does not reveal the delegator's decryption key.

Hence, in our MK-BFV-TRE scheme, the delegator's decryption key is compromised only when the N proxies and one receiver (i.e. the delegatee) collude.

1.4 Related Works

MK-FHE. López-Alt et al. were the first to propose an MK-FHE [24]. Following the work of [24], various MK-FHE schemes [33, 11, 10, 9] were proposed.

Those MK-FHE schemes require the users with the decryption keys corresponding to the encryption keys involved in the computation to participate in the threshold decryption. Therefore, the key holders must always be online during decryption. In addition, as the number of users involved in the threshold decryption increases, its computation and communication times increase.

Proxy Re-Encryption (PRE). PRE [5, 3, 8] is a cryptosystem that allows a proxy to convert a ciphertext for the delegator to that for the delegatee without decrypting it. To convert the ciphertext, the proxy uses a re-encryption key provided by the delegator instead of the delegator's decryption key. One of the desirable properties of PRE is *collusion resistance*. In collusion-resistant PRE schemes [32, 17], the adversarial delegatee and proxy cannot obtain the delegator's decryption key from the re-encryption key even if they collude.

To resolve the problem of the proxy becoming a single point of failure and for secure management of re-encryption keys, threshold PRE (TPRE) schemes have been proposed [34, 12]. In TPRE schemes, re-encryption keys are distributed among multiple proxies. The multiple proxies re-encrypt ciphertexts without reconstructing re-encryption keys.

However, those PRE and TPRE schemes [5, 3, 8, 32, 12, 17, 34] can neither perform an unbounded number of homomorphic operations nor homomorphic operations between ciphertexts encrypted with different keys.

Homomorphic PRE (H-PRE). SK-FHE can only perform homomorphic operations between ciphertexts encrypted with the same encryption key. H-PRE [26, 16] uses the conversion of ciphertexts, like PRE, to resolve this issue. In

H-PRE schemes, after converting ciphertexts encrypted with different keys to one that is encrypted with the same encryption key by using re-encryption keys, the computing server can perform homomorphic operations between converted ciphertexts. However, because H-PRE requires re-encryption before homomorphic operations, the number of required re-encryption keys and the computational cost of re-encryption increase when the number of inputs and users involved are large.

2 Preliminaries

2.1 Notations

Unless otherwise stated, we denote binary logarithm by log.

For a power of two n, we let $R = \mathbb{Z}[X]/(X^n + 1)$. We also let $R_q = R/(q \cdot R)$ for an integer $q \geq 2$, and let $|R_q|$ be a bit size of an element in R_q, i.e., $|R_q| = n \cdot \lceil \mathsf{log}(q) \rceil$. We denote elements in R_q in bold, and for $\boldsymbol{x} = \sum_{i=0}^{N-1} x_i \cdot X^i$ in R_q, we let $|\boldsymbol{x}|_\infty = \mathsf{max}_i(|x_i|)$. For d-element vectors $v = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d)$ and $w = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_d)$ over R_q, we denote an inner product of two vectors as $\langle v, w \rangle = \sum_{i=1}^{d} \boldsymbol{v}_i \cdot \boldsymbol{w}_i$. We write $x \overset{\mathrm{U}}{\leftarrow} S$ to sample x uniformly from a set S, and $x \leftarrow D$ to sample x according to a distribution D.

Let k and N be the number of users and proxies, and let U_i and P_j be the ith user and the jth proxy, respectively. We then let $\mathcal{U} = \{U_1, \ldots, U_k\}$ and $\mathcal{P} = \{P_1, \ldots, P_N\}$ be the sets of the users and proxies, respectively.

Throughout this paper, we assume *semi-honest* adversaries, i.e., adversaries try to learn information from users' inputs without deviating from the protocols. We also assume that each user and proxy is connected via a point-to-point secure and synchronous communication channel.

2.2 Ring Learning with Errors (RLWE) Problem [25]

A RLWE sample is given as $(\boldsymbol{a} \cdot \boldsymbol{s} + \boldsymbol{e}, \boldsymbol{a}) \in R_q^2$ for polynomials $\boldsymbol{a} \overset{\mathrm{U}}{\leftarrow} R_q$, $\boldsymbol{s} \leftarrow \chi_{\text{key}}$, and $\boldsymbol{e} \leftarrow \chi_{\text{err}}$, where χ_{key} and χ_{err} are key and error distributions, respectively. For parameters $(n, q, \chi_{\text{key}}, \chi_{\text{err}})$ depending on a security parameter λ, the decision RLWE problem is to distinguish polynomial number of RLWE samples from uniformly random elements in R_q^2. The BFV, MK-BFV, and our MK-BFV-TRE schemes are secure under the RLWE assumption, which means the RLWE problem cannot be solved in polynomial time.

2.3 (N, N)-ASS

For an element $\boldsymbol{x} \in R_q$, we denote the set of (N, N)-ASS shares of $\boldsymbol{x}$ as $[\boldsymbol{x}] = ([\boldsymbol{x}]_1, \ldots, [\boldsymbol{x}]_N)$ where $[\boldsymbol{x}]_i \in R_q$. In MK-BFV-TRE, we use the algorithms below to generate and reconstruct the shares.

– $[\boldsymbol{x}] \leftarrow \mathsf{Share}(\boldsymbol{x}, U_i)$: Let $U_i \in \mathcal{U}$ be a user who holds $\boldsymbol{x}$. U_i generates $\boldsymbol{x}_2, \ldots, x_N \overset{\mathrm{U}}{\leftarrow} R_q$, and sets $\boldsymbol{x}_1 = \boldsymbol{x} - \sum_{j=2}^{N} \boldsymbol{x}_j$. Then, U_i sets the shares as $[\boldsymbol{x}] = ([\boldsymbol{x}]_1 = \boldsymbol{x}_1, \ldots, [\boldsymbol{x}]_N = \boldsymbol{x}_N)$, and sends $[\boldsymbol{x}]_j$ to P_j for $j = 1, \ldots, N$. This algorithm requires one round and $|R_q| \cdot N$ bits as communication cost.
– $\boldsymbol{x} \leftarrow \mathsf{Open}([\boldsymbol{x}], U_i)$: Let $U_i \in \mathcal{U}$ be a user who want to reconstruct the shares. P_j sends its share $[\boldsymbol{x}]_j$ to U_i for $j = 1, \ldots, N$. Then, U_i adds the shares and obtains $\boldsymbol{x}$. This algorithm requires one round and $|R_q| \cdot N$ bits as communication cost.

2.4 MK-BFV [10]

The MK-BFV scheme is a natural extension of BFV. It is a tuple $\mathsf{MK\text{-}BFV} = (\mathsf{Setup}, \mathsf{DecKeyGen}, \mathsf{EncKeyGen}, \mathsf{RelinKeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{PartDec}, \mathsf{Merge})$ of probabilistic polynomial time (PPT) algorithms. For each algorithm and protocol taking a ciphertext or a set of keys, we let $\mathsf{id}.$ be an injective mapping from $\{1, \ldots, d\}$ to $\{1, \ldots, k\}$ where $1 \leq d \leq k$.

– $\mathsf{pp} = (t, n, q, \chi_{\mathrm{key}}, \chi_{\mathrm{err}}, \chi_{\mathrm{smdg}}, k) \leftarrow \mathsf{MK\text{-}BFV.Setup}(1^\lambda)$: Return the set of public parameters, where k is the number of users. All the other algorithms in $\mathsf{MK\text{-}BFV}$ implicitly take pp as an argument.
– $\boldsymbol{s} \leftarrow \mathsf{MK\text{-}BFV.DecKeyGen}(\mathsf{pp})$: Compute and output a decryption key $\boldsymbol{s} \leftarrow \chi_{\mathrm{key}}$.
– $(\boldsymbol{p}_0, \boldsymbol{p}_1) \leftarrow \mathsf{MK\text{-}BFV.EncKeyGen}(\boldsymbol{s})$: Given a decryption key $\boldsymbol{s}$, compute and output an encryption key $(\boldsymbol{p}_0, \boldsymbol{p}_1) = (-\boldsymbol{s} \cdot \boldsymbol{p}_1 + \boldsymbol{e}, \boldsymbol{p}_1)$ by sampling $\boldsymbol{p}_1 \leftarrow R_q$ and $\boldsymbol{e} \leftarrow \chi_{\mathrm{err}}$.
– $\mathsf{rlk}_{\{\mathsf{id}_1, \ldots, \mathsf{id}_d\}} \leftarrow \mathsf{MK\text{-}BFV.RelinKeyGen}(\boldsymbol{s}_{\mathsf{id}_1}, \ldots, \boldsymbol{s}_{\mathsf{id}_d})$: Given a set of d decryption keys $\boldsymbol{s}_{\mathsf{id}_1}, \ldots, \boldsymbol{s}_{\mathsf{id}_d}$ of users $U_{\mathsf{id}_1}, \ldots, U_{\mathsf{id}_d}$, generate and output a relinearization key $\mathsf{rlk}_{\{\mathsf{id}_1, \ldots, \mathsf{id}_d\}}$ for ciphertexts under the keys of the same set of users. For more details, see [10].
– $(\boldsymbol{c}_0, \boldsymbol{c}_i) \leftarrow \mathsf{MK\text{-}BFV.Encrypt}(\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{m})$: Given an encryption key $(\boldsymbol{p}_0, \boldsymbol{p}_1)$ and a plaintext $\boldsymbol{m} \in R_t$, compute and output a ciphertext $(\boldsymbol{c}_0, \boldsymbol{c}_1) = (\Delta \cdot (\boldsymbol{m} \bmod q) + \boldsymbol{u} \cdot \boldsymbol{p}_0 + \boldsymbol{e}_0, \boldsymbol{u} \cdot \boldsymbol{p}_1 + \boldsymbol{e}_1)$ by sampling $\boldsymbol{u} \leftarrow \chi_{\mathrm{key}}$ and $\boldsymbol{e}_0, \boldsymbol{e}_1 \leftarrow \chi_{\mathrm{err}}$ where $\Delta = \lfloor q/t \rfloor$.
– $\boldsymbol{m} \leftarrow \mathsf{MK\text{-}BFV.Decrypt}(\boldsymbol{s}_{\mathsf{id}_1}, \ldots, \boldsymbol{s}_{\mathsf{id}_d}, \boldsymbol{c}_0, \boldsymbol{c}_{\mathsf{id}_1}, \ldots, \boldsymbol{c}_{\mathsf{id}_d})$: Given a set of d $(1 \leq d \leq k)$ decryption keys $\boldsymbol{s}_{\mathsf{id}_1}, \ldots, \boldsymbol{s}_{\mathsf{id}_d}$ of users $U_{\mathsf{id}_1}, \ldots, U_{\mathsf{id}_d}$ and a corresponding ciphertext $(\boldsymbol{c}_0, \boldsymbol{c}_{\mathsf{id}_1}, \ldots, \boldsymbol{c}_{\mathsf{id}_d})$, decrypt the ciphertext and output the plaintext as $\boldsymbol{m} = \lfloor (t/q) \cdot \langle (\boldsymbol{c}_0, \boldsymbol{c}_{\mathsf{id}_1}, \ldots, \boldsymbol{c}_{\mathsf{id}_d}), (1, \boldsymbol{s}_{\mathsf{id}_1}, \ldots, \boldsymbol{s}_{\mathsf{id}_d}) \rangle \rceil \bmod t$.

The number of elements in a ciphertext of MK-BFV [10] can be more than two. This is because the ciphertext is *extended* before homomorphic operations in MK-BFV. Precisely, let $\mathsf{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_{\mathsf{id}_1}, \ldots, \boldsymbol{c}_{\mathsf{id}_d})$ and $\mathsf{ct}' = (\boldsymbol{c}'_0, \boldsymbol{c}'_{\mathsf{id}'_1}, \ldots, \boldsymbol{c}'_{\mathsf{id}'_{d'}})$ be two ciphertexts, where $1 \leq d, d' \leq k$, and where $\mathsf{id}.$ and $\mathsf{id}'.$ are injective mapping from $\{1, \ldots, d\}$ and $\{1, \ldots, d'\}$, respectively, to $\{1, \ldots, k\}$. Here, we let $\bar{d}$ be the number of users involved in at least either ct or ct', that is, $\mathsf{max}(d, d') \leq \bar{d} \leq k$, and we let $\bar{\mathsf{id}}.$ be an injective mapping from $\{1, \ldots, \bar{d}\}$ to $\{1, \ldots, k\}$ such that

$\{\mathsf{id}_1, \ldots, \mathsf{id}_d\} \cup \{\mathsf{id}'_1, \ldots, \mathsf{id}'_{d'}\} = \{\bar{\mathsf{id}}_1, \ldots, \bar{\mathsf{id}}_{\bar{d}}\}$. Then, we extend the ciphertexts as $\bar{\mathsf{ct}} = (\boldsymbol{c}_0, \bar{\boldsymbol{c}}_{\bar{\mathsf{id}}_1}, \ldots, \bar{\boldsymbol{c}}_{\bar{\mathsf{id}}_{\bar{d}}})$ and $\bar{\mathsf{ct}}' = (\boldsymbol{c}'_0, \bar{\boldsymbol{c}}'_{\bar{\mathsf{id}}_1}, \ldots, \bar{\boldsymbol{c}}'_{\bar{\mathsf{id}}_{\bar{d}}})$ as follows.

$$\bar{\boldsymbol{c}}_{\bar{\mathsf{id}}_i} = \begin{cases} \boldsymbol{c}_{\mathsf{id}_i} & \text{if } \bar{\mathsf{id}}_i = \mathsf{id}_i \text{ for some } 1 \leq i \leq d \text{, and} \\ 0 & \text{otherwise.} \end{cases}$$

$$\bar{\boldsymbol{c}}'_{\bar{\mathsf{id}}_i} = \begin{cases} \boldsymbol{c}'_{\mathsf{id}'_i} & \text{if } \bar{\mathsf{id}}_i = \mathsf{id}'_i \text{ for some } 1 \leq i \leq d' \text{, and} \\ 0 & \text{otherwise.} \end{cases}$$

After this extension procedure, homomorphic addition can simply be performed by adding each element in ciphertexts. Homomorphic multiplication is also possible by taking a tensor product (also followed by a relinearization operation with a relinearization key $\mathsf{rlk}_{\{\bar{\mathsf{id}}_1, \ldots, \bar{\mathsf{id}}_{\bar{d}}\}}$ to shorten the ciphertext).

However, it is not realistic to assume that a single user holds all the decryption keys in the decryption algorithm. In [10], a decryption method with distributed decryption keys is proposed, which uses the following two algorithms.

- $\boldsymbol{m}_{\mathsf{id}_i} \leftarrow \mathsf{MK\text{-}BFV.PartDec}(\boldsymbol{s}_{\mathsf{id}_i}, \boldsymbol{c}_{\mathsf{id}_i})$: Given a ciphertext element $\boldsymbol{c}_{\mathsf{id}_i}$ ($\mathsf{id}_i \neq 0$) and a corresponding decryption key $\boldsymbol{s}_{\mathsf{id}_i}$, sample a noise $\boldsymbol{e}_{\mathsf{id}_i} \leftarrow \chi_{\text{smdg}}$, and compute and output a partially-decrypted ciphertext with noise as $\boldsymbol{m}_{\mathsf{id}_i} = \boldsymbol{c}_{\mathsf{id}_i} \cdot \boldsymbol{s}_{\mathsf{id}_i} + \boldsymbol{e}_{\mathsf{id}_i}$.
- $\boldsymbol{m} \leftarrow \mathsf{MK\text{-}BFV.Merge}(\boldsymbol{c}_0, \boldsymbol{m}_{\mathsf{id}_1}, \ldots, \boldsymbol{m}_{\mathsf{id}_d})$: Given the first element $\boldsymbol{c}_0$ in a ciphertext and a set of d $(1 \leq d \leq k)$ partially-decrypted elements $\boldsymbol{m}_{\mathsf{id}_1}, \ldots, \boldsymbol{m}_{\mathsf{id}_d}$, decrypt the ciphertext and output the plaintext as
$$\boldsymbol{m} = \left\lfloor \frac{t}{q} \cdot \left(\boldsymbol{c}_0 + \sum_{i=1}^{d} \boldsymbol{m}_{\mathsf{id}_i} \right) \right\rceil \bmod t.$$

The error polynomial $\boldsymbol{e}_{\mathsf{id}_i}$ drawn from a distribution χ_{smdg} is a *smudging* noise to prevent leakage of $\boldsymbol{s}_{\mathsf{id}_i}$ from the sending values [2], where χ_{smdg} has larger variance than χ_{err}. The $\mathsf{MK\text{-}BFV.PartDec}$ algorithm is repeated for all elements in a ciphertext ($i = 1, \ldots, d$). After that, a decryptor runs the $\mathsf{MK\text{-}BFV.Merge}$ algorithm, which results in

$$\left\lfloor \frac{t}{q} \cdot \left(\boldsymbol{c}_0 + \sum_{i=1}^{d} (\boldsymbol{c}_{\mathsf{id}_i} \cdot \boldsymbol{s}_{\mathsf{id}_i} + \boldsymbol{e}_{\mathsf{id}_i}) \right) \right\rceil \bmod t.$$

We can see that the algorithm correctly decrypts the ciphertext as long as the noise including the smudging ones is smaller than the threshold determined by pp.

3 MK-FHE-TRE

3.1 Syntax

Our framework, MK-FHE-TRE, extends MK-FHE to enable converting a ciphertext into one that can be decrypted by a decryptor. After re-encryption, the decryptor can decrypt the ciphertext without using other users' decryption keys nor communicating with other users. Our MK-FHE-TRE is a tuple $\mathsf{MK\text{-}FHE\text{-}TRE} = (\mathsf{Setup}, \mathsf{DecKeyGen}, \mathsf{EncKeyGen}, \mathsf{RelinKeyGen}, \mathsf{Encrypt}, \mathsf{ReKeyGen}, \mathsf{ReEnc}, \mathsf{Decrypt})$ of PPT algorithms.

– $\mathsf{pp} \leftarrow \mathsf{MK\text{-}FHE\text{-}TRE.Setup}(1^\lambda)$: Return the set of public parameters, pp, including the number of users and proxies, k and N, respectively. All the other algorithms in $\mathsf{MK\text{-}FHE\text{-}TRE}$ implicitly take pp as an argument.
– $\mathsf{sk}_i \leftarrow \mathsf{MK\text{-}FHE\text{-}TRE.DecKeyGen}(\mathsf{pp}, U_i)$: $U_i \in \mathcal{U}$ computes and outputs U_i's decryption key sk_i.
– $\mathsf{pk}_i \leftarrow \mathsf{MK\text{-}FHE\text{-}TRE.EncKeyGen}(\mathsf{sk}_i, U_i)$: By using U_i's decryption key sk_i, U_i computes and outputs U_i's encryption key pk_i.
– $\mathsf{rlk}_{\{\mathsf{id}_1,\ldots,\mathsf{id}_d\}} \leftarrow \mathsf{MK\text{-}FHE\text{-}TRE.RelinKeyGen}(\mathsf{sk}_{\mathsf{id}_1}, \ldots, \mathsf{sk}_{\mathsf{id}_d})$: Given a set of d decryption keys $\mathsf{sk}_{\mathsf{id}_1}, \ldots, \mathsf{sk}_{\mathsf{id}_d}$ of users $U_{\mathsf{id}_1}, \ldots, U_{\mathsf{id}_d}$, generate and output a relinearization key $\mathsf{rlk}_{\{\mathsf{id}_1,\ldots,\mathsf{id}_d\}}$ for ciphertexts under the keys of the same set of users.
– $\mathsf{ct}_{\{U_i\}} \leftarrow \mathsf{MK\text{-}FHE\text{-}TRE.Encrypt}(\mathsf{pk}_i, \mathsf{m})$: Given U_i's encryption key pk_i and a plaintext m, an encryptor computes and outputs a ciphertext $\mathsf{ct}_{\{U_i\}}$. Note that we describe the extended ciphertexts encrypted by $\mathsf{pk}_{\mathsf{id}_1}, \ldots, \mathsf{pk}_{\mathsf{id}_d}$ as $\mathsf{ct}_{\{U_{\mathsf{id}_1},\ldots,U_{\mathsf{id}_d}\}}$.
– $(\mathsf{mk}_{i\to D}, [\mathsf{rk}_{i\to D}]) \leftarrow \mathsf{MK\text{-}FHE\text{-}TRE.ReKeyGen}(\mathsf{sk}_i, U_D)$: Given U_i's decryption key sk_i and a delegatee $U_D \in \mathcal{U}$, U_i and U_D run this algorithm and output a masking key $\mathsf{mk}_{i\to D}$ to U_D and the shares $[\mathsf{rk}_{i\to D}]$ of the re-encryption key $\mathsf{rk}_{i\to D}$ that re-encrypts a ciphertext element encrypted with U_i's key to the one encrypted with U_D's key.
– $\mathsf{rct}_{\{U_D\}} \leftarrow \mathsf{MK\text{-}FHE\text{-}TRE.ReEnc}([\mathsf{rk}_{\mathsf{id}_1\to D}], \ldots, [\mathsf{rk}_{\mathsf{id}_d\to D}], \mathsf{ct}_{\{U_{\mathsf{id}_1},\ldots,U_{\mathsf{id}_d}\}})$: Given d $(1 \le d \le k)$ sets of shares $[\mathsf{rk}_{\mathsf{id}_1\to D}], \ldots, [\mathsf{rk}_{\mathsf{id}_d\to D}]$ of re-encryption keys and an (extended) ciphertext $\mathsf{ct}_{\{U_{\mathsf{id}_1},\ldots,U_{\mathsf{id}_d}\}}$, N proxies run this algorithm and output a re-encrypted ciphertext $\mathsf{rct}_{\{U_D\}}$ to U_D.
– $\mathsf{m} \leftarrow \mathsf{MK\text{-}FHE\text{-}TRE.Decrypt}(\mathsf{mk}_{\mathsf{id}_1\to D}, \ldots, \mathsf{mk}_{\mathsf{id}_d\to D}, \mathsf{rct}_{\{U_D\}})$: Given a set of d $(1 \le d \le k)$ masking keys $\mathsf{mk}_{\mathsf{id}_1\to D}, \ldots, \mathsf{mk}_{\mathsf{id}_d\to D}$ and a corresponding re-encrypted ciphertext $\mathsf{rct}_{\{U_D\}}$, U_D run this algorithm and output the plaintext m.

3.2 Algorithms

We construct the instantiation of MK-FHE-TRE based on MK-BFV [10]. We call our instantiation MK-BFV-TRE.

$\mathsf{MK\text{-}FHE\text{-}TRE.Setup}$ is identical to $\mathsf{MK\text{-}BFV.Setup}$ except that pp includes N. $\mathsf{MK\text{-}FHE\text{-}TRE.DecKeyGen}$, $\mathsf{MK\text{-}FHE\text{-}TRE.EncKeyGen}$, $\mathsf{MK\text{-}FHE\text{-}TRE.RelinKeyGen}$, and $\mathsf{MK\text{-}FHE\text{-}TRE.Encrypt}$ are also identical to $\mathsf{MK\text{-}BFV.DecKeyGen}$, $\mathsf{MK\text{-}BFV.EncKeyGen}$, $\mathsf{MK\text{-}BFV.RelinKeyGen}$, and $\mathsf{MK\text{-}BFV.Encrypt}$, respectively. Hence, we omit the description of these algorithms.

Algorithm 1 describes the $\mathsf{MK\text{-}FHE\text{-}TRE.ReKeyGen}$ algorithm, which is run by two users U_i and U_D where $i \neq D$. U_i and U_D are delegator and delegatee of the decryption authority. Here, U_D first generates a masking key $\boldsymbol{r}_{i\to D}$, and send it to U_i. Then, U_i computes its re-encryption key as $\mathsf{rk}_{i\to D} = \boldsymbol{s}_i - \boldsymbol{r}_{i\to D}$. U_i

```
Input: $U_i$'s decryption key $\mathsf{sk}_i = \boldsymbol{s}_i \leftarrow \chi_{\text{key}}$ and a decryptor $U_D$.
Output: A masking key $\mathsf{mk}_{i\to D}$ of $U_D$, and shares of the re-encryption key
        $[\mathsf{rk}_{i\to D}]$ where $\mathsf{rk}_{i\to D}$ re-encrypts a ciphertext element encrypted
        with $U_i$'s key to the one encrypted with the decryptor's key.
1 $U_D$ randomly generates a masking key $\mathsf{mk}_{i\to D} = \boldsymbol{r}_{i\to D} \overset{\text{U}}{\leftarrow} R_q$, and send it to
  $U_i$.
2 $U_i$ computes the re-encryption key $\mathsf{rk}_{i\to D} = \boldsymbol{s}_i - \boldsymbol{r}_{i\to D}$, and distributes it
  among $N$ proxies as $[\mathsf{rk}_{i\to D}]$ by $\mathsf{Share}(\mathsf{rk}_{i\to D}, U_i)$.
3 $U_D$ outputs $\mathsf{mk}_{i\to D}$, and proxies output $[\mathsf{rk}_{i\to D}]$.
```
Algorithm 1: MK-FHE-TRE.ReKeyGen

distributes its re-encryption key among N proxies, $P_1, \ldots, P_N$ by (N, N)-ASS as $[\mathsf{rk}_{i\to D}]_j$.

We note that $\mathsf{rk}_{i\to D}$ is a *masked* decryption key by $\boldsymbol{r}_{i\to D}$. Hence, as explained in §1.3, $\mathsf{rk}_{i\to D}$ does not leak the information of U_i's decryption key unless an adversarial U_D obtains the reconstructed re-encryption key $\mathsf{rk}_{i\to D}$. To reconsturct $\mathsf{rk}_{i\to D}$, the adversarial U_D needs to collude with all the N proxies. Therefore, the re-encryption key does not leak the delegator's decryption key in our MK-BFV-TRE unless an adversary corrupts U_D and N proxies.

```
Input: Shares of re-encryption keys $[\mathsf{rk}_{\mathsf{id}_1\to D}], \ldots, [\mathsf{rk}_{\mathsf{id}_d\to D}]$ and an
       (extended) ciphertext $\mathsf{ct}_{\{U_{\mathsf{id}_1},\ldots,U_{\mathsf{id}_d}\}} = (\boldsymbol{c}_0, \boldsymbol{c}_{\mathsf{id}_1}, \ldots, \boldsymbol{c}_{\mathsf{id}_d})$ generated by
       MK-FHE-TRE.Encrypt and extension under the keys of users $U_{\mathsf{id}_1}, \ldots,$
       $U_{\mathsf{id}_d}$ where $1 \le \mathsf{id}_1, \ldots, \mathsf{id}_d \le k$ and $1 \le d \le k$.
Output: A re-encrypted ciphertext $\mathsf{rct}_{\{U_D\}}$.
1 $P_j$ $(j = 1, \ldots, N)$ locally computes
      $$[\boldsymbol{c}_0']_j = \sum_{i=1}^{d} \boldsymbol{c}_{\mathsf{id}_i} \cdot [\mathsf{rk}_{\mathsf{id}_i\to D}]_j + \boldsymbol{e}_j + \begin{cases} \boldsymbol{c}_0 & \text{if } j = 1, \text{ and} \\ 0 & \text{otherwise,} \end{cases}$$
  where $\boldsymbol{e}_j \leftarrow \chi_{\text{smdg}}$ is a smudging noise.
2 Proxies send $\boldsymbol{c}_{\mathsf{id}_1}, \ldots, \boldsymbol{c}_{\mathsf{id}_d}$ to $U_D$.
3 $U_D$ reconstructs $\boldsymbol{c}_0'$ by $\mathsf{Open}([\boldsymbol{c}_0'], U_D)$ and outputs $\mathsf{rct}_{\{U_D\}} = (\boldsymbol{c}_0', \boldsymbol{c}_{\mathsf{id}_1}' = \boldsymbol{c}_{\mathsf{id}_1},$
  $\ldots, \boldsymbol{c}_{\mathsf{id}_d}' = \boldsymbol{c}_{\mathsf{id}_d})$.
```
Algorithm 2: MK-FHE-TRE.ReEnc

Algorithm 2 describes the MK-FHE-TRE.ReEnc algorithm. Each proxy P_j computes a share of

$$\boldsymbol{c}_0' = \boldsymbol{c}_0 + \sum_{i=1}^{d} \boldsymbol{c}_{\mathsf{id}_i} \cdot \mathsf{rk}_{\mathsf{id}_i\to D} + \sum_{j=1}^{N} \boldsymbol{e}_j,$$

as $[\boldsymbol{c}_0']_j$. Then, proxies send the elements of the ciphertext $(\boldsymbol{c}_{\mathsf{id}_1}, \ldots, \boldsymbol{c}_{\mathsf{id}_d})$ to U_D. Finally, U_D reconstructs $\boldsymbol{c}_0'$ by $\mathsf{Open}([\boldsymbol{c}_0'], U_D)$, and returns a re-encrypted ciphertext $\mathsf{rct}_{\{U_D\}} = (\boldsymbol{c}'_0, \boldsymbol{c}'_{\mathsf{id}_1} = \boldsymbol{c}_{\mathsf{id}_1}, \ldots, \boldsymbol{c}'_{\mathsf{id}_d} = \boldsymbol{c}_{\mathsf{id}_d})$. The remaining d elements in this tuple is the same as in the ciphertext before re-encryption.

```
Input: Masking keys $\mathsf{mk}_{\mathsf{id}_1 \to D} = \boldsymbol{r}_{\mathsf{id}_1 \to D}, \ldots, \mathsf{mk}_{\mathsf{id}_d \to D} = \boldsymbol{r}_{\mathsf{id}_d \to D}$, and a
       re-encrypted ciphertext $\mathsf{rct}_{\{U_D\}} = (\boldsymbol{c}'_0, \boldsymbol{c}'_{\mathsf{id}_1}, \ldots, \boldsymbol{c}'_{\mathsf{id}_d})$.
Output: The corresponding plaintext $\boldsymbol{m} \in R_t$.
1 $U_D$ outputs $\boldsymbol{m}$ by $\mathsf{MK\text{-}BFV.Decrypt}(\boldsymbol{c}'_0, \boldsymbol{c}'_{\mathsf{id}_1}, \ldots, \boldsymbol{c}'_{\mathsf{id}_d}, \boldsymbol{r}_{\mathsf{id}_1 \to D}, \ldots, \boldsymbol{r}_{\mathsf{id}_d \to D})$.
```

Algorithm 3: MK-FHE-TRE.Decrypt

Note that the re-encrypted ciphertext is no longer decryptable by using only users' decryption keys in the straightforward manner because the masking keys of U_D are embedded in the re-encrypted ciphertext through MK-FHE-TRE.ReEnc. Therefore, we need to employ the MK-FHE-TRE.Decrypt algorithm to decrypt the re-encrypted ciphertext, of which procedure is described in Algorithm 3. Here, since

$$\begin{aligned}
&\boldsymbol{c}'_0 + \sum_{i=1}^{d} \boldsymbol{c}'_{\mathsf{id}_i} \cdot \boldsymbol{r}_{\mathsf{id}_i \to D} \\
&= \boldsymbol{c}_0 + \sum_{i=1}^{d} \boldsymbol{c}_{\mathsf{id}_i} \cdot \mathsf{rk}_{\mathsf{id}_i \to D} + \sum_{j=1}^{N} \boldsymbol{e}_j + \sum_{i=1}^{d} \boldsymbol{c}_{\mathsf{id}_i} \cdot \boldsymbol{r}_{\mathsf{id}_i \to D} \\
&= \boldsymbol{c}_0 + \sum_{i=1}^{d} \boldsymbol{c}_{\mathsf{id}_i} \cdot (\boldsymbol{s}_{\mathsf{id}_i} - \boldsymbol{r}_{\mathsf{id}_i \to D}) + \sum_{j=1}^{N} \boldsymbol{e}_j + \sum_{i=1}^{d} \boldsymbol{c}_{\mathsf{id}_i} \cdot \boldsymbol{r}_{\mathsf{id}_i \to D} \\
&= \left(\boldsymbol{c}_0 + \sum_{i=1}^{d} \boldsymbol{c}_{\mathsf{id}_i} \cdot \boldsymbol{s}_{\mathsf{id}_i} \right) + \sum_{j=1}^{N} \boldsymbol{e}_j,
\end{aligned}$$

$\mathsf{MK\text{-}BFV.Decrypt}(\boldsymbol{r}_{\mathsf{id}_1 \to D}, \ldots, \boldsymbol{r}_{\mathsf{id}_d \to D}, \boldsymbol{c}'_0, \boldsymbol{c}'_{\mathsf{id}_1}, \ldots, \boldsymbol{c}'_{\mathsf{id}_d})$ correctly outputs the plaintext $\boldsymbol{m}$ as long as the noise including the terms $\sum_{j=1}^{N} \boldsymbol{e}_j$ is small enough. We note that $\sum_{j=1}^{N} \boldsymbol{e}_j$ corresponds to the smudging noise $\sum_{j=1}^{k} \boldsymbol{e}_j$ in MK-BFV [10]. In [10], the smudging noise $\sum_{j=1}^{k} \boldsymbol{e}_j$ must be small enough that a decryptor can decrypt the ciphertext. Hence, our assumption that $\sum_{j=1}^{N} \boldsymbol{e}_j$ is sufficiently small is as reasonable as [10].

4 Security Proof of MK-BFV-TRE

As a security proof strategy, we consider the MK-BFV-TRE to be an extension of MK-BFV [10] with secure re-encryption protocols in the universal composability (UC) framework [7]. The established UC-secure protocols remain secure even when executed in parallel or concurrently with other secure or insecure protocols.

Therefore, we can prove the security of the entire MK-BFV-TRE under RLWE assumption if our proposed protocols about re-encryption are UC-secure.

As well as the original MK-BFV [10], $\mathsf{MK\text{-}FHE\text{-}TRE.Setup}$, $\mathsf{MK\text{-}FHE\text{-}TRE.DecKeyGen}$, $\mathsf{MK\text{-}FHE\text{-}TRE.EncKeyGen}$, $\mathsf{MK\text{-}FHE\text{-}TRE.RelinKeyGen}$, and $\mathsf{MK\text{-}FHE\text{-}TRE.Encrypt}$ are secure under RLWE assumption because these algorithms are identical to the algorithms of the MK-BFV except that pp includes N.

Note that $\mathsf{MK\text{-}FHE\text{-}TRE.Decrypt}$ consists of performing operations by a decryptor without communications. Hence, if sufficient smudging noise is added to the re-encrypted ciphertext, the decryptor can obtain only the plaintext as in the original MK-BFV [10]. We assume that a semi-honest adversary can corrupt up to N parties from N proxies and the decryptor. Therefore, even if the adversary corrupts $N-1$ proxies and the decryptor, the remaining uncorrupted proxy adds the smudging noise to the re-encrypted ciphertext during $\mathsf{MK\text{-}FHE\text{-}TRE.ReEnc}$, and the added smudging noise prevents information leakage regarding the users' decryption keys during $\mathsf{MK\text{-}FHE\text{-}TRE.Decrypt}$. Therefore, $\mathsf{MK\text{-}FHE\text{-}TRE.Decrypt}$ is as secure as $\mathsf{MK\text{-}BFV.Decrypt}$ [10] under RLWE assumption.

Hence, we focus on proving the security of $\mathsf{MK\text{-}FHE\text{-}TRE.ReKeyGen}$ and $\mathsf{MK\text{-}FHE\text{-}TRE.ReEnc}$ in the UC framework [7]. If we prove the security of $\mathsf{MK\text{-}FHE\text{-}TRE.ReKeyGen}$ and $\mathsf{MK\text{-}FHE\text{-}TRE.ReEnc}$ in the UC framework, we can prove the security of the entire MK-BFV-TRE under RLWE assumption.

$\mathsf{MK\text{-}FHE\text{-}TRE.ReKeyGen}$ consists of sharing a masking key and Share. For this reason, it is UC-secure under the assumptions of the secure channel and Share, i.e., UC-secure (N, N)-ASS.

Finally, we prove the security of $\mathsf{MK\text{-}FHE\text{-}TRE.ReEnc}$ in a hybrid model [7].

4.1 Security Definition

Real model. Let Π_{ReEnc} be the actual protocol of $\mathsf{MK\text{-}FHE\text{-}TRE.ReEnc}$. We also denote a semi-honest adversary by $\mathcal{A}$. $\mathcal{A}$ is a non-uniform PPT adversary.

Here, N proxies and a delegatee in $\mathcal{U}$ are regarded as $N+1$ parties without distinction. For simplicity of description, we temporarily set the delegatee as P_{N+1} in §4.1 and §4.2. We denote the number of corruptions by $\mathcal{A}$ by $T(< N+1)$. The output of the honest parties, the set of corrupted parties $I \subset \mathcal{P} \cup \{P_{N+1}\}$, and $\mathcal{A}$ in a real execution of Π_{ReEnc}, with inputs $x_1, \ldots, x_{N+1}$, auxiliary input aux for $\mathcal{A}$, and security parameter λ is denoted by $\mathrm{REAL}_{\Pi_{\mathsf{ReEnc}}, \mathcal{A}(\mathsf{aux}), I}(x_1, \ldots, x_{N+1}, \lambda)$.

Ideal model. We define the ideal model for the ideal functionality of our re-encryption $\mathcal{F}_{\mathsf{ReEnc}}$, receiving inputs from $N+1$ parties and providing them with outputs. We also define f as $(N+1)$-party functionality. The ideal execution proceeds as follows.

– **Send inputs to trusted party:** Each honest party P_j $(j = 1, \ldots, N+1)$ sends its specified input x_j to the trusted party.
– **Answer the parties by trusted party:** The trusted party computes $f(x_1, \ldots, x_{N+1}) = (y_1, \ldots, y_{N+1})$. It sends y_j to P_j for $j = 1, \ldots, N+1$.

– **Outputs:** All parties always output the output value they received from the trusted party. $\mathcal{A}$ outputs the initial inputs $\{x_i\}_{i \in I}$ and the messages received by the corrupted parties that are sent from the trusted party $\{y_i\}_{i \in I}$.

Let $\mathcal{S}$ be a non-uniform PPT adversary controlling $\{P_i\}_{i \in I}$. We also denote the output of the honest parties, the set of corrupted parties I and $\mathcal{S}$ in an ideal execution with $\mathcal{F}_{\mathsf{ReEnc}}$, with inputs $x_1, \ldots, x_{N+1}$, auxiliary input aux for $\mathcal{S}$, and security parameter λ by $\mathrm{IDEAL}_{\mathcal{F}_{\mathsf{ReEnc}}, \mathcal{S}(\mathsf{aux}), I}(x_1, \ldots, x_{N+1}, \lambda)$.

Definition 1. *Let $\mathcal{F}_{\mathsf{ReEnc}}$ be a $(N+1)$-party functionality, and let Π_{ReEnc} be a $(N+1)$-party protocol. We say that Π_{ReEnc} securely computes $\mathcal{F}_{\mathsf{ReEnc}}$ in the presence of an adversary controlling N semi-honest corrupted parties, if for every non-uniform PPT adversary $\mathcal{A}$ in the real world, there exists a non-uniform PPT simulator/adversary $\mathcal{S}$ in the ideal model with $\mathcal{F}_{\mathsf{ReEnc}}$ such that for I,*

$$\begin{aligned} &\{\mathrm{IDEAL}_{\mathcal{F}_{\mathsf{ReEnc}}, \mathcal{S}(\mathsf{aux}), I}(x_1, \ldots, x_{N+1}, \lambda)\} \\ \equiv\, &\{\mathrm{REAL}_{\Pi_{\mathsf{ReEnc}}, \mathcal{A}(\mathsf{aux}), I}(x_1, \ldots, x_{N+1}, \lambda)\} \end{aligned}$$

where $x_1, \ldots, x_{N+1}$ under the constraint that $|x_1| = \cdots = |x_{N+1}|$ and $\lambda \in \mathbb{N}$.

4.2 Security Proof

In a hybrid model [7], each party runs the real protocol with actual messages and can access the ideal subfunctionality that a trusted party computes. Let g be the subfunctionality computed by a trusted party, which can be replaced with a secure real protocol. Then, we say that protocol Π is secure in the g-hybrid model. We denote protocol Π secure in the g-hybrid model as Π^g. We also assume *input availability* (i.e., the inputs of all parties are fixed before the execution of the protocol begins) to prove UC of $\mathsf{MK\text{-}FHE\text{-}TRE.ReEnc}$. Hence, it is sufficient that we prove the security of $\mathsf{MK\text{-}FHE\text{-}TRE.ReEnc}$ in the classic stand-alone setting and automatically derive UC from a previous study [23].

Theorem 1. *If we assume the security of* Open *and let $\mathcal{F}_{\mathsf{Open}}$ be the ideal functionality of opening shares on (N, N)-ASS,* $\mathsf{MK\text{-}FHE\text{-}TRE.ReEnc}$ *protocol Π_{ReEnc} in the $\mathcal{F}_{\mathsf{Open}}$-hybrid model compute $\mathcal{F}_{\mathsf{ReEnc}}$ in the presence of an adversary controlling N semi-honest corrupted parties.*

Proof. In Π_{ReEnc}, we replace the call to Open by invoking $\mathcal{F}_{\mathsf{Open}}$. Then, $\Pi_{\mathsf{ReEnc}}^{\mathcal{F}_{\mathsf{Open}}}$ consists of invoking $\mathcal{F}_{\mathsf{Open}}$ and computations without communications. Note that the view of P_{N+1} can be constructed easily by Share because P_{N+1} receives only the shares of $\boldsymbol{c}'_0$ and original ciphertexts $(\boldsymbol{c}'_{\mathsf{id}_1} = \boldsymbol{c}_{\mathsf{id}_1}, \ldots, \boldsymbol{c}'_{\mathsf{id}_d} = \boldsymbol{c}_{\mathsf{id}_d})$. That is, P_{N+1}'s view regarding shares of $\boldsymbol{c}'_0$ can be generated by sampling random elements $\boldsymbol{a}_2, \ldots, \boldsymbol{a}_N$ from R_q and setting $[\boldsymbol{c}'_0]_1 = \boldsymbol{c}'_0 - \sum_{j=2}^{N} \boldsymbol{a}_j$ and $[\boldsymbol{c}'_0]_j = \boldsymbol{a}_j$ for $j = 2, \ldots, N$.

Hence, $\mathcal{S}$ can be composed when $\mathcal{A}$ corrupts N parties. Therefore, since Definition 1 is satisfied, Π_{ReEnc} securely computes $\mathcal{F}_{\mathsf{ReEnc}}$ in the presence of an adversary controlling N semi-honest corrupted parties assuming that Open is secure, i.e., (N, N)-ASS is UC-secure.

$\mathcal{F}_{\mathsf{ReEnc}}$ - (threshold proxy re-encryption)

1. Proxy P_j sends the message $(\mathsf{ReEnc}, \{[\mathsf{rk}_{\mathsf{id}_i}]_j\}_{i=1}^{d}, (\boldsymbol{c}_0, \boldsymbol{c}_{\mathsf{id}_1}, \ldots, \boldsymbol{c}_{\mathsf{id}_d}))$ to $\mathcal{F}_{\mathsf{ReEnc}}$ for $j = 1, \ldots, N$.
2. After receiving the messages, $\mathcal{F}_{\mathsf{ReEnc}}$ reconstructs the values, $\{\mathsf{rk}_{\mathsf{id}_i}\}_{i=1}^{d}$ by using $\{[\mathsf{rk}_{\mathsf{id}_i}]_j\}_{i=1}^{d}$ for $j = 1, \ldots, N$.
3. $\mathcal{F}_{\mathsf{ReEnc}}$ samples $\boldsymbol{e}_j \overset{\mathrm{U}}{\leftarrow} \chi_{\mathrm{smdg}}$ for $j = 1, \ldots, N$.
4. $\mathcal{F}_{\mathsf{ReEnc}}$ computes
$$\boldsymbol{c}_0' = \boldsymbol{c}_0 + \sum_{i=1}^{d} \boldsymbol{c}_{\mathsf{id}_i} \cdot \mathsf{rk}_{\mathsf{id}_i} + \sum_{j=1}^{N} \boldsymbol{e}_j.$$
5. $\mathcal{F}_{\mathsf{ReEnc}}$ sends $(\boldsymbol{c}_0', \boldsymbol{c}_{\mathsf{id}_1}' = \boldsymbol{c}_{\mathsf{id}_1}, \ldots, \boldsymbol{c}_{\mathsf{id}_d}' = \boldsymbol{c}_{\mathsf{id}_d})$ to P_{N+1} (that is a delegatee in $\mathcal{U}$).

Fig. 1. Ideal functionality for threshold proxy re-encryption

5 Performance Analysis

5.1 Implementation and Parameters

We implement our MK-BFV-TRE scheme in C++ from scratch except for using the NTL library [31] (for polynomial arithmetic over multi-precision integers) to show performance[1]. In our implementation, each coefficient of a secret $\boldsymbol{s} \in R_q$ and an error $\boldsymbol{e} \in R_q$ is drawn according to the discrete Gaussian distribution with standard deviation $\sigma_{\mathrm{err}} \approx 3.2$.

In the noise smudging algorithm (in $\mathsf{MK\text{-}BFV.PartDec}$ and $\mathsf{MK\text{-}FHE\text{-}TRE.ReEnc}$), we use an error polynomial whose coefficients are drawn from the discrete Gaussian distribution with a standard deviation $\sigma_{\mathrm{smdg}} = 2^{20}$, which is the same one used for smudging noise in the OpenFHE library [4]. Theoretically, we should make σ_{smdg} large as 2^{λ} for the security parameter λ according to Smudging lemma [2]. However, a reasonably large standard deviation is often used for efficiency in practice, as in the OpenFHE library. We choose σ_{smdg} from a practical standpoint as well as other implementations.

We set the plaintext modulus $t = 2^8$ and the (maximum possible) arithmetic circuit depth $L = 5$ in our experiment. Since it is well known that the noise growth of homomorphic multiplication is much more significant than the noise growth of homomorphic addition, we estimate the noise from a depth-L circuit of homomorphic multiplications. In addition, we only considered the dominant terms of the noise as in [10]. However, our noise estimation is bound-based as in [18], which tends to result in larger parameters compared to a variant-based approach in [10] in exchange for a higher decryption success rate.

[1] We note that there is room for improvement in optimization in our MK-BFV-TRE implementation because we do not apply the residue number system technique which avoids multi-precision integers computation as in the implementation of the original MK-BFV in [10], for simplicity.

Table 2. Parameters for which we used different values depending on the number of users.

#Users ($= k$)	2	4	8
$\log b$	46	40	34
$\lceil \log_b q \rceil$	5	6	7

Table 3. Size of primitives in MK-BFV-TRE scheme. MB=10^6 bytes.

#Users ($= k$)		2	4	8
Size [MB]	Decryption key $\boldsymbol{s}$	0.07		
	Encryption key $(\boldsymbol{p}_0, \boldsymbol{p}_1)$	0.45		
	Relinearization key rlk	3.38	4.06	4.73
	Ciphertext $(\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_k)$	0.68	1.13	2.03
	Masking key $\boldsymbol{r}_{\mathsf{id}.\to D}$	0.23		
	ReEnc key $\mathsf{rk}_{\mathsf{id}.\to D}$	0.23		

According to Homomorphic Encryption Standard [1], we used the parameters $\log q = 220$, $n = 8192$, and $\sigma_{\text{err}} \approx 3.2$ that provide a 128-bit of security level. As a result of estimation for the noise growth and security, we can use the same parameters when the number of users k is equal to $2, 4, 8$. We do not consider the case of $k = 1$ since the re-encryption algorithm is meaningless in that case.

The parameters for which we used different values depending on the number of users are provided in Table 2. The $\log b$ row shows the bit length of the base of the decomposition algorithm, which is a subroutine in MK-BFV.Relinearize. We remark that the base of the decomposition b affects the accumulated noise during MK-BFV.Relinearize, a subroutine of homomorphic multiplication, as shown in [10]. In addition, the dimension of the *gadget vector* $\lceil \log_b q \rceil$ affects the size of the rlk and the computational complexity of MK-BFV.Relinearize.

In Table 3, we estimate the size of primitives of our MK-BFV-TRE scheme in megabytes. We remark that the form of the decryption key, the encryption key, the relinearization key, and the ciphertext are all identical to the original MK-BFV scheme in [10]. The size of the decryption key $\boldsymbol{s}$ is estimated by assuming $|\boldsymbol{s}|_\infty \leq 10 \cdot \sigma_{\text{err}}$.

5.2 Our Experiments

Measurement of execution time. Our implementation is executed on a single core of Intel Xeon Silver 4114 CPU clocked at 2.20 GHz with 96 GB of memory. The program is compiled with GCC 7.5.0 (with `-O2` option) on Ubuntu 18.04.

We measured the execution time for key generation, encryption, homomorphic addition and multiplication, threshold decryption (partial decryption and merge), proxy re-encryption, and decryption using Google's Benchmark library [20]. As for execution time, we take an average of 10 times trials.

Table 4. Measured execution time of algorithms without communications of the MK-BFV-TRE scheme. ms=10^{-3}sec.

#Users ($= k$)		2	4	8
Exec. [ms]	KeyGen	651	782	911
	Encrypt	50.4		
	HomAdd	1.05	1.77	3.27
	HomMul	2345	10438	47022

Table 4 shows the time of key generation, encryption, and homomorphic addition and multiplication in our MK-FHE-TRE scheme for $k \in \{2, 4, 8\}$ where k is the number of users. We note that the parameter $d \in \{1, \ldots, k\}$, which is the number of users involved in the homomorphic computation for a certain ciphertext, is implicitly set to $d = k$ for brevity in our experiments. That is, we use extended ciphertexts in R_q^{k+1} to measure the time of HomAdd and HomMul. We remark that these algorithms are identical to the original MK-BFV scheme [10]. Therefore, the execution time of these algorithms is irrelevant to the number of proxies. In addition, these algorithms do not need network communication. In Table 4, the KeyGen row is the total time of MK-BFV.DecKeyGen, MK-BFV.EncKeyGen, and MK-BFV.RelinKeyGen for one user.

Table 5. Measured execution time and estimated communication time of threshold decryption, i.e., combination of MK-FHE-TRE.PartDec and MK-FHE-TRE.Merge. ms = 10^{-3} sec.

#Users ($= k$)		2	4	8
Exec. [ms]		26.9	27.6	28.9
Comm. [ms]	LAN	2.04	4.75	10.2
	WAN	290	677	1451

Table 6. Measured execution time and estimated communication time of MK-FHE-TRE.ReKeyGen. ms = 10^{-3} sec.

#Proxies ($= N$)		1	2	4	8
Exec. [ms]		51.2	102	203	409
Comm. [ms]	LAN	1.36	2.03	3.39	6.10
	WAN	193	290	484	871

Tables 5, 6, and 7 show the execution time of the threshold decryption (i.e., the combination of MK-BFV.PartDec and MK-BFV.Merge), MK-FHE-TRE.ReKeyGen, and MK-FHE-TRE.ReEnc, respectively. We assume that users and proxies can execute their local computations in parallel. Hence, the execution

Table 7. Measured execution time and estimated communication time of MK-FHE-TRE.ReEnc. ms = 10^{-3} sec.

#Users (= k)		2				4				8			
#Proxies (= N)		1	2	4	8	1	2	4	8	1	2	4	8
Exec. [ms]		250				298				393			
Comm. [ms]	LAN	2.03	2.71	4.07	6.78	3.39	4.07	5.42	8.14	6.10	6.78	8.14	10.8
	WAN	290	387	580	967	484	580	774	1169	871	967	1161	1548

Table 8. Measured execution time of MK-FHE-TRE.Decrypt. ms = 10^{-3} sec.

#Users (= k)	2	4	8
Exec. [ms]	0.05	0.95	1.85

time shown in these tables is the maximum total computation time for one user or one proxy.

Table 5 shows that the effect of the increase in the number of users on the execution time of threshold decryption is small even though MK-BFV.Merge requires $\mathcal{O}(k)$ addition over R_q because the MK-BFV.PartDec process is the dominant according to our profiling.

In Table 6, we can see that the execution time of MK-FHE-TRE.ReKeyGen appears linear regarding the number of proxies N. In the case of $N = 1$, we remark that one proxy holds all of the rk, and neither Share nor Open happens. It is consistent with the complexity of MK-FHE-TRE.ReKeyGen, which requires $\mathcal{O}(N)$ random polynomial generation and $\mathcal{O}(N)$ times additions over R_q.

Table 7 shows that the execution time of MK-FHE-TRE.ReEnc increases gradually as the number of users k increases. It is reasonable that MK-FHE-TRE.ReEnc requires $\mathcal{O}(k)$ multiplications and $\mathcal{O}(k)$ times additions over R_q.

Estimation of communication time. As in [29], we run each algorithm of MK-BFV-TRE with communications on a single server. We therefore estimate the communication costs assuming a communication environment.

MK-FHE-TRE.ReKeyGen and MK-FHE-TRE.ReEnc in our MK-BFV-TRE scheme and also the threshold decryption algorithm (MK-BFV.PartDec and MK-BFV.Merge) which is inherited from the original MK-BFV [10] need the network communication. The network bandwidth and latency may affect the performance of these algorithms. In Tables 5, 6 and 7, we also give the estimated communication time.

We describe how to estimate the communication time in our protocol. We consider both a local area network (LAN; 10 Gbps throughput, 0.5 ms latency) [27] and a wide area network (WAN; 72 Mbps throughput and 72 ms latency) [28]. The bandwidth and latency of LAN/WAN are based on a measurement between AWS US East and West regions. We assume that the communication channels connecting all users and servers have the same communication bandwidth and communication delay.

In LAN (resp. WAN) setting, we put $\mathsf{throughput} := 10$ [Gbps] $= 10 \cdot 10^{-3}$ [gigabit/ms] (resp. 72 [Mbps] $= 72 \cdot 10^{-3}$ [megabit/ms]) and $\mathsf{latency} := 0.5$ [ms] (resp. 72 [ms]). When a party sends an element in R_q, i.e., $n \cdot \log q$ bits, the required communication time is

$$\mathsf{CT}_{R_q} := \frac{n \cdot \log q}{\mathsf{throughput}} + \mathsf{latency} \text{ [ms]}.$$

The following is how to estimate the communication time of MK-FHE-TRE. ReKeyGen, MK-FHE-TRE.ReEnc, and MK-BFV.PartDec.

MK-FHE-TRE.ReKeyGen: When the delegatee U_D sends the masking key $\boldsymbol{r}_{i \to D} \in R_q$ to the user U_i, the communication time required is CT_{R_q}. In addition, sending the piece of share $[\mathsf{rk}_{i \to D}]_\ell$ to the proxy $P_\ell \in \mathcal{P}$ for all $\ell \in \{1, \ldots, N\}$ from the user U_i takes $N \cdot \mathsf{CT}_{R_q}$. As a result, the total communication time for MK-FHE-TRE.ReKeyGen is $(N + 1) \cdot \mathsf{CT}_{R_q}$.

MK-FHE-TRE.ReEnc: For $\ell \in \{1, \ldots, N\}$, the proxy ℓ needs to send $[\boldsymbol{c}'_0]_\ell \in R_q$ to U_D, which takes $N \cdot \mathsf{CT}_{R_q}$. After that one proxy sends $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k \in R_q$ to U_D, which takes $k \cdot \mathsf{CT}_{R_q}$. Therefore, we see that the total communication time for MK-FHE-TRE.ReEnc is $(k + N) \cdot \mathsf{CT}_{R_q}$.

MK-BFV.PartDec: First, the computation server sends $\boldsymbol{c}_i$ to each user U_i, which takes $k \cdot \mathsf{CT}_{R_q}$. After that, each user except for U_D sends the result of MK-BFV.PartDec to U_D, which takes $(k-1) \cdot \mathsf{CT}_{R_q}$. Thus, the total communication time required is $(2 \cdot k - 1) \cdot \mathsf{CT}_{R_q}$. We remark that any proxy is not related in MK-BFV.PartDec.

In our above estimation, we assume that each party among users and proxies communicates sequentially. Hence, our communication time estimation is the worst-case estimation. Actually, each user can use upstream and downstream and communicate in parallel, depending on the network interface they utilize. Therefore, actual communication times are smaller than our estimations.

Threshold decryption vs. re-encryption then decryption. When the number of users k exceeds the number of proxies N, the sum of the communication time of MK-FHE-TRE.ReEnc and that of MK-FHE-TRE.Decrypt is smaller than the communication time of threshold decryption (MK-BFV.PartDec and MK-BFV.Merge), since $(k + N) \cdot \mathsf{CT}_{R_q} < (2 \cdot k - 1) \cdot \mathsf{CT}_{R_q}$ in this situation.

Considering the execution time, MK-FHE-TRE.Decrypt after MK-FHE-TRE. ReEnc is not always faster than the threshold decryption in our experiment, e.g., in the case of $k = 8$ and $N = 2$. However, the execution time of the re-encryption algorithm heavily depends on the machine specification of the proxy servers. By improving computer resources themselves, it is relatively easy to improve the execution time compared to reducing communication time using the network.

Therefore, we believe that MK-FHE-TRE.Decrypt after MK-FHE-TRE.ReEnc is superior to threshold decryption when the number of users is much larger than the number of proxies. In the opposite situation, threshold decryption can be better than MK-FHE-TRE.Decrypt after MK-FHE-TRE.ReEnc in the sense of the time required. Since ciphertexts (before re-encryption) in our MK-BFV-TRE are identical to ciphertexts in MK-BFV, users in our MK-BFV-TRE can choose the threshold decryption and MK-FHE-TRE.ReEnc followed by MK-FHE-TRE.Decrypt according to the environment they are in.

6 Conclusions

We proposed the new framework, MK-FHE-TRE, and its instantiation based on MK-BFV [10], MK-BFV-TRE. MK-BFV-TRE can avoid not only the threshold decryption but also the decryption key compromise from re-encryption keys unless the adversarial delegatee colludes with N proxies. We also proved the security of MK-BFV-TRE under the RLWE assumption and the UC-secure (N, N)-ASS.

In addition, we implemented our MK-BFV-TRE and measured the size of keys and ciphertexts. By using our implementation, we also measured the running time of each algorithm and estimated the communication time of each algorithm. From these experimental results, we believe that MK-FHE-TRE.Decrypt after MK-FHE-TRE.ReEnc is superior to threshold decryption when the number of users is much larger than the number of proxies.

Since the MK-BFV-TRE can avoid the threshold decryption, secure computing applications built using it do not require the decryption key holders to be online, and there is less need to consider the effects of the communication environment and the user's device.

References

1. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic Encryption Security Standard (2018), http://homomorphicencryption.org/wp-content/uploads/2018/11/HomomorphicEncryptionStandardv1.1.pdf
2. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold fhe. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 7237, pp. 483–501. Springer (2012)
3. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. In: NDSS. The Internet Society (2005)
4. Badawi, A.A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., Zucca, V.: OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Paper 2022/915 (2022), https://eprint.iacr.org/2022/915
5. Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 1403, pp. 127–144. Springer (1998)

6. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory **6**(3), 13:1–13:36 (2014)
7. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS. pp. 136–145. IEEE Computer Society (2001)
8. Canetti, R., Hohenberger, S.: Chosen-ciphertext secure proxy re-encryption. In: CCS. pp. 185–194. ACM (2007)
9. Chen, H., Chillotti, I., Song, Y.: Multi-key homomorphic encryption from TFHE. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 11922, pp. 446–472. Springer (2019)
10. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: CCS. pp. 395–412. ACM (2019)
11. Chen, L., Zhang, Z., Wang, X.: Batched multi-hop multi-key FHE from ring-lwe with compact ciphertext extension. In: TCC (2). Lecture Notes in Computer Science, vol. 10678, pp. 597–627. Springer (2017)
12. Chen, X., Liu, Y., Li, Y., Lin, C.: Threshold proxy re-encryption and its application in blockchain. In: Cloud Computing and Security: 4th International Conference, ICCCS 2018, Haikou, China, June 8–10, 2018, Revised Selected Papers, Part IV 4. pp. 16–25. Springer (2018)
13. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 10624, pp. 409–437. Springer (2017)
14. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. J. Cryptol. **33**(1), 34–91 (2020)
15. community, F.: Fully Homomorphic Encryption — We are a community of researchers and developers interested in advancing homomorphic encryption and other secure computation techniques. (2023), `https://fhe.org/`
16. Derler, D., Ramacher, S., Slamanig, D.: Homomorphic proxy re-authenticators and applications to verifiable multi-user data aggregation. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 10322, pp. 124–142. Springer (2017)
17. Dutta, P., Susilo, W., Duong, D.H., Roy, P.S.: Collusion-resistant identity-based proxy re-encryption: Lattice-based constructions in standard model. Theor. Comput. Sci. **871**, 16–29 (2021)
18. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. p. 144 (2012)
19. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 8042, pp. 75–92. Springer (2013)
20. Benchmark. `https://github.com/google/benchmark/` (2023)
21. ICO: Homomorphic encryption (HE) — ICO (2023), `https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/data-sharing/privacy-enhancing-technologies/what-pets-are-there/homomorphic-encryption-he/`
22. ISO/IEC: ISO/IEC WD 18033-8 - Information security — Encryption algorithms — Part 8: Fully Homomorphic Encryption (2023), `https://www.iso.org/standard/83139.html`
23. Kushilevitz, E., Lindell, Y., Rabin, T.: Information-theoretically secure protocols and security under composition. SIAM J. Comput. **39**(5), 2090–2112 (2010)

24. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: STOC. pp. 1219–1234. ACM (2012)
25. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 6110, pp. 1–23. Springer (2010)
26. Ma, C., Li, J., Ouyang, W.: A homomorphic proxy re-encryption from lattices. In: ProvSec. Lecture Notes in Computer Science, vol. 10005, pp. 353–372 (2016)
27. Mohassel, P., Rindal, P.: ABY3: A Mixed Protocol Framework for Machine Learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. p. 35–52. CCS '18, Association for Computing Machinery, New York, NY, USA (2018), `https://doi.org/10.1145/3243734.3243760`
28. Mohassel, P., Zhang, Y.: SecureML: A System for Scalable Privacy-Preserving Machine Learning. Cryptology ePrint Archive, Paper 2017/396 (2017)
29. Mouchet, C., Troncoso-Pastoriza, J.R., Bossuat, J., Hubaux, J.: Multiparty homomorphic encryption from ring-learning-with-errors. Proc. Priv. Enhancing Technol. **2021**(4), 291–311 (2021)
30. NIST: Multi-Party Threshold Cryptography — CSRC (2023), `https://csrc.nist.gov/Projects/threshold-cryptography`
31. NTL: A library for doing Number Theory. `https://libntl.org/` (2021)
32. Paul, A., Srinivasavaradhan, V., Selvi, S.S.D., Rangan, C.P.: A cca-secure collusion-resistant identity-based proxy re-encryption scheme. In: ProvSec. Lecture Notes in Computer Science, vol. 11192, pp. 111–128. Springer (2018)
33. Peikert, C., Shiehian, S.: Multi-key FHE from lwe, revisited. In: TCC (B2). Lecture Notes in Computer Science, vol. 9986, pp. 217–238 (2016)
34. Raghav, Andola, N., Verma, K., Venkatesan, S., Verma, S.: Proactive threshold-proxy re-encryption scheme for secure data sharing on cloud. The Journal of Supercomputing pp. 1–29 (2023)
35. Microsoft SEAL (release 4.1). `https://github.com/Microsoft/SEAL` (Jan 2023), microsoft Research, Redmond, WA.
36. Standardization, H.E.: Homomorphic Encryption Standardization – An Open Industry / Government / Academic Consortium to Advance Secure Computation (2023), `https://homomorphicencryption.org/`
37. Yasuda, S., Koseki, Y., Hiromasa, R., Kawai, Y.: Multi-key homomorphic proxy re-encryption. In: ISC. Lecture Notes in Computer Science, vol. 11060, pp. 328–346. Springer (2018)