

Computing Blindfolded on Data Homomorphically Encrypted under Multiple Keys: An Extended Survey

ASMA ALOUFI* and PEIZHAO HU, Rochester Institute of Technology, New York
YONGSOO SONG and KRISTIN LAUTER, Microsoft Research, Redmond, WA

New cryptographic techniques such as homomorphic encryption (HE) allow computations to be outsourced to and evaluated blindfolded in a resourceful cloud. These computations often require private data owned by multiple participants, engaging in joint evaluation of some functions. For example, Genome-Wide Association Study (GWAS) is becoming feasible because of recent proliferation of genome sequencing technology. Due to the sensitivity of genomic data, these data should be encrypted using different keys. However, supporting computation on ciphertexts encrypted under multiple keys is a non-trivial task. In this paper, we present a comprehensive survey on different state-of-the-art cryptographic techniques and schemes that are commonly used. We review techniques and schemes including Attribute-Based Encryption (ABE), Proxy Re-Encryption (PRE), Threshold Homomorphic Encryption (ThHE), and Multi-Key Homomorphic Encryption (MKHE). We analyze them based on different system and security models, and examine their complexities. We share lessons learned and draw observations for designing better schemes with reduced overheads.

Reference Format:

Asma Aloufi, Peizhao Hu, Yongsoo Song, and Kristin Lauter. Computing Blindfolded on Data Homomorphically Encrypted under Multiple Keys: An Extended Survey. Cryptology ePrint Archive: Report 2020/xxx

1 INTRODUCTION

Modern data-driven applications involve highly sensitive data such as genome sequences [100], biometric data including iris scans and fingerprints [104], and location data and information of users' whereabouts [64]. These data are hard to change, sometime irreplaceable, once exposed. Therefore, privacy enhancing techniques have to be applied in order to ensure the privacy of user data. Conventional encryption protects data *in-transit* and *in-storage*. However, data has to be decrypted before performing any computation on it. Advanced encryption technique is needed to extend the protection of user data during computation; that is, *in-use*.

Homomorphic encryption (HE) supports arithmetic operations, such as addition and multiplication, on encrypted data without decrypting it first. Specifically, given two messages m and m' and homomorphic addition and multiplication operations \oplus and \otimes , we have $Enc(m) \oplus Enc(m')$ which decrypts to $m + m'$ and $Enc(m) \otimes Enc(m')$ which decrypts to $m \times m'$. For simplicity, we will use normal arithmetic operators to represent homomorphic operations in the rest of the paper. Generally speaking, any algorithm that can be reduced to just these arithmetic operations can be homomorphically evaluated on encrypted data. Logic gates, such as AND, OR, XOR, and NOT, can be translated into arithmetic forms; for example, $XOR(x, y) = x + y - 2xy$ if $x, y \in \mathbb{Z}$, or simply $XOR(x, y) = x + y$ if $x, y \in \mathbb{Z}_2$. Additionally, checking equality of two numbers $x, y \in \mathbb{Z}$ can be performed as $EQ(x, y) = \prod (XNOR(x_i, y_i))$ where x_i, y_i are bits of the input numbers and $XNOR(x_i, y_i) = NOT(OR(x_i, y_i))$, or simply $EQ(x, y) = x + y + 1$ if $x, y \in \mathbb{Z}_2$. The first set of HE schemes [10, 48, 60, 78] only realized *partial* homomorphisim, which supports either addition or

*Asma Aloufi is also affiliated with Taif University, Saudi Arabia.

Authors' addresses: Asma Aloufi, ama9000@rit.edu; Peizhao Hu, Peizhao.Hu@rit.edu, Rochester Institute of Technology, New York; Yongsoo Song, yongsoo.song@microsoft.com; Kristin Lauter, klauter@microsoft.com, Microsoft Research, Redmond, WA.

This paper is an early extended draft of the survey that is being submitted to ACM CSUR and has been uploaded to Cryptology ePrint Archive for feedback from stakeholders.

multiplication on ciphertexts but not both, such as Paillier [78] and ElGamal [48] schemes. Boneh-Goh-Nissim scheme [17] supports arbitrary number of additions and a single multiplication; hence, it is *somewhat* homomorphic. There are *leveled* HE schemes which support a predetermined number of multiplications based on the targeted function. In 2009, Gentry [55] proposed the first plausible construction to achieve a *fully* HE (FHE) scheme based on ideal lattices that can support arbitrary number of additions and multiplications. Homomorphic addition is performed almost free of cost but multiplication significantly increases the noise elements in the ciphertext. Hence, it is important to reduce the multiplicative depth (i.e., consecutive multiplications) of a homomorphic function. When the noise growth in the ciphertext is too large, it is not possible to retrieve the correct message after decryption. Gentry’s FHE scheme, which was built on a leveled HE scheme, can homomorphically evaluate its own decryption circuit in a *bootstrapping* step. Hence, it can refresh the encryption of an evaluated ciphertext, when the noise level reaches a defined threshold, to obtain a fresh ciphertext with small noise. In practice, the bootstrapping step is computationally expensive; therefore, most applications sufficiently use a leveled HE scheme. Also, HE schemes can be designed to be either symmetric (uses one key to encrypt and decrypt) or asymmetric (uses one key pair, a public key to encrypt and a secret key to decrypt).

1.1 System and security models for secure computation

An ideal form of secure computation is to send the sensitive data to a trusted party, who performs the computations and returns the result. However, the existence of this trusted party is not always possible especially with frequent security breaches and possibility of participants’ corruption. In a practical setting, the security of constructed protocols is emulated to the security of an ideal model, following the Ideal/Real model paradigm [69]. In other words, we focus on a security model which considers the existence of semi-honest or malicious adversaries.

The ability to compute while encrypted allows sensitive data to be outsourced for computations without compromising privacy. This ability enables many applications to be designed based on different system models, as illustrated in Fig. 1. These system models are often referred to as *cooperative* [1, 106] models (i.e., cooperation in the presence of competition). The simplest system model for secure outsourced computation is in a two-party setting (Fig. 1a) where Alice wants Bob to perform a computation $f(x, y)$, where f is a function or a trained machine learning model, on sensitive inputs x and y owned by them without revealing the data to each other. In this case, Alice encrypts the input $[x]$ under her public key and sends it to Bob, who homomorphically computes the function f on the inputs $[x], y$ and returns the encrypted result $[f(x, y)]$ to Alice. Alice can decrypt the evaluated results using her private key. In this system model, user data privacy is protected against passive adversary by encryption as long as the function f does not leak information about Bob’s input y . For some leaky functions such as summing two input numbers, HE primitives do not prevent leakage.

The two-party setting can be extended to a more general setting, secure outsourced computation among multiple parties. As illustrated in Fig. 1b, multiple parties may want to perform a joint computation on their encrypted data without revealing sensitive data to each others. This computation model is often based on interactive secure multi-party computation (MPC) protocols, which are characterized by the high communication overhead and require participation of all the parties during the computation.

There is an emerging system model which supports non-interactive computations and better suits the cloud computing model because data owners delegate computations on their data to a private or public cloud, as shown in Fig. 1c. Although there are existing work [19, 20, 31, 104] on outsourcing homomorphic computations to the cloud, these works focus on settings similar to the two-party setting; that is, the computations are performed on data from one data owner. In this paper, we focus

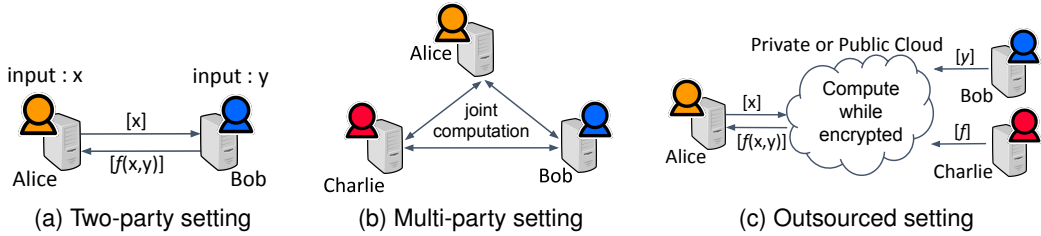


Fig. 1. System models for secure computation.

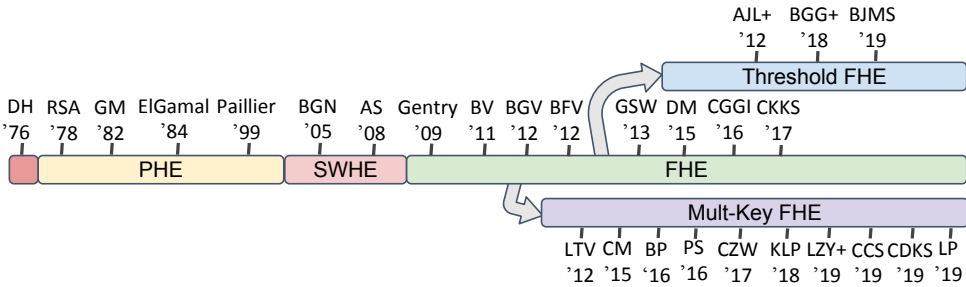


Fig. 2. Timeline featuring recent developments in HE schemes.

on a more challenging computation model in which multiple parties delegate their sensitive data or functions to the cloud. The cloud is an evaluator of some functions and should not learn anything from the encrypted data. Similar to previous system models, data owners do not want to reveal their data to others. To ensure user data privacy, encrypting data using a single key is not very practical in many application scenarios. *Can we realize a secure computation model in which data are encrypted using different keys of the corresponding owners?* This cloud-friendly computation model has been discussed in previous literature [77, 91]. However, it has not been realized due to the challenges in supporting homomorphic evaluation on data encrypted under different keys, in addition to issues in key management and lack of efficient decryption protocols. Note, we focus on the system model in Fig. 1c and assume inputs are encrypted under their owners’ keys, but we do not define under which key the result $f(x, y)$ is encrypted for now. We will be specific about this key in the subsequent sections when we review different approaches.

To the best of our knowledge, none of existing surveys provide a comprehensive review on how to support efficient homomorphic computations on encrypted data under multiple keys in the new computation model we discussed earlier. In this survey, our goal is to fill this gap and provide a comprehensive review, analysis, and lesson-learned of the state-of-the-art multi-key homomorphic encryption techniques for secure computation outsourcing.

1.2 Computing with different keys

Spanning over the last 40 years, there are many homomorphic encryption schemes, each provides different supports for performing arithmetic operations on homomorphically encrypted data. Figure 2 shows a timeline of different types of HE schemes (improved based on [3, 102]), including recent schemes that support homomorphic computations on encrypted data under multiple keys.

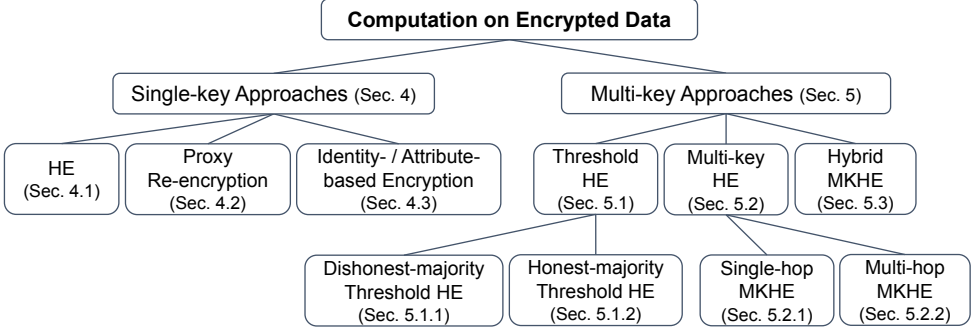


Fig. 3. Taxonomy of techniques for secure computation on encrypted data.

In this paper, we review and categorize these techniques according to how the multiple keys are used to secure data and how homomorphic computations are supported. Figure 3 shows a taxonomy of these surveyed techniques within this paper. Generally speaking, they can be roughly divided into two categories:

- A subset of these techniques is based on a *single* key, where ciphertexts are encrypted under a key owned by a single party at all times and decrypted using the corresponding secret key or can be made decryptable under another key. Most existing works on HE assumed data encrypted under a single key. However, it is clearly impractical in our *cooperative* system models illustrated in Fig. 1. Another example of single key schemes but offering additional functionalities is Proxy Re-Encryption (PRE) technique [12, 103] that transforms a stored ciphertext $c_i = \text{Enc}(pk_i, m)$ into a ciphertext $c_j = \text{ReEncrypt}(rk_{pk_i \rightarrow pk_j}, c_i)$ that is decrypted by authorized user's secret key sk_j . Also, there are Identity-based encryption (IBE) [15] and Attribute-based encryption (ABE) [86] schemes that provide access control mechanisms where data can be decrypted by authorized keys generated based on the user's identity or attributes.
- On the other hand, *multiple* keys may be involved in the encryption of ciphertexts and required to contribute to its decryption. Threshold HE (ThHE) [8] and Multi-key HE (MKHE) are examples of these techniques. An example of the latter allows the ciphertext c_i , encrypted under pk_i , to be extended to an additional key pk_j such that $\bar{c} = \text{Enc}(\{pk_i, pk_j\}, m)$. The message can be retrieved when using the two corresponding secret keys $m = \text{Dec}(\{sk_i, sk_j\}, \bar{c})$. Recently, there is a hybrid approach [4] which combines the advantages of both threshold and multi-key HE with a goal to reduce computation complexity and ciphertext size.

1.3 Related work

As discussed before, many surveys focused on secure outsourced computation techniques [91, 97, 102] and the construction of HE schemes [3, 52, 73, 98]. There is a notable lack in the literature for detailed review on secure outsourced computation with different keys in the *cooperative* system model. Tang et al. [97] focused on threat modeling for secure outsourced computation and studied techniques to realize security requirements including confidentiality, integrity, privacy, and access control. Techniques with different keys were studied only for data sharing in the cloud and did not address supporting homomorphic computations on encrypted data. Secure outsourcing of fundamental and application-specific homomorphic computations based on a single key were surveyed in [91, 102].

On the other hand, extensive reviews of the HE schemes construction have been provided both from engineering [73] and theoretical [3, 53] perspectives. Unfortunately, existing survey articles

did not discuss extended HE schemes, such as ThHE and MKHE, which support computing with multiple keys. Particularly, these multi-key techniques are witnessing rapid development in recent years as can be observed in Fig. 2. Some proposed MKHE work [4, 30, 66] give brief analysis of related work. A more-in-depth review [18] studied the first two MKHE constructions (i.e., LTV'12, CM'15). Overall, no existing survey provides detailed analysis and categorization of state-of-the-art multi-key approaches, which is a primary aim of our survey.

1.4 Contributions and organization

In this paper, we fill the gap in the literature and conduct a comprehensive survey on cryptographic techniques that enable multiple parties to compute on their data encrypted under multiple keys. In particular, we investigate design trends in the state-of-the-art schemes for different system and threat models. We share lesson learned and discuss new directions that is yet to be explored to achieve more practical solutions for secure computation outsourcing. We discuss potential applications that can benefit from the ability of homomorphically computing on encrypted data under multiple keys.

The rest of this survey is organized as follows. Different system models and security models for secure computation is defined in Section 2. After that, preliminaries and HE commonly used techniques are presented in Section 3. Single-key approaches such as attribute-based encryption and proxy re-encryption are presented in Section 4. Following that, the multi-key approaches, threshold HE, multi-key HE, and hybrid approaches are discussed in Section 5. In Section 6, we share lesson learned and open research directions. In Section 7, we review application scenarios for computation on data with multiple keys. Finally, Section 8 concludes the survey.

2 SECURITY CONSIDERATIONS

To design a secure protocol, we need to determine possible attacks that target a system model. Adversaries who launch the attacks often aim to compromise security requirements such as confidentiality, integrity, and availability. It is essential to understand these threats and their impacts on the security during threat modeling. In often cases, malicious adversaries can launch arbitrary attacks that deviates from the protocol, and it is difficult to protect against every threat. Moreover, some countermeasures may be computationally-intensive and affect the practicality of the protocol. Hence, we make security assumptions as a trade-off for efficiency. For example, we assume system users are semi-honest, i.e., they strictly follow the protocol specifications. Careful design of the security model is critical to achieving secure yet practical protocol for a given system model. In the rest of this section, we discuss the security model of a system in more details, including modeling *potential threats* and specifying the appropriate *security requirements* and *assumptions*.

2.1 Threat model

Threat modeling is the process of identifying potential vulnerabilities, circumstances, and actions in which a capable adversary can compromise the security of the system [95]. In the system models illustrated in Fig. 1, an adversary can be either an *internal* or *external*. An internal adversary is a participant in the protocol, e.g., Bob in the multi-party setting in Fig. 1b, and may want to learn confidential information of other users. On the other hand, an external adversary may intercepts or corrupts system users, such as colluding with the cloud evaluator, with the intention to breach the privacy of other users. Moreover, adversaries are assumed to be either *semi-honest*, or *malicious*.

A passive adversary in the *semi-honest* setting, say the cloud evaluator, follows the protocol execution and does not attempt to cheat; however, it passively collects the transmitted inputs and tries to infer useful information about the data owners. It is difficult to detect this type of passive attacks because no abnormal behaviour is observed [87]. Hence, it is fundamental that any secure protocol has to provide defense, rather than detection, against such passive adversary attacks.

Unlike passive adversaries, active or *malicious* adversary can launch arbitrary attacks such as deviating from the protocol’s specifications. A malicious cloud evaluator may use homomorphic properties to alter ciphertexts without decryption, or simply corrupt them. Active attacks may also take other forms to breach security, such as corrupting and impersonating authorized system users and feeding new or changed inputs to the protocol, or interfering with the communication channel by delaying and replaying messages. A system model is proven to be secure if it applies countermeasures against any possible attack defined in the threat model and launched by any corrupted user. But, it is often computationally expensive to account for all possible attacks.

2.2 Security requirements

Security requirements include the confidentiality and privacy of system inputs, the integrity of data and correctness of the evaluation, and the availability of data for authorized users. In threat modeling, attacks are often categorized based on which requirement they target. Subsequently, cryptographic primitives are applied to meet those requirements. Data confidentiality can be ensured by encryption, especially with those that offer semantic security, i.e., the ciphertext does not leak information about its plaintext. Data integrity can be checked by obviously verifying data inputs. Suppose a classification protocol that requires users’ inputs to be in the range $\{0, 1\}$, and maliciously providing inputs not in this range may leak information about the evaluation model [101]. To ensure integrity, each user must send additional information, in the form of Zero-Knowledge proofs, with their encrypted inputs to prove that the inputs are indeed valid without revealing them. Similarly, correctness of evaluation can be checked to prevent corrupted evaluator from applying unwanted functions. For example, the evaluation of a function can be represented as a graph with each computation as a node associated with a hash value. The correctness can be checked by running a proof, using zkSNARKs techniques [9], with the hash values generated during the evaluation.

Another important requirement for security is key management. The life cycle of cryptographic keys includes *generation, distribution, storage, use, and revocation*. It is vital to carefully manage keys through each of these phases because poor key management can easily defeats the purpose of using cryptography [50]. In key generation, cryptographic parameters such as key bit-length must be chosen appropriately based on a security parameter λ to protect against known attacks such as brute-force attack. Generated secret keys must be kept private by their rightful owners. Owners may provide information about their secret keys in a form of evaluation keys to perform homomorphic techniques like key switching or bootstrapping (discussed in Sec. 3.3). Those evaluation keys must be securely distributed and stored under encryption. The keys, or key pairs, should also be properly revoked when expired or changed across the system. Throughout this survey, we will focus on the case of asymmetric cryptosystems that generates key pairs, but the key management also applies to symmetric cryptosystems.

2.3 Security assumptions

Secure protocols must take countermeasures against possible attacks. As mentioned, modeling threats and defining security requirements are two essential steps. However, designing for a practical setting often faces a trade-off between security and efficiency. Therefore, it is common to make security assumptions to achieve a more efficient design. For example, many HE schemes base their security on the Learning with Errors (LWE) problem (formally defined in Sec. 3.1). But studies show it is more efficient to operate in a ring of polynomials instead of matrices, which enables Single Instruction Multiple Data (SIMD) operations. Many recent HE schemes that show practical performance have their ring variant constructed under the ring-LWE assumption. Another example is in the distributed decryption process of a Threshold HE scheme [8]. In this scheme, N participants create a joint public key, such that $pk^* = (pk_1 + \dots + pk_N)$, and use it for encryption. The corresponding secret key is

Table 1. Description of notations used throughout the survey.

Category	Notation	Description
General	\mathbf{a}	A vector of n elements where $a_i = \mathbf{a}[i]$ is the i -th element.
	\mathbf{B}	A matrix of elements where $b_{i,j}$ is the j -th element of the i -th row.
	$ \mathbf{a} $	The l_1 -norm of vector, $ \mathbf{a} = \sum_{i=0}^n a_i$.
	$\langle \mathbf{a}, \mathbf{b} \rangle$	The dot-product of the two vectors \mathbf{a}, \mathbf{b} as $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^n (a_i b_i)$.
	N	Total number of users in the system.
(R)LWE-specific	$\Phi(x)$	A cyclotomic polynomial $\Phi(x) = x^d + 1$, where d is a power of 2.
	R	A ring over polynomials with integer coefficients $R = \mathbb{Z}[x]/(\Phi(x))$.
	R_t	The ring of polynomials with coefficients in \mathbb{Z}_t , where t is plaintext modulus.
	R_q	The ring of polynomials with coefficients in \mathbb{Z}_q , where q is ciphertext modulus.
	ψ	The key distribution over R .
	χ	The noise distribution over R or \mathbb{Z} with small standard deviation.
HE Scheme-specific	pk_i	HE public key of the i -th user.
	sk_i	HE secret key of the i -th user.
	ek_i	HE evaluation key of the i -th user.
	$[m]_i$	HE encryption of a message m under pk_i . Also denoted as c

secretly shared among the N participants. Thus, users must jointly construct the secret key in order to decrypt. There are two assumptions that are considered for this case. First, we may assume the presence of at most $N - 1$ corrupted users. Similar to (N, N) secret sharing schemes [89], this means *all* N users are required to participate in the decryption; otherwise, the ciphertext will not be correctly decrypted. This design is based on the dishonest-majority assumption. The second assumption is the honest-majority assumption, which relaxes the security to achieve efficiency. It allows a subset $T < N$ of the participants to reconstruct the corresponding secret key and collaboratively decrypt the ciphertext, i.e., (T, N) scheme.

3 PRELIMINARIES

In this section, we define notations and definitions that will be used throughout this survey and provide background on general homomorphic primitives and techniques used in HE schemes.

3.1 Notations and definitions

We denote vectors as bold lowercase, such as \mathbf{a} , and matrices as bold uppercase \mathbf{B} . Specifically, given a vector $\mathbf{a} = (a_1, \dots, a_n)$, we define $\mathbf{a}[i] = a_i$ as the i -th element. Let $n \times n'$ be the matrix dimensions and $\mathbf{B}[i, j]$ be the j -th element of the i -th row. The dot product of the two vectors \mathbf{a}, \mathbf{b} is denoted by $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^n \mathbf{a}[i] \cdot \mathbf{b}[i]$, and the tensor product is denoted by $\mathbf{a} \otimes \mathbf{b}$. The multiplication of two elements a, b is denoted by ab or $a \cdot b$. For an element $a \in \mathbb{R}$, $\lfloor a \rfloor$ denotes the rounding to its nearest integer, and $\lceil a \rceil, \lfloor a \rfloor$ denote the rounding up and down, respectively. For a symmetric (asymmetric) encryption scheme, we denote the secret key and the public and private key pair as sk and (pk, sk) , respectively. For scheme-specific notations, we will introduce them when needed in the subsequent sections.

DEFINITION 1 (LEARNING WITH ERRORS (LWE) [85]). *For a security parameter λ , let $n = n(\lambda)$ be a dimension, $q = q(\lambda) \geq 2$ be an integer, and $\chi = \chi(\lambda)$ be a distribution over \mathbb{Z} . Sample a vector $\mathbf{a}_i \leftarrow \mathbb{Z}_q^n$, $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, and $e_i \leftarrow \chi$. The LWE problem is to distinguish the pair $(\mathbf{a}_i, b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i) \in \mathbb{Z}_q^{n+1}$ from a uniformly sampled (\mathbf{a}_i, b'_i) from \mathbb{Z}_q^{n+1} . The LWE assumption is that LWE problem is computationally infeasible.*

The LWE problem operates in the n -dimensional integer space \mathbb{Z}_q^n and consists of vectors and matrices. To achieve more efficient computations, the problem can be extended to rings of polynomials with integer coefficient. The aim is significantly reduce the dimension, such that $n = 1$, and correspondingly the size of keys and ciphertexts.

DEFINITION 2 (RING LEARNING WITH ERRORS (RLWE) [72]). For a security parameter λ , let $\Phi(x) = x^d + 1$ be a cyclotomic polynomial where $d = d(\lambda)$ is a power of 2, and $q = q(\lambda) \geq 2$ be an integer. Define the ring R over polynomials with integer coefficients $R = \mathbb{Z}[x]/(\Phi(x))$. Let $\chi = \chi(\lambda)$ be a discrete Gaussian distribution over R and bounded by $\mathcal{B} = \mathcal{B}(\lambda)$ such that $\mathcal{B} \ll q$. Let a and s be uniformly sampled elements from R_q , and let $e \leftarrow \chi$ be a sampled error term. The RLWE problem is to distinguish the pair of $(a_i, b_i = a_i s + e)$ from any uniformly sampled pair $(a_i, b'_i) \leftarrow R_q^2$. The RLWE assumption is that the RLWE problem is computationally infeasible.

An amortized version of the RLWE problem [7, 72] shows that it is equivalent to sampling s from a small distribution ψ instead of the ring R_q . This yields a smaller secret key in an RLWE-based cryptosystem, e.g., BGV SWHE scheme [22]

DEFINITION 3 (COMMON REFERENCE STRING (CRS)). For some distribution D , the CRS model starts in a trusted setup with sampling a value such that $d \leftarrow D$. This value is a common reference string [26, 40] (or a public parameter [51, 79]) that is made available to all participants before any computation starts.

In (R)LWE, the CRS is a uniformly sampled (ring) element $a \leftarrow R$ which is provided as a public parameter. When computing with multiple keys, it is a requirement for those keys to be related in a manner to ensure correct computation and decryption. Hence, schemes are often designed in the CRS model where participants are given access to this public parameter to generate their individual public keys. More details on this model is discussed in Sec. 5.

3.2 General homomorphic primitives

Homomorphic encryption (HE) is a class of encryption schemes that support computations such as addition and multiplication on encrypted data. Existing HE schemes can be divided into three main types based on the homomorphic operations supported by the evaluation function. In *Partial HE* (PHE) schemes, the evaluation function supports either addition (i.e. additive homomorphism), such as Goldwasser-Micali [60] and Paillier cryptosystem [78], or multiplication (i.e. multiplicative homomorphism), such as ElGamal cryptosystem [48], but *not both*. In contrast, *Fully HE* (FHE) allows arbitrary number of additions and multiplications. *Somewhat HE* (SWHE) schemes support *both* addition and multiplication on the ciphertexts. Yet, the number of multiplications allowed is limited due to the inherited construction of the scheme where ciphertexts contain noise that exponentially scales with multiplications. In general, HE scheme is a tuple of probabilistic polynomial-time (PPT) algorithms $HE = (\text{KeyGen}, \text{Enc}, \text{Eval}, \text{Dec})$. We define each algorithm as follow.

- $\text{HE.KeyGen}(1^\lambda) \rightarrow (\text{pk}, \text{sk})$: Given a security parameter λ determining the security level, the key generation algorithm outputs a public key pk , a private key sk .
- $\text{HE.Enc}(\text{pk}, m) \rightarrow c$: Given a public key pk and a message m , the encryption algorithm outputs a ciphertext c .
- $\text{HE.Eval}(\text{pk}, f, c, c') \rightarrow c_{\text{eval}}$: Given a public key pk , two ciphertexts c, c' , and a homomorphic function f , the evaluation algorithm outputs the evaluated ciphertext $c_{\text{eval}} = f(c, c')$.
- $\text{HE.Dec}(\text{sk}, c) \rightarrow m$: Given a ciphertext c encrypted under pk and the corresponding secret key sk , the decryption algorithm outputs the message m .

Note, the HE.Eval algorithm homomorphically performs a defined function f on the ciphertexts. This function is constructed using HE.Add and HE.Mult which are homomorphic addition and

multiplication respectively. In this paper, we will focus on HE schemes that are based on the LWE problem and its ring variant, RLWE.

3.3 Common techniques in HE

3.3.1 Key Switching. In many of the (R)LWE-based HE schemes, the initial output of homomorphically multiplying two ciphertexts is a longer ciphertext encrypted under a new secret key element s^2 . For instance, observe in the BGV scheme [22] presented in Scheme 1, the product between two ciphertexts c, c' is

$$\begin{aligned}\tilde{c}_{\text{mult}} &= c \cdot c' = (c_0, c_1) \cdot (c'_0, c'_1) \\ &= (c_0 \cdot c'_0, c_0 \cdot c'_1 + c'_0 \cdot c_1, c_1 \cdot c'_1) \\ &= (\tilde{c}_0, \tilde{c}_1, \tilde{c}_2) \in R_q^3\end{aligned}$$

The additional component c_2 corresponds to the quadratic element s^2 resulted from the multiplication $(c_0 + c_1 \cdot s)(c'_0 + c'_1 \cdot s) = \tilde{c}_0 + \tilde{c}_1 \cdot s + \tilde{c}_2 \cdot s^2$. This new ciphertext \tilde{c}_{mult} is not decryptable by the secret key s . Therefore, HE schemes employ *key switching* [22] as a transformation technique to reduce the dimension after each homomorphic multiplication. This transformation is accomplished with the aid of auxiliary information provided as evaluation key ek which encrypts s^2 under s . The following *gadget toolkit* is needed to perform the key switching operation:

- Gadget vector: $\mathbf{g} = (g_0, \dots, g_{\ell-1}) \in R^\ell$
- $\text{Decomp}(x)$: Given an element $x \in R_q$, decompose it into a *short* vector $\mathbf{u} = (u_0, \dots, u_{\ell-1}) \in R^\ell$ such that $\langle \mathbf{u}, \mathbf{g} \rangle = x \pmod{q}$.

The decomposition function is (informally) denoted by $\text{Decomp} = \mathbf{g}^{-1}$ to remark that $\langle \mathbf{g}^{-1}(x), \mathbf{g} \rangle = x \pmod{q}$ for all x . There have been proposed several gadget toolkits in the literature, for example, $(1, 2, \dots, 2^{\lceil \log q \rceil - 1})$ is a gadget vector corresponding to the bit decomposition $\text{BitDecomp} : x = \sum_{0 \leq i < \lceil \log q \rceil} 2^i u_i \mapsto (u_i)_{0 \leq i < \lceil \log q \rceil}$.

As discussed above, key switching is a commonly used building block of LWE-based HE schemes (e.g. BGV in Scheme 1) which reduces the dimension of a long ciphertext after homomorphic multiplication. Here, we review how to generate an evaluation key ek and perform the key switching operation on a three-dimensional ciphertext \tilde{c} encrypted under $(1, s, s^2)$.

- $\text{EvalKeyGen}(s)$: The evaluation key is a special encryption of s^2 under s . The evaluation key $\text{ek} = (\hat{\mathbf{b}}, \hat{\mathbf{a}}) \in R_q^{\ell \times 2}$ is generated during the scheme. It is generated by $\hat{\mathbf{a}}[i] \leftarrow R_q$, $\hat{\mathbf{e}}[i] \leftarrow \chi$, and $\hat{\mathbf{b}}[i] = -\hat{\mathbf{a}}[i] \cdot s + \hat{\mathbf{e}}[i] + \mathbf{g}[i] \cdot s^2 \pmod{q}$ for $0 \leq i < \ell$.
- $\text{KeySwitch}(\text{ek}, \tilde{c})$: Given an evaluated ciphertext $\tilde{c} = (\tilde{c}_0, \tilde{c}_1, \tilde{c}_2)$ encrypted under the secret element s^2 , and the evaluation key $\text{ek} = (\hat{\mathbf{b}}, \hat{\mathbf{a}})$, compute $\mathbf{g}^{-1}(\tilde{c}_2) = (u_0, \dots, u_{d-1})$. Then, return the ciphertext $(c_0, c_1) = (\tilde{c}_0, \tilde{c}_1) + \sum_{0 \leq i < \ell} u_i \cdot (\hat{\mathbf{b}}[i], \hat{\mathbf{a}}[i]) \pmod{q}$.

This technique can be generalized and used for other purposes beside dimension reduction. For example, it can be employed in proxy re-encryption (see Sec. 4.2) to transform a ciphertext from one encrypting a message m under one key s_1 to one encrypting the same message m under a different key s_2 . It can also be used to facilitate the *bootstrapping* step [25, 55], which accomplishes fully HE scheme from a leveled SWHE scheme.

3.3.2 Bootstrapping. Following the blueprint proposed by Gentry [55], one can construct a fully HE scheme from a somewhat HE scheme. Mainly, when a ciphertext reaches the maximum defined level, a *bootstrapping* technique is applied which *reencrypts* the ciphertext by homomorphically evaluating the decryption circuit and outputting a fresh ciphertext with minimum noise. Given a ciphertext at the maximum level c and the corresponding secret key $\text{sk} = s$, encrypt them both under

the same key pk and obtain a fresh ciphertext by performing $\check{c} = \text{Bootstrap}(\text{Enc}(pk, c), \text{Enc}(pk, sk)) = \text{Enc}(pk, \text{Dec}(sk, c))$. Generally, this function has to be of a limited depth to be performed with the somewhat HE scheme. This means the degree of the decryption polynomial must be low for the ciphertext to be bootstrapable [55]. Integer-based schemes, such as Gentry’s scheme [55], require a squashing technique which additionally makes a sparse subset-sum assumption to decrease the degree of the decryption polynomial. Recent LWE-based HE schemes, such as the BV scheme [24], uses relinearization to reduce the ciphertext dimension after each multiplication. As a result, the decryption polynomial has a low degree and does not need squashing.

More technically, the two main algorithms from key switching $\text{BitDecomp}(\cdot, q)$ and $\text{PowersOf2}(\cdot, q)$ can be used to obtain the bootstrapped ciphertext. Let the secret key be decomposed to its binary representation $\text{BitDecomp}(s, q) = (u_0, u_1, \dots, u_{\lfloor \log q \rfloor})$. Compute the powers of base two for the ciphertext element c_0 such that $\text{PowersOf2}(c_0, q) = (2^0 c_0, 2^1 c_0, \dots, 2^{\lfloor \log q \rfloor} c_0)$. The decryption algorithm is then homomorphically evaluated under the encryption of the key pk such that $c_1 - \sum_{i=0}^{\lfloor \log q \rfloor} u_i 2^i c_0 = c_1 - c_0 s = m \bmod q \pmod{t}$. This technique can be repeatedly applied to support arbitrary number of homomorphic computations, but it is computationally intensive. However, recent works in the literature show that bootstrapping can be done in less than 1 second [47] and can be made even faster to less than 0.1 second [36] using different cryptographic constructions.

3.3.3 Distributed decryption. In many HE protocols based on the multi-party computation (MPC) protocol, the involved parties may be required to help decrypting the final evaluation result. For example, in threshold encryption (discussed in Sec. 5.1), the final result is encrypted under the combination of all the parties’ keys; hence, each party must participate to partially decrypt the result with their own secret key. This process becomes a *distributed decryption protocol* since the decryption now is executed as an MPC protocol. Two algorithms are performed within this protocol. First, a partial decryption that is performed by each party who locally decrypts the result with their own secret key and shares the output. Then, a final decryption is performed by the designated party who aggregates the shared partial decryptions to obtain the final decrypted result.

Two main security issues has to be addressed for this decryption protocol. First, the shared partially decrypted ciphertext may leak information on the party’s secret key; hence, additional security measures, such as *noise smudging* (or noise flooding), are considered to prevent this leakage. To make sure that no secret shares can be learned, we need to add larger errors following the Smudging Lemma [8, 76], which states that adding a large noise “smudges out” the small values in the ciphertext. Hence, adding a large noise to the decryption component prevents leaking information about the secret share.

LEMMA 1 (SMUDGING NOISE [8]). *Let $B_1 = B_1(\lambda)$ and $B_2 = B_2(\lambda)$ be two positive integers and let $e_0 \in [-B_1, B_1]$ be a fixed integer. Let $e_1 \leftarrow [-B_2, B_2]$ be chosen uniformly at random. Then the distribution of e_1 is statistically indistinguishable from that of $e_1 + e_0$ as long as $B_1/B_2 = \epsilon$, where $\epsilon = \epsilon(\lambda)$ is a negligible function.*

The second security issues is related to the final decryption algorithm, which may leak the encrypted result. Specifically, an attacker who falsely acquired the shared partial decryptions can perform the final decryption by combining the shares and retrieve the final result.

3.3.4 Noise reduction techniques. As a result of homomorphic computations, the embedded noise in ciphertexts increases in magnitude. For a noise magnitude \mathcal{B} , each homomorphic addition doubles the noise as $2\mathcal{B}$, and each homomorphic multiplication squares it as \mathcal{B}^2 . However, in BFV scheme (Sec. 4.1.2) the noise growth is dominated by $I_1\mathcal{B} + I_2\mathcal{B}$ where $I_i = \Delta m_i - m_i$ is a factor introduced when we scale up the messages m_i with Δ . To ensure correct decryption of the evaluated

Table 2. Notations specific to single-key approaches.

Category	Notation	Description
HE-specific	pp	HE scheme's public parameters including security parameter λ and circuit depth L .
	Δ	A scaling factor used in scale-invariant or rescaling techniques, $\Delta = \lfloor q/t \rfloor$
	ℓ	The bit length, $\ell = 0, \dots, \lceil \log q \rceil$.
	G	The diagonal gadget matrix constructed as $G = I_n \otimes g$, where $g = (1, 2, 4, \dots, 2^{\ell})$.
	$G^{-1}(\cdot)$	The bit-decomposition function (inverse gadget), such that $GG^{-1}(a) = a$.
PRE-specific	$\text{rk}_{A \rightarrow B}$	The re-encryption key switching an encryption from under Alice's key to Bob's key.
IBE/ABE-specific	\mathcal{U}	The universe set of attributes/identities.
	ω'	A subset of attributes $\omega' \subset \mathcal{U}$ chosen for encrypting a ciphertext.
	ω	The set of attributes provided for decrypting a ciphertext.

result, the noise must not exceed the $\frac{q}{2}$ range. We briefly discuss some of the noise reduction techniques used in HE schemes and refer the readers to [21, 22, 35] for more details.

Modulus switching. Modulus switching is a technique proposed by Brakerski *et al.* [22] to scale down the noise after each multiplication. For a L -depth circuit (i.e., requires at most L multiplications), define $L + 1$ moduli $\{q_L, q_{L-1}, \dots, q_0\}$. After the i -th multiplication, we can transform a ciphertext $c \bmod q_i$ into the ciphertext $c' \bmod q_{i-1}$, where $q_{i-1} < q_i$, such that the noise scales down approximately by a factor of $\frac{q_{i-1}}{q_i}$. This brings the noise magnitude back to \mathcal{B} and changes to a smaller modulus. This is proven to increase the number of supported homomorphic multiplications before the need for bootstrapping.

Scale-invariant. A following work of Brakerski [21] proposed the scale-invariant technique which uses *one* modulus q to homomorphically evaluate L -depth circuit. This technique completely removes the need for modulus switching, which requires choosing $L + 1$ decreasing moduli. In a nutshell, a message is scaled at encryption by a factor of $\Delta = \lfloor q/t \rfloor$, where q is significantly larger than t ; hence, the message is scaled up. After each homomorphic multiplication and at decryption, the ciphertext is scaled down by a factor of (q/t) , resulting in a significant decrease in the noise magnitude. The origin technique was proposed for LWE-based schemes with $t = 2$, and was later ported [49] to the RLWE assumption for efficiency.

These noise reduction techniques are applied in the base HE schemes discussed in Sec. 4 and can be readily extended to the multi-key (Sec. 5) setting.

4 SINGLE-KEY APPROACHES

Many existing work on homomorphic encryption assume data is always encrypted under one key in symmetric schemes or one key pair in asymmetric schemes. This key or key pair is typically owned by a user or an organization. However, in some scenarios such as in the multi-party and outsourced system models (in Figs. 1b and 1c), data owners may want to reveal the homomorphically computed result to data consumers who should not be able to decrypt any input data. To accomplish this, additional step may be performed to transform the encrypted result to an encryption under a different key through proxy re-encryption (PRE). In other scenarios, we may want to generate decryption keys for a specific group of authorized users. In this case, we can construct protocols based on identity-based encryption (IBE) or attribute-based encryption (ABE). In this section, we review these single-key approaches and discuss their robustness in supporting secure computation. For simplicity, we list descriptions of the notations used throughout this section in Table 2.

4.1 Homomorphic encryption

Many cloud-based applications can leverage homomorphic encryption to support secure computation outsourcing without compromising the privacy of user data. In majority of proposed HE protocols for these applications [77, 91, 102, 104], data used in computation is encrypted under the same key. The primary focus in these works is on the construction of efficient homomorphic algorithms to evaluate a targeted function in the encrypted domain. Applying these protocols in the outsourced system model (Fig. 1c), where privacy of individual data owners is important, requires making security assumptions such as semi-honest cloud and non-collusion between the cloud evaluator and participating parties. These assumptions obviously weaken the security of the protocol, especially in malicious settings where corruption or collusion compromises data privacy of other data owners. Ideally, users should keep private data protected under their own keys. However, supporting multi-key HE is complex and inefficient today. In this survey, we start with reviewing the construction of some well-known base HE schemes. Then, we review the techniques applied to support homomorphic computations on data encrypted under multiple keys in later sections.

4.1.1 The Brakerski-Gentry-Vaikuntanathan (BGV) scheme. Brakerski et al. [22], proposed an efficient *leveled* HE scheme to allow arbitrary number of additions but *limited* consecutive multiplications determined according to the depth of the evaluated circuit.

The BGV scheme is based on the original Brakerski and Vaikuntanathan's scheme [24], which is the base of the second generation of HE schemes that improve the efficiency after Gentry's breakthrough [55]. The BV scheme was the first scheme basing its security solely on the hardness of standard LWE assumption, which has proven to be as hard as solving the shortest vector problem in lattices [71]. With this reduction, the scheme removes the need for making additional strong assumptions, such as the secret subset sum assumption, and avoids the squashing technique in bootstrapping. The main construction starts with a SHWE scheme of depth L , which also considers the depth of its own decryption circuit, then converts to a fully HE scheme through bootstrapping. The original LWE-based BV scheme encrypts a message bit $m \in \{0, 1\}$ with the secret $s \in \mathbb{Z}_q^n$ as $c = (b = -\langle a, s \rangle + m + 2e, a) \in \mathbb{Z}_q^{n+1}$, where $a \in \mathbb{Z}_q^n$ is a random vector and e is a small even noise. Essentially, we observe the pattern of masking the message with a large element and some noise to hold the LWE assumption. To decrypt, we use the secret s and the provided vector a in the ciphertext to compute $b + \langle a, s \rangle \pmod{q}$, which effectively removes the mask and obtains $m + 2e$. The message is then retrieved by removing the even noise by computing $m + 2e \pmod{2}$. In subsequent BV variants, such as the BGV scheme [22], the message can be an integer in \mathbb{Z}_t , where t is a chosen plaintext module that is significantly smaller than q . The noise is scaled at-most by t .

The BGV scheme can be instantiated based on LWE or its ring variant RLWE, which operates on a ring or polynomials and is proven to be more efficient. It also applies optimizations such as the Smart-Vercauteren [94] batching technique, which packs multiple plaintext messages into one ciphertext so that computations can be performed in a SIMD manner. For a circuit depth L and public parameters $pp = (\lambda, d, q, t)$, we summarize the ring variant of the BGV scheme in Scheme 1.

Additive homomorphism is straightforward in BV-type schemes, but multiplicative homomorphism is more complicated. Multiplying two ciphertexts c, c' results in a higher dimensional ciphertext that is now encrypted under s^2 rather than s . Additionally, the noise within the ciphertext grows significantly (i.e., e^2). Without control, the noise can grow exponentially with respect to the number of multiplications. Hence, two new techniques were proposed, namely *relinearization* and *modulus switching*, to address these two issues. We briefly explain here how relinearization works and defer the modulus switching to the next subsection. The core technique used in relinearization is key switching (described in Sec. 3.3.1), which will transform the encryption from s^2 back into s without decryption. Given the ciphertext product $\hat{c}_{\text{mult}} = (\hat{c}_0, \hat{c}_1, \hat{c}_2) \in R_q^3$, where $\hat{c}_0 = c_0c'_0$, $\hat{c}_1 = (c_0c'_1 + c'_0c_1)$,

Scheme 1: The Brakerski-Gentry-Vaikuntanathan (BGV) scheme [22]

- BGV.Setup($1^\lambda, 1^L$): Given the security parameter λ and a multiplicative depth L , choose a cyclotomic polynomial $\Phi(x) = x^d + 1$, where d is a power of 2. Define $R = \mathbb{Z}[x]/(\Phi(x))$ as a polynomial ring of degree d with integer coefficients. Generate the error and key distributions χ and ψ over R , respectively. Choose the ciphertext modulus q and the plaintext modulus t . Finally, output $\text{pp} = (R, \chi, \psi, q, t)$ as the public parameter. We assume all following algorithms implicitly take pp as an input.
- BGV.KeyGen(pp): Given the scheme's public parameters pp , sample a small element $s \leftarrow \psi$ and a small noise $e \leftarrow \chi$. Also uniformly sample $a \leftarrow R_q$. Set the secret key as $\text{sk} = s$ and the public key as $\text{pk} = (b, a) \in R_q^2$ where $b = -as + te \pmod{q}$.
- BGV.Enc(pk, m): Given a plaintext message $m \in R_t$, a public key $\text{pk} = (b, a)$, uniformly sample a random $r \leftarrow \psi$ and errors $e_0, e_1 \leftarrow \chi$, and encrypt the message m as $\mathbf{c} = (c_0, c_1) \in R_q^2$, where $c_0 = rb + m + te_0$ and $c_1 = ra + te_1$.
- BGV.Dec(sk, \mathbf{c}): Given a ciphertext $\mathbf{c} = (c_0, c_1) \in R_q^2$ and the secret key $\text{sk} = s$, set $\mathbf{s} = (1, s) \in R_q^2$ and decrypt by computing $m = \langle \mathbf{c}, \mathbf{s} \rangle \pmod{q} \pmod{t} \in R_t$.
- BGV.Add(\mathbf{c}, \mathbf{c}'): Adding two ciphertexts $\mathbf{c} = (c_0, c_1)$, $\mathbf{c}' = (c'_0, c'_1)$ results in $\mathbf{c}_{\text{add}} = (c_0 + c'_0, c_1 + c'_1) \in R_q^2$.
- BGV.Mult(\mathbf{c}, \mathbf{c}'): Given two ciphertexts $\mathbf{c}, \mathbf{c}' \in R_q^2$, their homomorphic multiplication yields first an extended ciphertext $\tilde{\mathbf{c}}_{\text{mult}} = (\hat{c}_0, \hat{c}_1, \hat{c}_2)$ that is encrypted under the element s^2 . A special evaluation key must be provided by $\text{ek} = (\hat{\mathbf{b}}, \hat{\mathbf{a}}) \leftarrow \text{EvalKeyGen}(s^2, s) \in R_q^{\ell \times 2}$ to perform relinearization as follows:
 - Apply gadget decomposition $\mathbf{g}^{-1}(c_2) = (u_0, \dots, u_{\ell-1})$ as described in Sec. 3.3.1.
 - Output the new relinearized ciphertext as $\mathbf{c}_{\text{mult}} = (\hat{c}_0, \hat{c}_1) + \sum_i u_i \cdot (\hat{\mathbf{b}}[i], \hat{\mathbf{a}}[i]) \in R_q^2$.

and $\hat{c}_2 = c_1 c'_1$. The latter contains the product mm' encrypted under s^2 . Hence, we need an evaluation key $\text{ek} = (\hat{\mathbf{b}}, \hat{\mathbf{a}})$, where $\hat{\mathbf{a}} \leftarrow R_q^\ell$, $\mathbf{e} \leftarrow \chi^\ell$ and $\hat{\mathbf{b}} = -\hat{\mathbf{a}} \cdot s + t \cdot \mathbf{e} + s^2 \cdot \mathbf{g} \pmod{q}$. The evaluation key encrypts information about s^2 encrypted under s . For stronger security, the secret s^2 should be encrypted under a different secret s' to avoid making the circular security assumption, which states that the security of the secret key is ensured under the protection of its own public key. The transformation of the ciphertext is done by computing the new relinearized ciphertext as following: (1) apply the gadget decomposition algorithm to output the vector $\mathbf{g}^{-1}(\hat{c}_2)$; (2) compute the two dot products $u = \langle \mathbf{g}^{-1}(\hat{c}_2), \hat{\mathbf{a}} \rangle$ and $v = \langle \mathbf{g}^{-1}(\hat{c}_2), \hat{\mathbf{b}} \rangle$; (3) finally, add the results u, v to the first two elements and output the new ciphertext as $\mathbf{c}_{\text{mult}} = (\hat{c}_0 + v, \hat{c}_1 + u)$. This is essentially a way of homomorphically computing the decryption $\hat{c}_2 \cdot s^2$ with secret s^2 and embedding it in the ciphertext result, so it becomes decryptable with s .

4.1.2 The Brakerski/Fan-Vercauteren (BFV) scheme. Similar to the BGV scheme, the BFV scheme [21, 49] is also designed based on the original BV scheme [24]. Yet, instead of the modulus switching, it introduces an approach, called *scale-invariant*, to reduce noise accumulated through homomorphic multiplications. The BFV scheme also adopts the original BV's key switching technique for relinearization, which brings back the initial ciphertext product to be an encryption under s as explained in the previous subsection. We present the RLWE-based BFV scheme in Scheme 2 for a circuit depth L and a set of public parameters $\text{pp} = \{\lambda, d, q, t\}$.

The intuition behind scale-invariance is that elements' features should not change when scaled with a common factor. Given two messages m and m' scaled by a shared factor Δ . If we aggregate

$\Delta m + \Delta m'$ and scaled back by $\frac{1}{\Delta}$, then the result is roughly $m + m'$, i.e., the relation is preserved regardless to the scale. Now, let $\Delta = \lfloor q/t \rfloor$ be the factor used to scale up the message, since q is significantly larger than t , at encryption then add an initial noise term as described in BFV.Enc. If we decrypt, we need to scale back by the factor $\frac{t}{q}$ to retrieve the message m . As a result, the noise is significantly reduced by this scaling down operation. Observe the following proof of decryption correctness for the decryption $\left\lfloor \frac{t}{q} \langle c, s \rangle \right\rfloor \in R_t$, where we compute $\langle c, s \rangle$ as follows.

$$\begin{aligned} \langle c, s \rangle &= c_0 + c_1 s = (rb + e_0 + \Delta m) + (ra + e_1) s \\ &= r(b + as) + (e_0 + e_1 s) + \Delta m = \tilde{e} + \Delta m \end{aligned}$$

for a small error $\tilde{e} = re + e_0 + e_1 s$. Then, we re-scale the result by a factor $\frac{t}{q}$ as follows to obtain the message m .

$$\left\lfloor \frac{t}{q} \langle c, s \rangle \right\rfloor = \left\lfloor \frac{t}{q} (\tilde{e} + \Delta m) \right\rfloor = m \pmod{t}$$

We can use the scaling aspect to effectively reduce the noise after homomorphic evaluations – namely multiplication, which causes the embedded noise in the ciphertext to grow. In many HE schemes, the noise growth from multiplication is dominated by $e_1 e_2$ from the two original ciphertexts c_1 and c_2 , but in BFV the noise growth is dominated by $I_1 e_2 + I_2 e_1$ where $I_i = \Delta m_i - m_i$ is a factor introduced when we scale up the messages with Δ . Because $\Delta = \lfloor \frac{q}{t} \rfloor$ and $q \gg t$, then $I_1 e_2 + I_2 e_1$ is larger than $e_1 e_2$. Moreover, multiplying two ciphertexts results in squaring the scaling factor, such that we get $\Delta^2(mm')$. To address this issue, the ciphertext must be scaled down by a factor of $\frac{t}{q}$ after each multiplication to reduce the noise and brings the result scale back to $\Delta(mm')$. Note, we could have divided the ciphertext by the factor of Δ . However, this will incur twice the rounding error. We may not effectively reduce the large noise in the ciphertext scaled during multiplication; hence, we directly scale by $\frac{t}{q}$ instead. Homomorphic addition does not require a scaling step because the sum of two ciphertexts directly obtains $\Delta(m + m')$ and does not change the shared factor.

Unlike modulus switching, we observe right away that scale-invariance requires *one* modulus q instead of a set of decreasing $L + 1$ moduli $\{q_L > q_{L-1} > \dots, q_0\}$. The modulus switching used in the BGV scheme performs a gradual scaling after each multiplication to map a ciphertext from R_{q_i} to $R_{q_{i-1}}$. This causes the ciphertext to be scaled by a factor of $\frac{q_i}{q_{i-1}}$. This means the noise magnitude after each multiplication remains constant, but at the cost of changing to a smaller modulus q .

4.1.3 The Gentry-Sahai-Waters (GSW) scheme. Gentry et al. [56] proposed another LWE-based HE scheme based on eigenvectors and eigenvalues of a transformation matrix. This new scheme is simpler and asymptotically faster than other BV variants because it does not require the complex relinearnization for dimension reduction and modulus switching or scale-invariant for noise reduction. The intuition can be illustrated as follow. Given a transformation matrix (ciphertext), C , and an eigenvector (secret key), s , we can retrieve an eigenvalue (message), m , such that $sC = ms \pmod{q}$. From this toy construction, homomorphic addition and multiplication of two ciphertexts C and C' , encrypted under the same secret s , can be realized easily by computing $s(C + C') = (m + m')s \pmod{q}$ and $s(C \cdot C') = (mm') \cdot s \pmod{q}$, respectively.

However, this toy construction is insecure as it can be easily broken because finding eigenvectors for a given transformation matrix is as easy as solving a system of linear equations. To address this problem, the actual GSW scheme is built on the *approximated eigenvectors* method, where a small noise e is added to satisfy the LWE assumption, such that $sC = ms + e \pmod{q}$. Note, we can view sC as the decryption of some ciphertext C containing a message m with a secret key s associated to a public key A .

Scheme 2: The Brakerski/Fan-Vercauteren (BFV) scheme [21, 49]

- **BFV.Setup**($1^\lambda, 1^L$): Given the security parameter λ and a multiplicative depth L , choose a cyclotomic polynomial $\Phi(x) = x^d + 1$, where d is a power of 2. Define $R = \mathbb{Z}[x]/(\Phi(x))$ as a polynomial ring of degree d with integer coefficients. Generate the error distribution χ over R bounded by \mathcal{B} . Generate the key distribution ψ over R . Choose two integers q and t as the ciphertext and plaintext moduli, respectively. Uniformly sample an element $a \leftarrow R_q$. Finally, output $\text{pp} = (R, \chi, \psi, \mathcal{B}, q, t, a)$ as the public parameters of scheme. We assume all following algorithms implicitly take pp as an input.
- **BFV.KeyGen**(p): Given the public parameters p , sample a secret $s \leftarrow \psi$, an element $a \leftarrow R_q$, and a small noise $e \leftarrow \chi$. Set the secret key as $\text{sk} = s$ and the public key as $\text{pk} = (b, a) \in R_q^2$ where $b = -as + e \pmod{q}$.
- **BFV.Enc**(pk, m): Given a public key $\text{pk} = (b, a)$ and a message $m \in R_t$, sample a random $r \leftarrow \psi$ and noise elements $e_0, e_1 \leftarrow \chi$. Scaled up the message by a factor Δ such that $\Delta m = \lfloor \frac{q}{t} \rfloor m$. Encrypt the scaled message as $\mathbf{c} = (c_0, c_1) \in R_q^2$ where $c_0 = rb + e_0 + \Delta m$ and $c_1 = ra + e_1$.
- **BFV.Dec**(sk, \mathbf{c}): Given a ciphertext $\mathbf{c} = (c_0, c_1) \in R_q^2$ and the corresponding secret key sk , set $s = (1, s)$ and decrypt by computing $\lfloor \frac{t}{q} \langle \mathbf{c}, \mathbf{s} \rangle \rfloor \in R_t$.
- **BFV.Add**(\mathbf{c}, \mathbf{c}'): To add two ciphertexts $\mathbf{c} = (c_0, c_1)$ and $\mathbf{c}' = (c'_0, c'_1)$, compute $\mathbf{c}_{\text{add}} = (c_0 + c'_0, c_1 + c'_1) \pmod{q}$.
- **BFV.Mult**(\mathbf{c}, \mathbf{c}'): To multiply two ciphertexts $\mathbf{c}, \mathbf{c}' \in R_q^2$, compute first the ciphertext $\tilde{\mathbf{c}}_{\text{mult}} = (\tilde{c}_0, \tilde{c}_1, \tilde{c}_2) \in R_q^3$ by $\tilde{c}_0 = \lfloor (t/q) \cdot (c_0 c'_0) \rfloor$, $\tilde{c}_1 = \lfloor (t/q) \cdot (c_0 c'_1 + c_1 c'_0) \rfloor$, and $\tilde{c}_2 = \lfloor (t/q) \cdot (c_1 c'_1) \rfloor$. The ciphertext is encrypted under the element s^2 and need to be transformed to one under element s through the *key switching* technique. We denote this process by $\mathbf{c}_{\text{mult}} = \text{KeySwitch}(\tilde{\mathbf{c}}_{\text{mult}}, \text{ek})$.

Like other LWE-based schemes, the public key in GSW is generated from an LWE instance over \mathbb{Z}_q . For a given secret vector \hat{s} and a uniformly chosen matrix \mathbf{B} , we set the public key as $\mathbf{A} = (\mathbf{B}\hat{s} + \mathbf{e}, \mathbf{B})$ and the secret key as $\mathbf{s} = (1, -\hat{s})$. When the secret key \mathbf{s} is provided at decryption, we have $\mathbf{sA} \approx 0$ because $\mathbf{sA} = (1, -\hat{s})(\mathbf{B}\hat{s} + \mathbf{e}, \mathbf{B}) = \mathbf{B}\hat{s} + \mathbf{e} - \mathbf{B}\hat{s} = \mathbf{e} \approx 0$. This operation removes the masking matrix \mathbf{B} in the ciphertext. To encrypt a message m under the public key \mathbf{A} , we sample a random matrix \mathbf{R} , to ensure semantic security of the scheme, and compute $\mathbf{C} = \mathbf{AR} + m$ such that the decryption $\mathbf{sC} = \mathbf{s}(\mathbf{AR} + m) = \mathbf{sAR} + m\mathbf{s} = m\mathbf{s} + \mathbf{eR}$, where \mathbf{eR} is small when \mathbf{R} has entries in \mathbb{Z}_2 . If \mathbf{s} is an integer vector and m is restricted to the message space $\{0, 1\}$, we can determine the original message from $(m\mathbf{s} + \mathbf{e})$ with high probability. Specifically, if $m = 0$, then \mathbf{sC} is close to 0; or an integer vector otherwise.

Homomorphic evaluations are performed as natural matrix operations. The product of multiplying two GSW ciphertexts does not contain a long secret s^2 . Therefore, it does not require the complex relinearization operation used in the previous BV variant schemes. Yet, homomorphic multiplication still increases the noise. The product of multiplying two GSW ciphertexts \mathbf{C} and \mathbf{C}' is

$$\begin{aligned}
 \mathbf{sC}_{\text{mult}} &= \mathbf{sCC}' = (m\mathbf{s} + \mathbf{e})\mathbf{C}' \\
 &= m(m'\mathbf{s} + \mathbf{e}') + \mathbf{C}'\mathbf{e} \\
 &= mm'\mathbf{s} + (m\mathbf{e}' + \mathbf{C}'\mathbf{e})
 \end{aligned}$$

The new error term is $(me' + C'e)$. Multiplication works if this term is small. While e and e' are small noise, we need to make m and C' small. The former can be achieved by restricting input message, $m \in \{0, 1\}$, but C' (mod q) is large making $C'e$ a large noise. Without optimizations, the noise growth for multiplying two ciphertexts is exponential and bounded around $(n\ell + 1)\mathcal{B}^2$ for some bound \mathcal{B} . This shows a significant noise increase compared to homomorphic addition that is bounded by $2\mathcal{B}$. To achieve a better noise management, the scheme applies a *flattening* technique to transform C' with integer coefficient module q into bits through BitDecomp and PowersOf2, as discussed in Section 3.3. Hence, the noise growth becomes linear and bounded by $(n\ell + 1)\mathcal{B}$ as a result of multiplying the noise term with small values in \mathbb{Z}_2 .

A variant of GSW [6] later suggested the use of special *gadget matrix* for a simpler design with a tighter bound on the noise growth. The gadget matrix G is a diagonal matrix containing powers of two $\mathbf{g} = (1, 2, 4, \dots, 2^\ell)$, where $\ell = \lceil \log q \rceil$ and associated with a function $G^{-1}(\cdot)$ such that $GG^{-1}(a) = a$ for any given $a \in \mathbb{Z}_q$. It also proposed a bootstrapping technique for GSW to support arbitrary homomorphic evaluations. We describe the main functions of this GSW variant in Scheme 3.

Due to the construction of the GSW, there is another way to decrypt and retrieve the message bit $m \in \{0, 1\}$. More specifically, if we view a GSW ciphertext $C = \mathbf{AR} + m\mathbf{G}$ as two aggregated parts, a randomized public key \mathbf{AR} and a message m masked by a gadget matrix G represented in the powers-of-2 form. Then the i -th column c_i of this ciphertext contains the encryption of $m2^i$ and can be decrypted by computing $\langle c_i, \mathbf{s} \rangle \pmod{2}$. The collection of entries are required for homomorphic evaluations, but decryption can be done using a single column. The penultimate (i.e., second-to-last) column corresponds to encryption of $m \cdot 2^{\ell-2}$, which decrypts to plaintext in the interval $(q/4, q/2]$ appropriate for decoding the message bit. Hence, extracting the penultimate column is sufficient to use for decryption. To illustrate, we provide a proof of correctness for the GSW decryption as follows. Let $\mathbf{w} = (0, 0, \dots, \lceil q/2 \rceil)$ be the penultimate column of the gadget matrix G corresponding to the power $2^{\ell-2}$ and compute \mathbf{x} as following.

$$\begin{aligned} \mathbf{x} &= \mathbf{sCG}^{-1}(\mathbf{w}^T) = \mathbf{s}(\mathbf{AR} + m\mathbf{G})G^{-1}(\mathbf{w}^T) \\ &= m\mathbf{sGG}^{-1}(\mathbf{w}^T) \\ &= m\mathbf{sw}^T \end{aligned}$$

This process extracts the penultimate column of the GSW ciphertexts, which is sufficient to for decryption. Compute $m = \left\lfloor \left\lfloor \frac{\mathbf{x}}{q/2} \right\rfloor \right\rfloor$ and retrieve the message $\tilde{m} \in \{0, 1\}$ after checking whether it is closer to 0 or $q/2$. Using a single column from the ciphertext to decrypt certainly has its benefits. One of them is reducing the size of the ciphertext results transmitted over the network, especially in cases we discuss later in Sec. 5.2, where ciphertexts are extended to multiple keys and may suffer exponential increase in their size. More importantly, this property provides a mechanism to decrypt a ciphertext without fully revealing information about the embedded secret key. We review in later sections how this property is exploited to build a secure proxy re-encryption that is based on the GSW scheme.

4.1.4 The Cheon-Kim-Kim-Song (CKKS) scheme. Previous HE schemes support linear transformations of encrypted integers or bits through homomorphic addition and multiplication. However, many machine learning algorithms, such as logistic regression and neural networks, often operate on real numbers (e.g., 12.345). In integer-based HE schemes, a common way to deal with these real numbers is to scale them up by a relatively large factor (e.g., 10^6) prior to encryption so that we only need to work with integers (e.g., 12.345×10^6). The scaling dramatically increases the input values and subsequently requires a much larger plaintext modulus and even larger ciphertext modulus. As a result, the computation time will be significantly impacted.

Scheme 3: A variant of the Gentry-Sahai-Waters (GSW) scheme [6]

- $\text{GSW.Setup}(1^\lambda, 1^L)$: Given a security parameter λ and a circuit depth L , choose a lattice dimension n , an error distribution χ bounded by \mathcal{B}_χ , and a modulus q such that the LWE problem holds. Set $\ell = \lceil \log q \rceil$ and choose a random matrix $B \leftarrow \mathbb{Z}_q^{n-1 \times n\ell}$. Output the scheme public parameters as $\text{pp} = (q, n, \chi, \mathcal{B}_\chi, B)$.
- $\text{GSW.KeyGen}(\text{pp})$: Sample a secret vector $\hat{s} \leftarrow \mathbb{Z}_q^{n-1}$ and the noise vector $e \leftarrow \chi^{n\ell}$. Set $b = \hat{s}B + e \in \mathbb{Z}_q^{n\ell}$. Output the secret key sk as $s = (1, -\hat{s}) \in \mathbb{Z}_q^n$ and the public key pk as $A = (b, B) \in \mathbb{Z}_q^{n \times n\ell}$, observe that $sA \approx 0$ with respect to some small noise.
- $\text{GSW.Enc}(\text{pk}, m)$: Let $m \leftarrow \{0, 1\}$ be a message bit and pk be a given public key. To encrypt the message, choose a random matrix $R \leftarrow \{0, 1\}^{n\ell \times n\ell}$ compute the ciphertext as $C = AR + mG \in \mathbb{Z}_q^{n \times n\ell}$, such that the property $sC \approx msG$ holds.
- $\text{GSW.Dec}(\text{sk}, C)$: To decrypt a given ciphertext C with the associated secret key sk , define a vector $w = (0, \dots, 0, \lceil q/2 \rceil) \in \mathbb{Z}_q^n$. Then compute $x = sCG^{-1}(w^T) \in \mathbb{Z}_q^n$. Output the decrypted message as $\tilde{m} = \left\lfloor \left\lceil \frac{x}{q/2} \right\rceil \right\rfloor$.
- $\text{GSW.Add}(C, C')$: To add two ciphertexts $C = (AR + mG)$, $C' = (AR' + m'G)$, directly output $C_{\text{add}} = C + C' = A(R + R') + (m + m')G \in \mathbb{Z}_q^{n \times n\ell}$, such that $sC_{\text{add}} \approx (m + m')sG$.
- $\text{GSW.Mult}(C, C')$: To multiply two ciphertexts C, C' , utilize the gadget matrix and output $C_{\text{mult}} = C \cdot C' = CG^{-1}(C') \in \mathbb{Z}_q^{n \times n\ell}$. The product matrix satisfies $sC_{\text{mult}} \approx sCG^{-1}(C') \approx msGC' \approx (m \cdot m')sG$.

Cheon et al. [35] proposed a HE scheme that supports fixed-point arithmetic over encrypted real numbers. The scheme bases its security on the RLWE problem and is based on a construction that is similar to the rest of BV-type schemes discussed in Sections 4.1.1 and 4.1.2. The main intuition is treating the noise embedded at encryption, to realize the LWE assumption, as the rounding error in fixed-point arithmetic since real numbers inherently carry error through rounding. We give an overview of the RLWE-based CKKS scheme in Scheme 4.

What is unique in the CKKS scheme is the proposal of new message encoding and decoding methods and a rescaling operation that reduces the accumulating noise in a similar way as rounding in plaintext fixed-point arithmetic.

Before encryption, the encoding method applies a canonical embedding to map a vector of complex or real numbers into a polynomial in R . Let d being the degree of the cyclotomic polynomial $\Phi(x) = x^d + 1$ and determines the number of slots for ciphertext packing, the encoding procedure allows $d/2$ messages $(z_1, z_2, \dots, z_{d/2})$ to be packed into a polynomial $m' \in R$ so that efficient evaluation can be carried out in SIMD manner. If the number of messages in the vector is less than $d/2$, the remaining slots are set to zero. The other half of slots in m' will be filled with conjugates of the corresponding message z_i during the canonical embedding. The polynomial m' is then scaled by a large factor Δ , say $\Delta = 2^{40}$, to place the significand far away from the LSBs (least significant bits). Note, placing significand far away from the LSBs is to accommodate the noise growth and to preserve precision during the rescaling operation, as illustrated in Fig. 4. The output of the encoding method is a plaintext $m \in R$ containing the packed messages scaled by a factor Δ . At encryption, the plaintext m is masked with a randomized public key. A small noise e is added to ensure the security, such that $c = r(b, a) + (e_0, +m, e_1)$. If we decrypt the ciphertext, we obtain the message $\text{Dec}(\text{sk}, c) = m + e$, where e is a small noise which can be considered as small rounding error inherited from fixed-point

Scheme 4: The Cheon-Kim-Kim-Song (CKKS) scheme [35]

- CKKS.Setup($1^\lambda, 1^L$): Given the security parameter λ and a multiplicative depth L , choose a cyclotomic polynomial $\Phi(x) = x^d + 1$, where d is a power of 2. Define $R = \mathbb{Z}[x]/(\Phi(x))$ as a polynomial ring of degree d with integer coefficients. Generate the error distribution χ over R bounded by \mathcal{B} . Generate the key distribution ψ over R . Choose an integer q as the ciphertext modulus. Uniformly sample a vector $\mathbf{a} \leftarrow R_q$. Finally, output $\text{pp} = (R, \chi, \psi, \mathcal{B}, q, \mathbf{a})$ as the public parameters of scheme. We assume all following algorithms implicitly take pp as an input.
- CKKS.KeyGen(pp): Given the public parameters pp , sample a small element $s \leftarrow \chi$ and a small noise $e \leftarrow \psi$. Uniformly sample $a \leftarrow R_q$. Output the secret key as $\text{sk} = s$ and the public key as $\text{pk} = (b, a) \in R_q^2$ where $b = -as + e \pmod{q}$.
- CKKS.Enc(pk, m): Given a public key pk and a plaintext message m , uniformly sample a random $r \leftarrow \psi$ and noise elements $e_0, e_1 \in \chi$. Encrypt the message m as $\mathbf{c} = (c_0, c_1) \in R_q^2$ where $c_0 = rb + e_0 + m$ and $c_1 = ra + e_1$.
- CKKS.Dec(sk, \mathbf{c}): Given a ciphertext $\mathbf{c} = (c_0, c_1) \in R_q^2$ and the secret key $\text{sk} = s$, set $\mathbf{s} = (1, s)$ and decrypt by computing $m = \langle \mathbf{c}, \mathbf{s} \rangle \pmod{q}$.
- CKKS.Add(\mathbf{c}, \mathbf{c}'): For two ciphertexts $\mathbf{c} = (c_0, c_1)$ and $\mathbf{c}' = (c'_0, c'_1)$, output $\mathbf{c}_{\text{add}} = (c_0 + c'_0, c_1 + c'_1) \in R_q^2$.
- CKKS.Mult(\mathbf{c}, \mathbf{c}'): For two ciphertexts $\mathbf{c}, \mathbf{c}' \in R_q^2$, their initial homomorphic multiplication yields a long ciphertext $\tilde{\mathbf{c}}_{\text{mult}} = (\tilde{c}_0, \tilde{c}_1, \tilde{c}_2) \in R_q^3$ encrypted under the secret $(1, s, s^2)$. With a provided evaluation key ek containing encryption of s^2 , the key switching technique can be used to transform $\tilde{\mathbf{c}}_{\text{mult}}$ into a normal ciphertext $\mathbf{c}_{\text{mult}} = (c_0, c_1) \in R_q^2$ that is decryptable under s .
- CKKS.Rescale(\mathbf{c}): For an evaluated ciphertext $\mathbf{c} = (c_0, c_1) \in R_q^2$, compute $c'_i = \lfloor \Delta^{-1} \cdot c_i \rfloor$ for $i \in \{0, 1\}$ and a scaling factor Δ and return the re-scaled ciphertext $\mathbf{c}' = (c'_0, c'_1) \in R_{q'}^2$ where $q' = \Delta^{-1} \cdot q$.

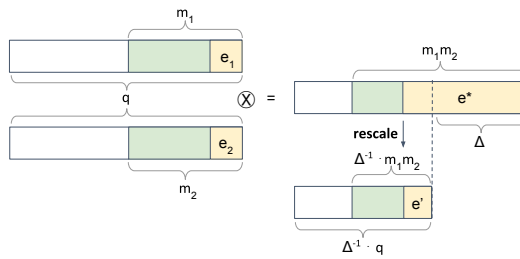


Fig. 4. Illustration of ciphertext re-scaling in CKKS

arithmetic. The decrypted plaintext is then decoded from a polynomial in R to a real or complex message vector $(z_1, z_2, \dots, z_{d/2})$ using a reverse procedure of the encoding.

Homomorphic multiplication causes the scaling factor to square and increases embedded noise. In order to maintain the same precision and prevents the noise from blowing up, the result must be “rounded” homomorphically. To do so, the scheme rescales the ciphertext after each multiplication. This method can be viewed as similar to the rounding performed on the plaintexts in approximate computations. In particular, the ciphertext \mathbf{c} can be multiplied by the scaling factor, such that $\Delta^{-1}\mathbf{c}$,

which discards $\log \Delta$ least significant bits as illustrated in Fig. 4. As a result, we obtain a ciphertext encrypting the message $\frac{m_{\text{mult}}}{\Delta}$ with a reduced noise $\Delta^{-1} \cdot e_{\text{mult}}$, and the modulus is switched to $q' = \Delta^{-1} \cdot q$ corresponding to the new scaled ciphertext. CKKS uses a technique similar to modulus switching to perform rescaling. For a circuit depth L , choose a large modulus q_0 and define a decreasing chain of moduli q_L, q_{L-1}, \dots, q_0 , such that the modulus for a level $l \in L$ is defined as $q_l = q_0 \cdot \Delta^l$. A ciphertext $c \pmod{q_l}$ is rescaled as $\lfloor \Delta^{-1} \cdot c \rfloor \pmod{q_{l-1}}$. The modulus switching technique used here is for a different purpose compared to its use in the BGV. While it preserves the underlying plaintext and manages the noise growth in BGV, it is used in CKKS to remove the LSBs of the ciphertext to maintain the precision. Because of this recurring discard of the LSBs, we need to choose appropriate parameters for the modulus q_0 and the scaling factor Δ according to the circuit depth L to prevent precision loss. Specifically, we ensure that $(\log q_L \geq L \cdot \log \Delta)$ to construct a leveled HE scheme sufficient to evaluate an L -depth circuit. After reaching the level q_0 , bootstrapping [28, 33] may be applied to output a refreshed ciphertext at a higher level that supports further homomorphic operations.

A subsequent construction of the CKKS [34] proposed a variant based on the Residue Number System (RNS) to optimize the performance. Most HE scheme implementations requires that the modulus q is at least 128 bits (often significantly larger), which may not fit in the native 64-bit modern hardware. As an optimization, they apply Chinese Remainder Theorem (CRT) and similar techniques to represent the modulus as a set of moduli less than 64-bit. The RNS-variant decomposes the large modulus q into a sequence of smaller distinct prime moduli q'_0, q'_1, \dots, q'_L , such that $q = \prod_{i=0}^L q'_i$. By choosing a chain of moduli that are pairwise coprimes, we can no longer perform the original rescaling technique since the moduli do not correspond to the scaling factor Δ . To address this issue, the RNS-variant proposes an *approximate* rescaling technique, which chooses the primes such that their ratio is as close as the scaling factor. For example, the prime moduli q'_l is chosen such that $q'_l \approx \Delta$ so that $\frac{q_l}{q_{l-1}} = q'_l \approx \Delta$. After homomorphic multiplication, the ciphertext product with a squared scale Δ^2 is rescaled by the factor $\frac{q_l}{q_{l-1}}$ to a ciphertext encrypting of $m_1 m_2$ with an approximate scale Δ . The new ciphertext contains an additional small approximation error, but the most significant bits of the plaintext remain extractable.

4.1.5 Discussion. From reviewing these base HE schemes, we observe similarities in their design pattern and methods to decrease in ciphertext dimension and to deal with noise. We share these similarities and discuss some initial ideas to support homomorphic computation on ciphertexts encrypted under multiple keys.

In term of similarity, the most recent HE schemes are constructed based on the LWE problem, or its ring variant, RLWE. The first unique characteristic is in the way that some small noise is added when masking a secret. Another characteristic is that the public key or the ciphertext contains a counter-part that can cancel the large masking element sampled from some mathematical space. These two characteristics are clearly visible in the KeyGen and Enc algorithms of the presented HE schemes. Homomorphic multiplication is highly inefficient because it often raises the ciphertext dimension from an encryption of s to s^2 (except GSW due to its construction) and dramatically increases the embedded noise. Hence, focuses are on designing techniques to reduce the ciphertext dimension and noise. Among these techniques, key switching used in dimension reduction is useful to support multi-key HE. Whereas common noise reduction techniques, such as modulus switching in BGV, scale invariant in BFV, rescaling in CKKS, follow a typical pattern of factorizing the noise by a scaling factor after each multiplication. Often, these reduction techniques have impact on the embedded messages, hence plaintext messages are typically scaled up before encryption.

There are two common tricks we can apply to extend the base HE schemes to support homomorphic computation on ciphertexts encrypted using multiple keys.

The first trick is to follow a concept called *layered encryption* [70]. Let c be a ciphertext encrypting a message m under Alice's key pk_A . We can add second layer of encryption under Bob's key pk_B , such that $\tilde{c} = \text{Enc}(pk_B, \text{Enc}(pk_A, m))$. In theory, the ciphertext can be further extended indefinitely. Obviously, the problem is at decryption. Bob requires Alice, and other involved parties if any, to remove the corresponding layer of encryption. Homomorphic evaluations can be applied, but all ciphertexts must be encrypted in the same order. The number of evaluation is very restricted due to exponential increase of ciphertext size and noise magnitude as we further discuss in Sec. 5.

On the other hand, one may leverage *key switching* to change the encryption of a message from one key pk to another pk' . This method follows the notion of proxy re-encryption technique we discuss in the following section. Recall that a special evaluation key $ek_{pk \rightarrow pk'}$ which embeds information about both keys, we can perform $\text{KeySwitch}(ek_{pk \rightarrow pk'}, \text{Enc}(pk, m)) = \text{Enc}(pk', m)$ to achieve this purpose. The major issue in this method is to ensure the security of the secret key sk because the generation of the evaluation key requires encrypting sk under the new key pk' . If the secret key sk' is owned by another party, then he or she can obtain sk violating the privacy requirement of data encrypted under pk .

These simple tricks are neither secure nor efficient in practice. Hence, we review additional techniques which offer more than just simple extension of the base HE schemes.

4.2 Proxy Re-encryption

Changing a ciphertext from the encryption of one key to another *without* exposing the plaintext is a useful feature in many applications. Especially, in cloud computing applications where a user Alice stores her encrypted data in the cloud as illustrated in Fig. 1c and another user Bob wants to perform some evaluations and retrieve the result from the cloud. It is reasonable to assume that Alice does not want to share her private key.

Without Alice's private key, Bob will not be able to decrypt the result. To address this problem, we can design the system using an interactive model. The cloud can blind the homomorphically evaluated ciphertext with a randomness and sends it to Alice, who decrypts it with sk_A and sends back another ciphertext re-encrypted under pk_B . Upon receiving the new ciphertext, the cloud evaluator homomorphically removes the blinding element and continues remaining evaluations or forwards the result to Bob, who can decrypt with sk_B . Although this method is also applicable for changing the encryption under different schemes as needed, e.g., to convert from a SWHE scheme to a PHE scheme for efficiency [20], it is cumbersome due to the interactivity.

However, changing the encryption under different keys within the same scheme can be done non-interactively. This gives the advantage of fully outsourcing computations to the cloud, i.e., Alice does not need to participate in evaluations.

Proxy re-encryption (PRE) is the process of converting a ciphertext from an encryption under a public key pk to one under another pk' without decryption. A third party (the *proxy*) is delegated to perform this re-encryption without disclosing the underlying message or the secret key sk . Consider the previous cloud application addressed by the PRE scheme as shown in Fig. 5. With a PRE scheme, Alice (the *delegator*) provides to the proxy cloud a special re-encryption key $rk_{A \rightarrow B}$ used to re-encrypt the ciphertext, so it becomes decryptable by Bob (the *delegatee*). The key is generated based on her secret key and Bob's public key, such as $rk_{A \rightarrow B} = \text{ReKeyGen}(sk_A, pk_B)$. Variations of proposed PRE schemes can be unidirectional $rk_{A \rightarrow B}$ [68, 92], where Alice's ciphertext can be only converted into Bob's, but not the other way around, or bidirectional $rk_{A \leftrightarrow B}$ [12, 27], where the ciphertext can be converted from Alice's to Bob's, or vice versa. We focus on the former in this review.

The formal construction of PRE consists of five PPT algorithms, $\text{PRE} = (\text{KeyGen}, \text{ReKeyGen}, \text{Enc}, \text{ReEncrypt}, \text{Dec})$. Existing HE schemes can be extended to PRE by adding the two PRE-specific

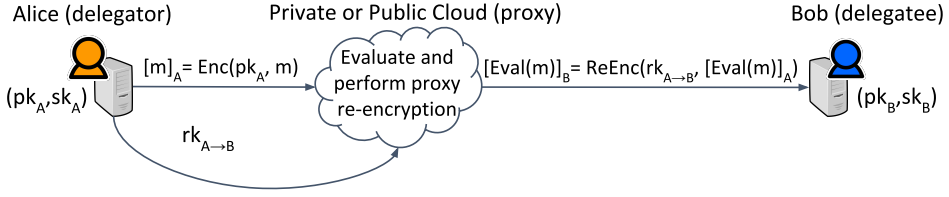


Fig. 5. Illustration of proxy re-encryption.

algorithms (ReKeyGen, ReEncrypt). We briefly describe these algorithms with the PRE variant of ElGamal scheme in Scheme 5 as an example. The generation of the re-encryption key in ReKeyGen is scheme-dependent, i.e., it depends on the mathematical construction of the base scheme. For example, the key rk in ElGamal scheme is generated as $rk_{A \rightarrow B} = g^{s_B/s_A}$ from Bob's public key $pk_B = g^{s_B}$ and Alice's secret key $sk_A = s_A$. The re-encryption process leverages basic rules for exponentiation to remove s_A from the encryption. Regarding security, recall that ElGamal bases its security on the discrete logarithm problem¹. This means it is difficult for the proxy, or Bob, to learn the secret key s_A from the re-encryption key.

Scheme 5: A PRE-variant of ElGamal scheme [12]

- ElGamalPRE.KeyGen(\mathbb{G}, p, g): Let \mathbb{G} be a cyclic group of order p and a generator g . Sample two random values $s_A, s_B \leftarrow \mathbb{Z}_p$ and compute $h_A = g^{s_A}, h_B = g^{s_B}$. Set the key pairs $\{pk_A = (\mathbb{G}, p, g, h_A), sk_A = s_A\}$ for Alice, and $\{pk_B = (\mathbb{G}, p, g, h_B), sk_B = s_B\}$ for Bob.
- ElGamalPRE.ReKeyGen(sk_A, pk_B): To allow Bob to transform a ciphertext under Alice's public key to one under his, Alice generates a re-encryption key $rk_{A \rightarrow B}$. Given Alice's own secret key sk_A and Bob's public key pk_B , compute $rk_{A \rightarrow B} = pk_B^{1/sk_A} = (g^{s_B})^{1/s_A} = g^{s_B/s_A}$.
- ElGamalPRE.Enc(pk_A, m): To encrypt a given a message m under Alice's public key, choose a random value $r \in \{1, \dots, p-1\}$ and compute the ciphertext $c = (c_1, c_2) = (g^r, mh_A^r)$.
- ElGamalPRE.ReEncrypt($rk_{A \rightarrow B}, c$): Given a ciphertext $c = (c_1, c_2)$ and a re-encryption key $rk_{A \rightarrow B}$, a proxy can re-encrypt the ciphertext as $c_2 \cdot rk_{A \rightarrow B} = (mg^{r s_A} \cdot g^{s_B/s_A}) = mg^{r(s_A s_B/s_A)} = mg^{r s_B}$ to obtain $\hat{C} = (c_1, c'_2)$ under Bob's public key.
- ElGamalPRE.Dec(sk_B, \hat{C}): Given a re-encrypted ciphertext $\hat{C} = (c_1, c'_2)$ and a corresponding secret key sk_B , Bob decrypts by performing $c'_2/c_1^{s_B} = (mg^{r s_B})/(g^r)^{s_B} = m$.

In LWE-based HE schemes, performing key switching obtains a re-encrypted message under a different key without revealing it. The process requires an evaluation key, which is conceptually the same as re-encryption key in PRE. In this context, we will refer to the evaluation key as the re-encryption key. As discussed in Sec. 4.1, the technique is often utilized for relinearization to convert initial results of homomorphic multiplication from one under s^2 back to one under s . The re-encryption key is generated as a special encryption of s^2 under s . For example in the BGV scheme, the re-encryption key is generated by the party who has access to both keys $sk = s^2$ and $pk' = (a's' + te', -a')$. To avoid making circular security assumption, BGV chooses pk' to be new public key containing a different secret from s for each evaluation level. The security of the secret s^2 is ensured by the LWE problem.

¹For a carefully chosen cyclic group $\mathbb{G} = \mathbb{Z}_p^\times$ with a prime modulus p and a generator g , the problem states that given an element $h = g^x \in \mathbb{G}$, finding the discrete logarithm x is difficult.

In theory, any HE scheme can be extended to PRE by following the key switching technique. The PALISADE software library [2] implements the PRE primitive for its HE schemes, including BGV, BFV, and CKKS, where the re-encryption key is generated based on two provided secret keys. However in practice, due to the scheme construction based on the LWE problem, the secret key is retrievable if one can decrypt the re-encryption key. Specifically, Alice generates $rk_{A \rightarrow B}$ as an encryption of her secret key sk_A under Bob's public key pk_B . This design is insecure since Bob may collude with the proxy and obtain sk_A after decrypting $rk_{A \rightarrow B}$ with his secret key sk_B . The re-encrypted message under pk_B may also leak information about sk_A when decrypted. Therefore, additional measures must be taken when extending LWE-based schemes to PRE to prevent leaking the delegator's secret key.

Recently, Yasuda et al. [105] extended a GSW variant [56, 81] to PRE using key switching. We describe this PRE variant of GSW in Scheme 6. Alice generates $rk_{A \rightarrow B}$ as a GSW encryption of sk_A under Bob's public key pk_B and sends it to the proxy. Note, decrypting the re-encryption key with Bob's secret key $sk_B = s_B$ yields Alice's secret key according to this proof.

$$s_B rk_{A \rightarrow B} = s_B (A_B X + \begin{pmatrix} s_A G \\ \mathbf{0}_{(n-1) \times n\ell} \end{pmatrix}) = s_B A_B X + (1, -s_B) \begin{pmatrix} s_A G \\ \mathbf{0}_{(n-1) \times n\ell} \end{pmatrix} \approx s_A G$$

Given a ciphertext C under Alice's public key, the proxy cloud re-encrypts it as $\hat{C} = rk_{A \rightarrow B} \cdot G^{-1}(C)$. However, if Bob receives and decrypts \hat{C} , he will be able to learn information about Alice's secret key from the plaintext $s_B \hat{C} \approx ms_A G$. Instead, the PRE scheme leverages the special GSW property where a ciphertext can be decrypted using only its penultimate column, which corresponds the power-of-two $2^{\ell-2}$. The last element of this column contains *partial* elements of the re-encrypted ciphertext, $\hat{C} = rk_{A \rightarrow B} \cdot G^{-1}(C)$, which contains in its last element the message $m2^{\ell-2}$. The proxy sends the penultimate column \hat{c} of the re-encrypted ciphertext to Bob. With releasing this column only, we not only reduce the size of the ciphertext, but also withhold possible information about s_A to prevent its leakage. This design renders a single-hop PRE scheme, which means \hat{c} cannot be re-encrypted or homomorphically evaluated after its re-encryption because it loses the GSW ciphertext structure. Still, it is useful to use the PRE primitive with other extended GSW schemes, such as the multi-key variant [81] (discussed later in Sec. 5.2) where the final result is re-encrypted under the receiver's key to avoid distributed decryption.

Proxy re-encryption is a powerful method to support secure outsourced evaluations. Ciphertexts encrypted under different keys can be individually re-encrypted under a receiver's (delegatee's) key so they can be homomorphically evaluated under the same key. Other applications, such as secure file sharing, may use this technique for basic access control to make ciphertexts decryptable with authorized user's secret keys, but it requires providing re-encryption keys to do so. We discuss an alternative approach to support this access control functionality without re-encryption keys in the following section.

4.3 Identity-/Attribute-based Encryption

Identity-based encryption (IBE) [15, 90], and its generalization attribute-based encryption (ABE) [86], is a type of encryption scheme that provides a fine-grained access control to the encrypted message based on users' identities or attributes.

Compared to the PRE scheme, IBE/ABE schemes do not require data owners to provide re-encryption keys for changing the ciphertexts encrypted under their public key to ones encrypted under the intended user's key. A data owner can encrypt their data under a derived public key based on predefined identities or attributes for authorized users. In IBE schemes, the data owner encrypts the data under a key derived from a user's identity ID, e.g., their email address. The ciphertext can be only decrypted with an authorized secret key sk_{ID} issued corresponding to the user's identity ID.

Scheme 6: A PRE-variant of GSW scheme [105]

- GSWPRE.Setup($1^\lambda, 1^L$): Given a security parameter λ and a circuit depth L , choose a lattice dimension n , an error distribution χ bounded by \mathcal{B}_χ , and a modulus q such that the LWE problem holds. Set $\ell = \lceil \log q \rceil$ and choose a random matrix $B \leftarrow \mathbb{Z}_q^{n-1 \times n\ell}$. Output the scheme public parameters as $\text{pp} = (q, n, \chi, \mathcal{B}_\chi, B)$.
- GSWPRE.KeyGen(pp): Given a security parameter λ , circuit depth L , and a GSW public parameters $\text{pp} = (q, n, \chi, \mathcal{B}_\chi, B)$, sample a secret vector $\hat{s} \leftarrow \mathbb{Z}_q^{n-1}$ and the noise vector $e \leftarrow \chi^{n\ell}$. Set $\mathbf{b} = \hat{s}B + e \in \mathbb{Z}_q^{n\ell}$. Output the secret key sk as $\mathbf{s} = (1, -\hat{s}) \in \mathbb{Z}_q^n$ and the public key pk as $A = (\mathbf{b}, B) \in \mathbb{Z}_q^{n \times n\ell}$, observe that $\mathbf{s}A \approx 0$ with respect to some small noise.
- GSWPRE.ReKeyGen(sk_A, pk_B): Given Alice's own secret key $\text{sk}_A = \mathbf{s}_A$ and Bob's public key $\text{pk}_B = A_B$, Alice generates a re-encryption key $\text{rk}_{A \rightarrow B}$ as $\text{rk}_{A \rightarrow B} = A_B X + \begin{pmatrix} \mathbf{s}_A G \\ \mathbf{0}_{(n-1) \times n\ell} \end{pmatrix}$, where X is a random binary matrix. Observe that the property $\mathbf{s}_B \cdot \text{rk}_{A \rightarrow B} \approx \mathbf{s}_A G$ holds.
- GSWPRE.Enc(pk_A, m): To encrypt a given a message m under Alice's public key pk_A , choose a random matrix $R \leftarrow \{0, 1\}^{n\ell \times n\ell}$ and compute the ciphertext as $C = A_A R + mG \in \mathbb{Z}_q^{n \times n\ell}$, such that the propriety $\mathbf{s}_A C \approx m \mathbf{s}_A G$ holds.
- GSWPRE.ReEncrypt($\text{rk}_{A \rightarrow B}, C$): Given a ciphertext C and a re-encryption key $\text{rk}_{A \rightarrow B}$, a proxy can re-encrypt the ciphertext by computing $\hat{C} = \text{rk}_{A \rightarrow B} \cdot G^{-1}(C)$. Output the re-encrypted ciphertext as the penultimate column \hat{c} in \hat{C} .
- GSWPRE.Dec(sk_B, \hat{C}): Given a re-encrypted ciphertext \hat{c} and the corresponding secret key sk_B , Bob decrypts as $(\hat{c}, \mathbf{s}_B) \pmod{2}$.

ABE schemes encrypt data according to a set of attributes predefined by the data owner and often described as an *access policy structure*, such that the ciphertext can only be decrypted by a user whose attributes satisfy this policy. The IBE scheme can viewed as a special type of ABE scheme where the sole defined attribute is the identity of the user. Henceforth, we focus our descriptions on the ABE scheme.

An ABE scheme has four core cryptographic functions, $\text{ABE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$. A trusted central authority, acting as an attribute authority, generates a master key pair (mpk, msk) in the Setup function and uses the master secret key msk to derive decryption keys in the KeyGen for authorized users after checking their attributes. Data owners and users interact with this attribute authority to obtain authorized key based on the attributes to perform encryption and decryption. Fig. 6 shows an example ABE scheme where a data owner encrypts their data and defines the access policy stating that “*the ciphertext can be only decrypted by a user who is both a teacher and in the computer science department*”. In general, there are two types of ABE schemes based on where the access structure is defined, Key-policy (KP-ABE) [62] and Ciphertext-policy (CP-ABE) [11]. In CP-ABE, the ciphertext is encrypted based on the access policy, and the secret key is generated based on the attributes as illustrated in Fig. 6. In KP-ABE, the ciphertext is encrypted based on a set of attributes, and the user's secret key is generated based on a defined access policy.

The first construction of ABE was proposed by Sahai and Waters [86] as a fuzzy form of identity-based encryption (IBE), in which data are encrypted under a key derived from user's identity [15, 90]. The fuzzy IBE scheme (FIBE) realizes access control through combined techniques of Shamir's linear secret sharing [89] and Lagrange polynomial interpolation. The construction is based on Bilinear maps, where \mathbb{G}_1 and \mathbb{G}_2 are two groups of prime order p . The group \mathbb{G}_1 has a generator g and a defined bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, such that for all $a, b \in \mathbb{Z}_p$, we have $e(g^a, g^b) = e(g, g)^{ab}$. The

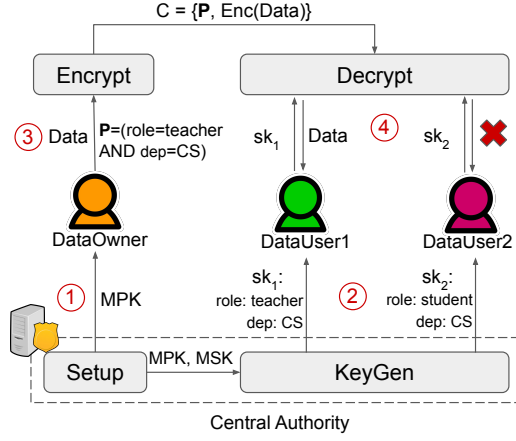


Fig. 6. Illustration of CP-ABE framework where the access policy is embedded in the ciphertext

security of the scheme is based on the Decisional Bilinear Diffie-Hellman assumption (DBDH) [13], which states that given a group generator g and uniformly chosen values $a, b, c \in \mathbb{Z}_p$, then no polynomial-time adversary is able to distinguish the tuple $(g^a, g^b, e(g, g)^{ab})$ and $(g^a, g^b, e(g, g)^c)$ with non-negligible advantage. We briefly describe each function in the FIBE scheme in Scheme 7.

This scheme enables a generalized controlled access to encrypted data when there are at least d overlapping attributes in the ciphertext and the secret key. Although the attributes are public, the access control is still cryptographically enforced by the underlying security assumption. Note that each attribute in \mathcal{U} has a corresponding unique secret value t_i in the master secret key. The attribute authority issues a secret key for a user depending on their provided attributes, such that it computes the value $D_i g^{\frac{q(i)}{t_i}}$ for the i -th attribute based on the master secret key. Hence without direct access to msk, it is not possible for an external adversary to forge a new secret key with desired attributes without knowing the corresponding secret values t_i . The secret values cannot be retrieved from the public parameters g^{t_i} due to the fundamental hardness of the discrete log, which the DBDH builds upon. Moreover, an existing unauthorized user cannot modify their secret key to selectively add attributes to their set ω to satisfy $|\omega \cap \omega'| \geq d$, and decrypt a ciphertext.

Traditional ABE schemes provide access control to encrypted messages. Homomorphic ABE (ABHE) schemes can further support evaluations on these ciphertexts, which adds a fifth function, Eval to the ABE functions. Gentry et al. [56] proposed a leveled homomorphic ABE based on LWE. The scheme is based on an earlier work by Gorbunov et al. [61] which uses a variation of garbled circuits as the access policy structure, but it is the first ABE scheme that also supports computation on the encrypted data. Mainly, let ℓ be the number of defined attributes, and C be a garbled circuit with ℓ input wires and one output wire, each input wire corresponds to the encoded input of two independent public/secret key pair. The scheme uses an encoding mechanism [61] with a random seed to encode each input wire and use the encoded output of the circuit as a masking value when encrypting the message. Each authorized user receives a secret key which embeds a set of recoding keys associated with their attributes and corresponding to those used in the encryption. These recoding keys are similar to a translation table for a garbled circuit, such that the decryption is only possible when a user is able to recreate the encoded value which was used to mask the message.

Any ABE scheme has to be collision-resistant to ensure security of the data. In other words, users cannot combine their decryption keys to gain access to encrypted data when none of them is

Scheme 7: The Fuzzy Identity-based Encryption (FIBE) scheme [86]

- FIBE.Setup(λ): This function is performed by a trusted third party (TTP), acting as the Attribute Authority, and takes as an input the security parameter λ which determines the size of inputs. It defines a universe of attributes \mathcal{U} , randomly chooses a unique value t_i where $i = 1, \dots, |\mathcal{U}|$ to corresponds to each defined attribute, and chooses a secret value y . The outputs of this function are the public key $\text{mpk} = \{\{T_i = g^{t_i}\}_{i \in 1, \dots, |\mathcal{U}|}, Y = e(g, g)^y\}$ and the master key $\text{msk} = \{t_1, \dots, t_{|\mathcal{U}|}, y\}$. Both the universe of attributes \mathcal{U} and the public key mpk are published, but the master key msk is kept secret with the trusted Attribute Authority.
- FIBE.Enc(pk, ω', m): This function takes as input a message m , a set of attributes $\omega' \subset \mathcal{U}$, and the public key mpk . It chooses a random $r \in \mathbb{Z}_p$ for semantic security and outputs the ciphertext $c = (\omega', mY^r, \{T_i^r\}_{i \in 1, \dots, |\omega'|})$.
- FIBE.KeyGen(msk, ω): This function is performed by the TTP and generates a secret key sk associated with a given set of user's attributes ω . Specifically for each user, it defines an independent polynomial $q(x)$ of degree $d - 1$ and $q(0) = y$, where y is a secret defined in msk . The function outputs $\text{sk} = \{D_i = g^{\frac{q(i)}{t_i}}; i = 1, \dots, |\omega|\}$.
- FIBE.Dec(sk, ω, c): This function takes as input a secret key sk and a ciphertext c to decrypt based on Lagrange interpolation and secret sharing techniques. The decryption is successful if and only if $|\omega \cap \omega'| \geq d$, i.e., the user has at least d attributes in ω matching the attributes ω' embedded in the ciphertext. Let a set $s = \omega \cap \omega'; |s| = d$ be arbitrary elements which can reconstruct the secret element y , and $\Delta_{i, s[0]}$ are defined Lagrange coefficients, then the ciphertext is decrypted as:

$$\begin{aligned}
 m &= mY^r / \prod_{i=1, \dots, |s|} (e(g^{\frac{q(i)}{t_i}}, g^{r t_i}))^{\Delta_{i, s[0]}} \\
 &= me(g, g)^{r y} / \prod_{i=1, \dots, |s|} (e(g, g)^{r q(i)})^{\Delta_{i, s[0]}} \\
 &= me(g, g)^{r y} / e(g, g)^{r y} = m
 \end{aligned}$$

initially authorized. This can be ensured by choosing an independent polynomial embedding unique random elements for each user's share (secret keys) as in [62, 86], or by using a masking technique which uniquely randomizes the user's secret key as in [11]. In the homomorphic ABE [61], the collision is not possible since each user's secret key hides a different "translation table" and hence even if two users collude, they will not be able to decrypt if none of them can independently satisfy the circuit predicate.

Revocation of user's secret key is another important requirement for ABE schemes. One naive revoking approach is to generate new public key and master key and update all users' secret keys except for the revoked user's. However this approach is infeasible in practice since it requires invoking the setup and key generation functions, and re-encrypting the existing ciphertexts with the new public key. Alternatively, the scheme can include time stamps [82] as additional attributes which state an expiry date TS for each generated secret key and an encryption date TS' for each ciphertext. Hence, a user can decrypt only if their secret key's expiry date is beyond the ciphertext's last decryption date (or $TS \geq TS'$).

Homomorphic ABE/IBE schemes provide means for data owners to control the access to their encrypted messages. Homomorphic evaluations can be performed but only on ciphertexts encrypted under the same identity or attributes. Several works [37] in the literature proposed multi-identity

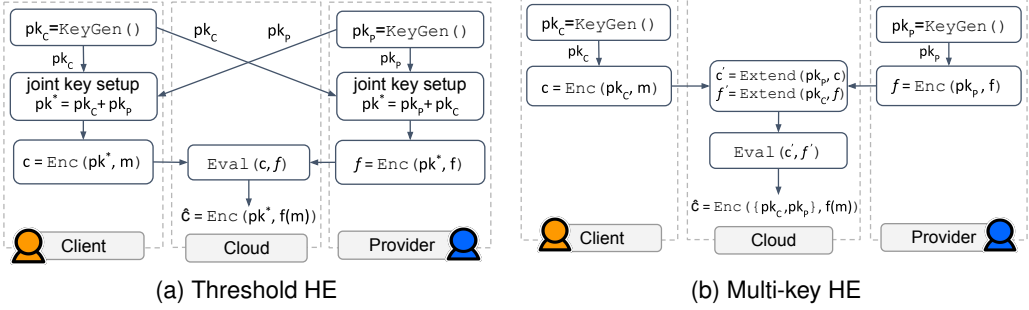


Fig. 7. Encryption step of schemes supporting multiple keys.

IBE schemes, which support computation on ciphertexts under different identities. However, the users with those identities are required to cooperate in a distributed decryption. The design of these schemes is similar in the design to multi-key HE schemes, discussed in the subsequent section, which support homomorphic evaluations on ciphertexts encrypted under different keys.

5 MULTI-KEY APPROACHES

Many outsourced computations require homomorphic evaluations on data provided by different owners and encrypted using their own keys. For example in Fig. 1c, the function, or trained model, f takes two inputs $[x]_A$ and $[y]_B$ encrypted under Alice’s and Bob’s public keys, respectively. Single-key approaches, discussed in previous section, enable re-encryption or decryption of ciphertexts with a different key, but computations *must* be performed under the same key. Multi-key approaches extend many base HE schemes to support homomorphic evaluation with data encrypted by multiple keys.

In general, HE schemes can be extended to *threshold* or *multi-key* settings, or a *hybrid* of both. In threshold HE schemes (ThHE), participants generate a joint public key pk^* in advance from their individual public keys, e.g., a linear combination of their public keys as shown in Fig. 7a. Inputs are homomorphically encrypted and evaluated under this joint key pk^* . In contrast, multi-key HE schemes (MKHE) support “on-the-fly” evaluation under different keys without a prior key setup as shown in Fig. 7b. A hybrid approach utilizing both ThHE and MKHE techniques can be used to improve practicality in special scenarios, where a group of system users always participate in evaluation.

In all three approaches, schemes are often designed in the common reference string (CRS) model as discussed in Sec. 3.1. This means the participants have access to a public parameter in the form of a ring element and use it to generate their individual keys. This produces individual keys that are related to each other and ensures correct computation with multiple keys. Moreover, in all three approaches, decryption is commonly done in a distributed manner among participants because no single user has the corresponding decryption key (i.e., the aggregation and concatenation of individual secret keys for ThHE and MKHE, respectively).

We review in this section the fundamentals and challenges of the state-of-the-art approaches and discuss their security and efficiency. For simplicity, we describe the notations use throughout this section in Table 3.

Table 3. Notations specific to multi-key approaches.

Category	Notation	Description
ThHE-specific	N	Total number of users in the system.
	T	Required number (threshold) of secret shares to decrypt.
	pk^*	A joint public key generated from individual public keys $\{pk_1, \dots, pk_N\}$
	sk^*	A joint secret key, secretly shared among N users.
	ek^*	A joint evaluation key generated corresponding to pk^* .
MKHE-specific	\mathcal{K}	A bound on number of keys we can extend a ciphertext under.
	\bar{K}	Total number of keys, such that $K \leq \bar{K}$
	\bar{pk}	An extended public key concatenated as (pk_1, \dots, pk_K)
	\bar{sk}	An extended secret key concatenated as (sk_1, \dots, sk_K)
	\bar{ek}	An extended evaluation key generated corresponding to \bar{pk} .
	\bar{c}	An extended ciphertext encrypted under the concatenated public keys.
Distributed decryption	ρ_i	The decryption component constructed by the i -th user in distributed decryption.

5.1 Threshold Homomorphic Encryption

Supporting homomorphic computations on inputs from different data owners without compromising data privacy requires data to be encrypted under individual's keys. Threshold encryption is a common approach for fulfilling this requirement, with the generation of a joint key in a key setup before any computation. This joint key can be changed or revoked at any point but with the cost of rerunning the key setup and re-encrypting all input data.

In a threshold encryption scheme [8, 14, 43, 44], a set of users can encrypt their data under a joint key but have to cooperate for decryption. Shamir's secret sharing [89] forms the base of threshold schemes. Mainly, a secret s is divided into shares s_i and distributed among N users, such that no single user knows the secret s . The secret is reconstructed if and only if a predefined number of shares (say a threshold T) are provided. This threshold encryption scheme follows the design of Shamir's (T, N) threshold scheme.

The threshold value T is chosen according to the security model and requirements. For example, if all N users in the system are required to participate in decryption, the threshold T is set to $T = N$. This setting follows a *dishonest-majority* assumption, where $T \geq N/2$ system users may be corrupted. In this case, even with $N - 1$ corrupted users, it is not possible to decrypt. This assumption can be relaxed to a *honest-majority* assumption, which implies that the number of dishonest users is less than $N/2$ and a subset of users can decrypt. On the other hand, if every user in the system has the right to independently decrypt, the threshold is set to $T = 1$ which corresponds to threshold *broadcast* encryption [27, 42, 57].

A ThHE scheme combines threshold functionality with the ability to compute on encrypted data under a joint key. There are three main components in ThHE schemes: a distributed key setup protocol, a homomorphic encryption function, and a distributed decryption protocol [88]. Formally, we can define a general ThHE scheme as tuple of PPT algorithms $\text{ThHE} = (\text{Setup}, \text{JointKeyGen}, \text{Enc}, \text{Eval}, \text{Dec})$.

- $\text{ThHE.Setup}(1^\lambda) \rightarrow ((pk_1, sk_1) \dots, (pk_N, sk_N))$: Given a security parameter λ , the setup algorithm outputs a set of N key pairs (pk_i, sk_i) .
- $\text{ThHE.JointKeyGen}(pk_1, \dots, pk_N) \rightarrow pk^*, ek^*$: Given the input of N public keys (pk_1, \dots, pk_N) , the interactive algorithm outputs a joint public key pk^* and the evaluation key ek^* , if required.
- $\text{ThHE.Enc}(pk^*, m) \rightarrow c$: Given a joint public key pk^* and a message m , the encryption algorithm outputs a ciphertext c .
- $\text{ThHE.Eval}(pk^*, f, c, c') \rightarrow c_{\text{eval}}$: Given a joint public key pk^* and two ciphertexts c, c' , the evaluation algorithm outputs the evaluated ciphertext $c_{\text{eval}} = f(c, c')$.

- ThHE.PartDec(sk_i, c) $\rightarrow \rho_i$: Given a ciphertext c encrypted under pk^* and a secret share of the key sk_i , perform the partial decryption algorithm and output a partially decrypted message ρ_i .
- ThHE.Combine(ρ_1, \dots, ρ_T) $\rightarrow m$: Given a set of partial decryptions (ρ_1, \dots, ρ_T) for some threshold set, combine decryption components to perform the final decryption and output the message m .

In earlier threshold schemes in the literature [39, 44], the joint key pair may be generated first by a trusted dealer, then the public key and shares of the secret key are distributed among users. Alternatively, the public key can be generated from users' individual public keys, as implied in ThHE.JointKeyGen, and they need to collaborate to decrypt with their individual secret keys at the end of computation. The two algorithms ThHE.PartDec and ThHE.Combine are the routines performed in distributed decryption, which we discussed as a common technique in Sec. 3.3.3. Another way to decrypt may utilize the key switching technique (Sec. 3.3.1) to re-encrypt the ciphertext result from an encryption under the joint key to one under the intended user's key. This method avoids performing distributed decryption and enable the user to directly decrypt with their own secret key. But, this method can lead to security problem as discussed in Sec.4.1.5. Mouchet et al. [75] proposed two versions of key switching, depending on whether parties have access to the shares of the new secret or public key, based on their construction of multiparty HE cryptosystem. In the rest of this section, we survey ThHE schemes based on two defined security assumptions, dishonest-majority, and honest-majority.

5.1.1 Dishonest-majority threshold HE. Many HE schemes can be extended to a threshold setting by leveraging the additive homomorphism of the key space. This property enables the establishment of a joint key from individual keys that are generated and owned by individual users without a trusted party. By direct aggregation of the users' public keys, we can effortlessly set up a (N -out-of- N) threshold encryption scheme following the dishonest-majority model. In other words, threshold HE transforms the setting from computing with different keys to computing with a single joint key.

Both ElGamal and Paillier schemes can be extended to a threshold version [39, 44, 80]. The aggregation of public keys in the former is less complex than the latter. Specifically in the multiplicatively homomorphic ElGamal scheme, assume we have a set of N users where each user i has an independently generated key pair $(pk_i = g^{s_i}, sk_i = s_i)$, where g is a group generator. Then, the joint key is computed as $pk^* = \prod_{i=1}^N pk_i = g^{\sum_{i=1}^N s_i} = g^{s^*}$. A user can encrypt data under the generated joint public key pk^* with a uniform random r for semantic security as $c = (c_0, c_1) = (g^r, m \cdot (g^{s^*})^r)$. Homomorphic computations can be done on the ciphertext using the homomorphic primitives of the scheme. Decryption of ciphertexts must be performed with the help of all N users to remove the element g^{rs^*} from c_1 . In particular, each user i uses their own secret key s_i to construct a component $\rho_i = (c_0)^{-s_i} = (g^r)^{-s_i} = g^{-rs_i}$ such that combining these components yields $\sum_{i=1}^N \rho_i = g^{-rs^*}$. Then, each user i takes turn and uses their component ρ_i to partially remove their secret key s_i by computing $\hat{c}_1 \cdot \rho_i = m \cdot g^{rs^*} \cdot g^{-rs_i}$. After partial decryption by all parties, we yield a plaintext message m .

In a similar manner, (R)LWE-based HE schemes can also be extended to a threshold setting. Asharov et al. [8] proposed a threshold variant of the BGV scheme [22] that we present in Scheme 8. The scheme is designed as a 3-round MPC protocol, where the joint public key pk^* and the joint evaluation key ek^* are generated in the first two rounds, and the distributed decryption is performed in the third round. Formally, given a set of N public keys $pk_i = (b_i, a) = (as_i + te_i, a)$, where the element $a \in R_q$ is a shared element in the CRS model and s_i is the i -th user's secret key. We generate a joint public key $pk^* = (\sum_{i=1}^N b_i, a)$ such that we obtain $(a \sum_{i=1}^N s_i + t \sum_{i=1}^N e_i, a) = (as^* + te^*, a)$. Note that only the first component b of the individual public keys is aggregated such that the underlying

secret keys s_i are homomorphically added under the RLWE assumption. If we add both components of all the public keys, the joint key will be $pk^* = (as^* + te^*, Na)$. Subsequently, a ciphertext encrypted under the joint key will be $c = (c_0, c_1) = (ras^* + te^* + m, Nra)$. Obviously, the decryption $c_0 - c_1s^*$ will fail because $(Nras^* \neq ras^*)$.

Scheme 8: A Threshold-variant of BGV scheme [8]

- ThBGV.Setup($1^\lambda, 1^L$): Given the security parameter λ and a multiplicative depth L , run BGV.Setup and output $pp = (R, \chi, \psi, \mathcal{B}, q, t, \mathbf{a})$ as the public parameters of scheme. Let \mathbf{a} be the shared CRS.
- ThBGV.KeyGen(pp) $\rightarrow ((pk_1, sk_1), \dots, (pk_n, sk_n))$: Given the public parameters pp , uniformly sample a set of N secrets $s_i \leftarrow \psi$ and errors $e_i \leftarrow \chi$, where $i \in \{1, \dots, N\}$ and output the set of key pairs (pk_i, sk_i) such as $pk_i = (as_i + te_i, a)$ and $sk_i = s_i$.
- ThBGV.JointKeyGen(pk_1, \dots, pk_N) $\rightarrow pk^*, ek^*$: Given the input of N public keys (pk_1, \dots, pk_N) , aggregate the first component of pk_i to generate the joint public key as $pk^* = (\sum (as_i + te_i), a)$. Also, compute the evaluation key $ek^* = \{\text{Enc}(pk^*, \sum s^*[i]s^*[j])\}$.
- ThBGV.Enc(pk^*, m) $\rightarrow c$: Given a joint public key pk^* and a message m , the encryption algorithm outputs a ciphertext $c = (c_0, c_1)$ where $c_0 = r(as^* + te^*) + m$ and $c_1 = ra$.
- ThBGV.Eval(pk^*, f, c, c') $\rightarrow c_{\text{eval}}$: Given a joint public key pk^* and two ciphertexts c, c' , the evaluation algorithm outputs the evaluated ciphertext $c_{\text{eval}} = f(c, c')$. Similar to the base BGV scheme, evaluation can be either homomorphic addition or multiplication. The latter requires the joint evaluation key ek^* to perform relinearization via key switching.
- ThBGV.PartDec(sk_i, c) $\rightarrow \rho_i$: Given a ciphertext c encrypted under pk^* and a secret share of the key sk_i , generate a decryption component as $\rho_i = (c_1s_i + te'_i)$.
- ThBGV.Combine(ρ_1, \dots, ρ_N, c) $\rightarrow m$: Given decryption components (ρ_1, \dots, ρ_N) , decrypt the ciphertext as $(c_0 - \sum \rho_i) = (c_0 - c_1 \sum s_i) = (c_0 - c_1s^*) = \tilde{m} \bmod t = m$.

As mentioned, homomorphic multiplication requires additional evaluation keys to perform relinearization that brings the quadratic ciphertext in $(s^*)^2$ back to be linear in s^* . Recall that the evaluation key encrypts the powers of base of the secret key $(s^*)^2 = (s_1 + \dots + s_N)^2$. Since the secret key is shared among N users, they cannot simply calculate the sum of the square of their individually generated secret keys s_i , because $(s_1^2 + \dots + s_N^2) \neq (s_1 + \dots + s_N)^2$. The generation of this evaluation key ek^* is trickier than the public key due to its fundamentally complex structure. It requires all N parties to cooperate in a 2-round setup phase to compute the joint evaluation key from their secret shares. Let $\alpha, \beta \in \{0, d-1\}$ be the indexes of coefficients of the secret s^* , which is a polynomial with degree $d-1$, and let $\ell \in \{0, \lfloor \log q \rfloor\}$. In the first round, every party will broadcast $\{\text{Enc}(pk^*, t^\ell s_i[\alpha])\}$, which is a set of encryption of all the powers of base t of the i -th secret share's coefficients. Note if we aggregate the shares from all parties, we will obtain $\{\text{Enc}(pk^*, t^\ell \cdot s^*[\alpha])\}$. In the second round, each party performs a pairwise multiplication between the coefficients of the aggregated encrypted secret key s^* and the coefficients of its secret shares s_i , yielding $ek^* = \{\text{Enc}(pk^*, t^\ell \cdot s^*[\alpha] \cdot s_i[\beta])\}$. Combining the shares from all parties in the second round outputs the joint evaluation key $\{ek^*\}$, such that for $a \leftarrow R_q$, the evaluation key is $ek^* = (as^* + te + t^\ell \cdot s^* \cdot s^*, a)$. To avoid making circular security assumption, the secret $(s^*)^2$ should be encrypted under a different secret s^* , hence we can generate L joint keys, one for each level $l \in L$, and encrypt $(s_{l-1}^*)^2$ corresponding to level $l-1$ under pk_l^* as discussed in Sec. 4.1.1.

To decrypt, each user contributes their partial key s_i by computing the component $\rho_i = c_1s_i + te_i$. Then, all users collaboratively produce a component that contains the sum of all secret keys shares,

that is $\sum_{i=1}^N \rho_i = (c_1 \sum_{i=1}^N s_i + t \sum_{i=1}^N e_i) = (c_1 s^* + t e^*) = \rho^*$. Refer to the decryption process $\text{Dec}(\text{sk}, c) = c_0 - \rho^*$, the message can be decrypted correctly when computing $c_0 - (c_1 s^* + t e^*)$.

These (N, N) threshold schemes enable the group of N users to homomorphically compute on their data that is encrypted under a joint key from their individual keys. Decryption however is impossible unless all of them participate with their secret shares. In practice, this may become a single point of failure if a user becomes uncooperative or was offline at the time for decryption. Hence, designing a more flexible threshold scheme is necessary where only a subset of secret shares is sufficient to decrypt.

5.1.2 Honest-majority threshold HE. Threshold HE scheme proposed by Asharov et al [8] targets a dishonest-majority security assumption. However, some applications may need to relax this assumption to an honest-majority one to avoid single point of failure by enabling only a subset of keys to decrypt. Desmedt and Frankel [44] proposed a threshold version of ElGamal scheme [48] where only T users out of N are needed to decrypt. The scheme leverages Shamir's linear secret sharing [89]. Given a threshold T and a large prime p as parameters, a *trusted* dealer performs a key setup as follows. The dealer uniformly sample a secret key $s^* \bmod p$ and set it as the constant term of a random polynomial $f(x) \bmod p$, such that $f(0) = s^*$. The degree of this polynomial is determined as $T - 1$ if we want T users to reconstruct the secret. The dealer then uses this polynomial to generate N secret shares $s_i = f(i) \bmod p$ as unique points for each user $i \in \{1, N\}$. Finally, broadcast $\text{pk}^* = g^{s^*}$ as the joint public key and keep both s^* and $f(x)$ secret from the users. The intuition is that Lagrange interpolation can be used to reconstruct the secret s^* if at least T points (shares) were provided.

Each user can encrypt a message m_i as $(a_i, b_i) = (g^{r_i}, m_i g^{r_i s^*})$ using the public key g^{s^*} and a random value r_i . However, at least T users are required to use their secret shares to decrypt. This construction of threshold ElGamal requires a trusted dealer to generate and distribute the secret in the key setup. Pedersen [80] improved the scheme later by removing the need for this trusted dealer at key setup and enabled validation of the secret shares for robustness. However, the decryption still requires a trusted key authority to generate the unique polynomial and decrypt.

Threshold schemes proposed after that were based on different encryption schemes [38, 93], not necessarily homomorphic, and removed the need of this trusted dealer by relying on Diffie-Hellman key generation instead, such as in threshold RSA [41, 83].

Boneh et al [16] proposed the first $(T\text{-out-of-}N)$ ThHE scheme based on LWE problem. In this scheme, the decryption key is split into N shares among the participants. Shamir's linear secret sharing scheme (LSSS) technique is also leveraged here to allow T key shares to sufficiently decrypt according to a defined threshold access structure. Moreover, a *universal thresholdizer* was proposed to extend general HE schemes to threshold HE schemes. Specifically, a universal thresholdizer split a given cryptographic key into N valid shares that can be used as individual public keys.

5.1.3 Discussion. Key management in ThHE schemes is essential to system security. In the key setup, the joint key is often generated based on participants' individual keys before any computation. Each user encrypts their inputs under this one joint key to produce compact ciphertexts, which means the ciphertext size is *independent* of the number of users N . The security depends on the fact that no single user has the decryption key, which can be only constructed if a threshold set of users cooperated. The joint key can be revoked when a user is added or removed at any point, but this requires running the threshold key setup again and re-encrypting all inputs with the new joint key. It may be a better choice to leverage ThHE schemes in outsourced computations if participants do not change often. If they do, the efficiency quickly degrades because the joint key needs to be updated accordingly. In this case, a more flexible approach is needed to allow computation with multiple keys "on-the-fly" when required.

Potentially, we investigate if the key homomorphism can be used to directly transform ciphertexts from encryptions under individual keys to ones encrypted under a joint key without prior key setup. For example in the threshold BGV scheme [8], let $[m]_A = (c_0, c_1) = (ra(s_A) + te_A + m, ra)$ be a ciphertext encrypting a message m under Alice's public key pk_A . A cloud evaluator may perform joint homomorphic evaluation on this ciphertext for Alice and Bob such that the result is encrypted under their joint key pk_{AB} . Using Bob's public key $pk_B = (as_B + te_B, a)$, the cloud can construct a component $\xi_B = (ras_B + te_B)$ which is the encryption of zero if the randomness r is known. Then, the cloud can add this component to the ciphertext as $[m]_{AB} = (c_0 + \xi_B, c_1) = (ra(s_A + s_B) + t(e_A + e_B) + m, ra)$, which is an encryption under pk_{AB} .

Unfortunately, this approach has two major flaws which breaches the security of the scheme. First, the encryption randomness r may need to be *fixed* and shared with the cloud similar to the CRS element a . Recall that the BGV scheme encryption algorithm depends on r to randomize the ciphertexts. Fixing this element renders deterministic encryption scheme which loses its semantic security property. i.e., encrypting the same message at different times will yield the same ciphertexts. The second flaw is more vital since this technique relies on *disclosing* the randomness r to the cloud evaluator. Obviously, this break the security of the scheme. Specifically, given a ciphertext $[m]_A$, the cloud evaluator can easily decrypt it by constructing a component (ras_A) using the public key pk_A and compute $\tilde{m} = c_0 - ras_A = m + te_A \pmod{q}$, which removes the large element ras_A and yield $m = \tilde{m} \pmod{t}$.

Due to these security issues, ThHE cannot be simply modified to support on-the-fly computation with different keys. Alternatively, a multi-key technique targets this problem and extends HE schemes to dynamic transformation of ciphertexts by concatenating the different public keys and treating them as one key. We review this technique in depth in the following section.

5.2 Multi-key Homomorphic Encryption

The first notion of multi-key (MKHE) schemes was introduced by López-Alt et al. [70] to support the homomorphic evaluation on ciphertexts encrypted under different keys. Compared to ThHE schemes, this type of HE schemes remove the need of a key setup phase to generate a joint key from individual keys prior to any computation. Instead, a cloud evaluator can dynamically extend ciphertexts from encryption under individual keys to ones under the concatenation of individual users' keys (pk_1, \dots, pk_K) . However, both ThHE and MKHE schemes are similar in requiring users to cooperate and run a distributed decryption protocol when retrieving the evaluated result.

We categorize MKHE scheme as *single-hop* if the extended ciphertext cannot be further extended to additional keys after being homomorphically evaluated. Otherwise, we categorize them as *multi-hop* MKHE schemes. In general, an MKHE scheme is a tuple of PPT algorithms $MKHE = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Extend}, \text{EvalKeyGen}, \text{Eval}, \text{Dec})$. We formally define each algorithm as follow.

- $MKHE.\text{Setup}(1^\lambda, 1^K) \rightarrow pp$: Given a security parameter λ and a bound \mathcal{K} on the number of keys, the setup algorithm outputs the public parameters pp .
- $MKHE.\text{KeyGen}(pp) \rightarrow (pk, sk)$: Given the public parameters pp , the key generation algorithm outputs a public key pk , a private key sk .
- $MKHE.\text{Enc}(pk, m) \rightarrow c$: Given a public key pk and a message m , the encryption algorithm outputs a ciphertext c .
- $MKHE.\text{Extend}(\{pk_1, \dots, pk_K\}, c) \rightarrow \bar{c}$: Given a set of K public keys pk_1, \dots, pk_K where $K < \mathcal{K}$, and a ciphertext c , the output is the extended ciphertext \bar{c} under the concatenated public key \bar{pk} .
- $MKHE.\text{EvalKeyGen}(\bar{pk}) \rightarrow \bar{ek}$: Given a concatenated public key \bar{pk} generate the corresponding evaluation (linearization) key \bar{ek} .

- $\text{MKHE.Eval}(\overline{pk}, f, \bar{c}, \bar{c}') \rightarrow c_{\text{eval}}$: Given two extended ciphertexts \bar{c}, \bar{c}' under the same concatenated public key \overline{pk} , the evaluation algorithm outputs the evaluated ciphertext $\bar{c}_{\text{eval}} = f(\bar{c}, \bar{c}')$.
- $\text{MKHE.Dec}((sk_1, \dots, sk_K), \bar{c}) \rightarrow m$: Given a set of concatenated secret shares (sk_1, \dots, sk_K) and an extended ciphertext \bar{c} , the interactive decryption algorithm performs decryption and outputs the message m

To design an MKHE scheme, two main points have to be addressed: the generation of the evaluation key and maintaining the encryption randomness. Homomorphic multiplication on the extended ciphertexts requires providing an evaluation key corresponding to the concatenated key to perform dimension reduction (i.e., relinearization). Also, decryption in its essence depends on canceling out the large randomized elements in the ciphertext to retrieve the message. Hence, we must ensure that decrypting with different secret keys is done correctly in MKHE schemes, i.e., it does not yield remaining randomized elements.

5.2.1 MKHE via onion encryption. In theory, any standard HE scheme can be extended to multi-key for a *constant* number of keys through an *onion* encryption and decryption technique [70]. This technique is similar to onion routing [59] used in anonymous communication in Tor [45]. López-Alt et al. [70] shows an HE construction for encrypting a message m under multiple keys. Given an HE scheme which encrypts a message $m \in \mathbb{Z}_t$, say $t = 2$, into a ciphertext $c \in \mathbb{Z}_q$. In this scenario, let m be bit-decomposed such that $m = (m_1, \dots, m_\ell) \in \{0, 1\}^\ell$ for a bit length ℓ . Define the bit-wise encryption of a message m and the decryption of ciphertext c as following.

$$\begin{aligned}\overline{\text{Enc}}(\text{pk}, m) &= (\text{Enc}(\text{pk}, m_1), \dots, \text{Enc}(\text{pk}, m_\ell)) \\ \overline{\text{Dec}}(\text{sk}, c) &= (\text{Dec}(\text{sk}, c_1), \dots, \text{Dec}(\text{sk}, c_\ell))\end{aligned}$$

Now, define the onion encryption and decryption for a message m under a set of $K \in \mathbb{N}$ keys as following.

$$\begin{aligned}\text{Enc}^*(\text{pk}, m) &= \overline{\text{Enc}}(\text{pk}, m) \\ \text{Enc}^*(\text{pk}_1, \dots, \text{pk}_K, m) &= \text{Enc}^*(\text{pk}_K, \text{pk}_{K-1}, \dots, \overline{\text{Enc}}(\text{pk}_1, m)) \\ &= \overline{\text{Enc}}(\text{pk}_K, \overline{\text{Enc}}(\text{pk}_{K-1}, \dots, \overline{\text{Enc}}(\text{pk}_1, m))) \\ \text{Dec}^*(\text{sk}, c) &= \overline{\text{Dec}}(\text{sk}, c) \\ \text{Dec}^*(\text{sk}_1, \dots, \text{sk}_K, c) &= \text{Dec}^*(\text{sk}_{K-1}, \dots, \text{sk}_1, \overline{\text{Dec}}(\text{sk}_K, c)) \\ &= \overline{\text{Dec}}(\text{sk}_1, \overline{\text{Dec}}(\text{sk}_2, \dots, \overline{\text{Dec}}(\text{sk}_K, c)))\end{aligned}$$

Figure 8 shows an example of this HE scheme which adds encryption layers repeatedly to extend ciphertexts to additional keys. Retrieving the result is done by applying decryption in reverse order of encryption using the set of corresponding secret keys.

Before adding a new encryption layer, the ciphertext is represented in binary as $c \in \{0, 1\}^\mu$, where $\mu = \text{poly}(\lambda)$ for some polynomial $\text{poly}(\cdot)$. Each layer of encryption expands the ciphertext size by a factor of μ because the encryption function $\overline{\text{Enc}}(\text{pk}, m)$ outputs a bit-wise encrypted ciphertexts. Hence, the size of a ciphertext grows exponentially with each additional encryption layer. Specifically, in the case where a message space $\{0, 1\}$ and a ciphertext space $\{0, 1\}^\mu$, the size of a ciphertext encrypted under K keys is approximately μ^K . Due to this large ciphertext expansion after each layer, onion encryption method is feasible only for a very small number of keys, which must be chosen before computation.

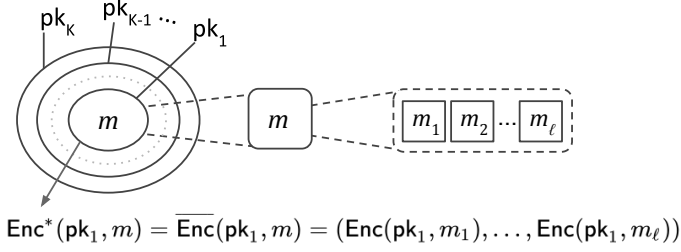


Fig. 8. Achieving MKHE via onion encryption.

Given K ciphertexts $c_i = \overline{\text{Enc}}(pk_i, m_i)$, we can extend them to be encryptions under K key via onion encryption described above. Homomorphic operations are performed on ciphertexts encrypted under the same set of keys but must be *in order*. The result is decrypted with the corresponding ordered secret keys such that $\text{Dec}^*(sk_1, \dots, sk_K, f(m_1, \dots, m_K)) = f(m_1, \dots, m_K)$.

As mentioned before, this technique is generic to extend any standard HE scheme to the multi-key setting. It proposes the notion of on-the-fly computation, but a minimum setup is required before computation; i.e., choosing the number of keys K as a parameter. Onion encryption shares the setup requirement with ThHE schemes. A key setup phase must be done in the ThHE to generate the joint key from participants' keys, and in onion encryption the number of keys is selected. However, onion encryption has a restriction on the order of evaluations and decryptions as they must be done in the order of the keys. In contrast, evaluations in ThHE are performed on data encrypted under the joint key, and decryption is done in a distributed manner that does not depend on the order.

Scheme 9: The López-Alt-Tromer-Vaikuntanathan (LTV) scheme [70]

- LTV.Setup(1^λ): Given a security parameter λ , choose a cyclotomic polynomial $\Phi(x) = x^d + 1$ and define $R = \mathbb{Z}[x]/(\Phi(x))$ as a polynomial ring of degree d with integer coefficients. Let R_q be the quotient ring R/qR where q is a modulus and χ be an error distribution. Output the set of parameters $\text{pp} = (\lambda, R_q, \chi, q)$.
- LTV.KeyGen(pp): Given a security parameter λ , sample $f', g \leftarrow \chi$. Compute $f = 2f' + 1$ such that $f \equiv 1 \pmod{2}$. If f is not invertible in R_q , re-sample. Otherwise, find f^{-1} and set $h = 2gf^{-1} \in R_q$. Output the key pair as $(\text{pk}, \text{sk}) = (h, f)$. Output the evaluation key $\text{ek} = hu' + 2e' + \text{PowersOf2}(f) \in R_q$, where $u', e' \leftarrow \chi$.
- LTV.Enc(pk, m): Given a public key $\text{pk} = h$ and a message $m \in \{0, 1\}$. Sample $u, e \leftarrow \chi$ and output the ciphertext as $c = hu + 2e + m \in R_q$.
- LTV.Eval($\text{pk}, f, c_1, \dots, c_\ell$): Given a set of public keys $\text{pk} = (pk_1, \dots, pk_K)$ and ℓ ciphertexts, perform the evaluation f and output \bar{c}_{eval} . Evaluation can be addition or multiplication. For addition, output $\bar{c}_{\text{add}} = \bar{c}_1 + \dots + \bar{c}_\ell \in R_q$. For multiplication, perform $\bar{c}_{\text{mult}} = \bar{c}_1 \cdots \bar{c}_\ell \in R_q$ and use evaluation keys $\{\text{ek}_1, \dots, \text{ek}_K\}$ for relinearization.
- LTV.Dec($(sk_1, \dots, sk_K), \bar{c}$): Given a set of secret keys $(sk_1, \dots, sk_K) = \{f_1, \dots, f_K\}$ and an extended ciphertext \bar{c} , perform the decryption as $z = f_1, \dots, f_n \bar{c} \in R_q$. Output the result as $m = z \pmod{2}$.

López-Alt et al. [70] proposed an MKHE based on NTRU HE scheme [96], which we describe in Scheme 9. Let $c_1 = h_1u_1 + 2e_1 + m_1$ and $c_2 = h_2u_2 + 2e_2 + m_2$ be two ciphertexts encrypted under

two public keys h_1, h_2 . We can add them and correctly decrypt to the sum using the joint secret keys of $f_1 f_2$ as long as the noise e_{add} is not too large. Also, note that $f_1 \equiv f_2 \equiv 1 \pmod 2$.

$$\begin{aligned}
f_1 f_2 (c_1 + c_2) &= f_1 f_2 h_1 u_1 + 2f_1 f_2 e_1 + f_1 f_2 m_1 + f_1 f_2 h_2 u_2 + 2f_1 f_2 e_2 + f_1 f_2 m_2 \\
&= f_1 f_2 (h_1 u_1 + h_2 u_2) + 2f_1 f_2 (e_1 + e_2) + f_1 f_2 (m_1 + m_2) \\
&= 2(f_2 g_1 u_1 + f_1 g_2 u_2 + f_1 f_2 e_1 + f_1 f_2 e_2) + f_1 f_2 (m_1 + m_2) \\
&= 2e_{\text{add}} + f_1 f_2 (m_1 + m_2) \\
&= m_1 + m_2 \pmod 2
\end{aligned}$$

Similarly, we can decrypt to the product using the joint secret keys $f_1 f_2$ as following.

$$\begin{aligned}
f_1 f_2 (c_1 c_2) &= f_1 f_2 (h_1 u_1 + 2e_1 + m_1)(h_2 u_2 + 2e_2 + m_2) \\
&= f_1 f_2 h_1 u_1 h_2 u_2 + 2f_1 f_2 h_1 u_1 e_2 + f_1 f_2 h_1 u_1 m_1 + 2f_1 f_2 h_2 u_2 e_1 + 4f_1 f_2 e_1 e_2 \\
&\quad + 2f_1 f_2 e_1 m_2 + f_1 f_2 m_1 h_2 u_2 + 2f_1 f_2 m_1 e_2 + f_1 f_2 m_1 m_2 \\
&= 4g_1 u_1 g_2 u_2 + 4f_2 g_1 u_1 e_2 + 2f_2 g_1 u_1 m_1 + 4f_1 g_2 u_2 e_1 + 4f_1 f_2 e_1 e_2 \\
&\quad + 2f_1 f_2 e_1 m_2 + 2f_1 m_1 g_2 u_2 + 2f_1 f_2 m_1 e_2 + f_1 f_2 (m_1 m_2) \\
&= 2e_{\text{mult}} + f_1 f_2 (m_1 m_2) \\
&= m_1 m_2 \pmod 2
\end{aligned}$$

A naive decryption in LTV requires keys to be constructed based on the evaluated circuit. For example, the ciphertext $c_1^2 c_2$ is decrypted by multiplying it with the keys $f_1^2 f_2$, and $c_1 c_2 + c_2 c_3$ is decrypted with the keys $f_1 f_2^2 f_3$. In other words, the power of the secret key f_i corresponds to the times the ciphertext c_i is used in the evaluated circuit. This impacts the efficiency of the scheme since the size of the decryption key grows based on both the number of involved keys and the depth of the circuit. Instead, they apply key switching on evaluated ciphertexts to make keys independent of the circuit. Specifically, the technique makes a ciphertext decryptable with f_i instead of f_i^2 , e.g., the ciphertext $c_1 c_2 + c_2 c_3$ is decrypted with $f_1 f_2 f_3$ instead of $f_1 f_2^2 f_3$.

After the LTV scheme, many works propose MKHE constructions with different capabilities and security assumptions. We provide a comparison of these different schemes in Table 4. Observed from the table, we can improve the ciphertext size from growing quadratically in early works [46, 70, 76, 81] to growing linearly with the number of keys in the recent works [23, 30, 32, 67]. The majority of schemes are multi-hop, which means an extended ciphertext can be further extended to additional keys after homomorphic evaluations. Most schemes [30, 32, 76, 81] also require key generation to use a public parameter in the CRS model. Moreover, the RLWE-based MKHE schemes [4, 30, 32] support packing multiple messages in one ciphertext which enables SIMD evaluations.

5.2.2 Single-hop MKHE. Clear and McGoldrick [37] proposed a compiler for a multi-identity homomorphic IBE scheme based on the GSW scheme [56]. As discussed in Sec. 4.3, this compiler produces ciphertexts that are encrypted under different identities, instead of only supporting single-identity in the original GSW scheme. Identities can be viewed as an analog to keys in HE schemes; hence, we can also obtain a multi-key HE scheme in this way. Mukherjee and Wichs [76] improved this IBE-based MKHE and built a general two-round MPC protocol on top of it, including the first introduction of a detailed distributed decryption protocol. We describe the latter multi-key variant of the GSW in Scheme 10.

Compared to the LTV scheme [70], the new GSW-based scheme *does not* limit the maximum number of keys the ciphertext can be extended to. But, the ciphertext *cannot* be further extended after performing a homomorphic evaluation.

Table 4. Comparison between proprieties of existing different MKHE schemes.

Scheme	Security	Ctxt size growth	Protocol rounds	Multi-hop	Bounded # of keys	CRS	Packing	Need bootstrapping
LTV12 [70]	NTRU	Quadratic	-	✓	✓			
CM15 [37]	LWE	Quadratic	-			✓		
MW16 [76]		Quadratic	2			✓		
PS16 [81]		Quadratic	2	✓	✓	✓		
BP16 [23]		Linear	2	✓	✓	✓		✓
DHRW16 [46]		piO	Quadratic	2	✓			
CZW17 [32]	RLWE	Linear	2	✓	✓	✓	✓	
YKHK18 [105]		Linear	2	✓	✓	✓	✓	
LZYH+19 [67]		Linear	2	✓	✓	✓	✓	
CDKS19 [30]		Linear	2	✓	✓	✓	✓	
AH19 [4]		Constant	4 *	✓	✓	✓	✓	
CCS19 [29]	TLWE	Linear	2	✓	✓	✓		✓

* Two extra rounds are for the ThHE setup phase to generate the joint key.

To understand how the scheme works, let us consider the following toy example. Let $C_1 = A_1R_1 + m_1G$ be a fresh GSW ciphertext encrypted with the public key $pk_1 = A_1 = (b_1, B)$, where B is a shared matrix in the CRS model. For the corresponding secret key $sk_1 = s_1$, the property $s_1C_1 \approx m_1s_1G$ holds as mentioned in Sec. 4.1.3.

To involve this ciphertext in an homomorphic evaluation with another ciphertext under a different key, we need to extend it first to the new key. Suppose $pk_2 = (b_2, B)$ is another public key which encrypts a ciphertext C_2 . Before performing evaluation between C_1 and C_2 , we need to extend these ciphertexts to the other public keys. We extend C_1 to pk_2 as $\tilde{C}_1 = \begin{pmatrix} C_1 & 0 \\ 0 & C_1 \end{pmatrix} \in \mathbb{Z}^{2n \times 2n\ell}$, which is encrypted under the concatenated key (pk_1, pk_2) . The same process applies to extending C_2 to pk_1 , but we will focus our discussion on C_1 . Note that, the property $(s_1, s_2)\tilde{C}_1 \approx m_1(s_1, s_2)\tilde{G}$, where $\tilde{G} = I_{2n} \otimes g$ and I_{2n} is the identity matrix of dimension $2n$, no longer holds.

Observe that $(s_1 \ s_2) \begin{pmatrix} C_1 & 0 \\ 0 & C_1 \end{pmatrix} = (s_1C_1 + 0 \ 0 + s_2C_1)$. Since s_1 corresponds to the public key $pk_1 = A_1$, we correctly get $C_1s_1 \approx m_1s_1G$ for the first element. Note that a correct decryption should obtain $m_1(s_1, s_1)G$. However, decrypting with s_2 does not retrieve the message m_1s_2G alone because s_2 does not cancel pk_1 as $s_2A_1 \approx 0$. Hence, it yields a “lingering” element $R_1(b_1 - b_2)$ according to the decryption of s_2C_1 shown below.

$$\begin{aligned}
s_2(A_1R_1 + m_1G) &= s_2R_1A_1 + m_1s_2G \\
&= R_1(1, -\hat{s}_2)(b_1, B) + m_1s_2G \\
&= R_1(1, -\hat{s}_2)(\hat{s}_1B + e, B) + m_1s_2G \\
&= R_1(\hat{s}_1B - \hat{s}_2B) + m_1s_2G \\
&= R_1(b_1 - b_2) + m_1s_2G
\end{aligned}$$

To solve this issue, we have to provide an auxiliary matrix X_2 such that $s_1X_2 = R_1(b_2 - b_1)$, which is needed to eliminate this lingering element. To create this matrix, we need to provide the randomness R_1 that is used to encrypt the message m_1 for semantic security but *without* disclosing it to avoid security breach as we discussed in Sec. 5.1. We encrypt each element $r_{\alpha, \beta}$ of the randomness matrix R_1 under pk_1 as $U_{\alpha, \beta} = R'_{\alpha, \beta}A_1 + r_{\alpha, \beta}G$, where α, β are in the indexes and R' is another randomness. We provide this encrypted randomness R_1 with the ciphertext C_1 as the tuple (U, C_1) .

Scheme 10: The Mukherjee-Wichs MKHE scheme [76]

- MW.Setup($1^\lambda, 1^L$): Given a security parameter λ and a circuit depth L , run the GSW.Setup to output the set of public parameters $pp = (q, n, \chi, \mathcal{B}_\chi, B)$. Let B be a common shared matrix among users.
- MW.KeyGen(pp): Given the public parameters pp , run the key generation algorithm GSW.Setup for each user and output the individual key pair $(pk, sk) = (A, s)$ where $A = (b = \hat{s}B + e, B)$ and $s = (1, -\hat{s})$.
- MW.Enc(pk, m): To encrypt a message bit m with the public key $pk = A$ and a randomness R , run GSW.Enc to obtain the GSW ciphertext $C = AR + mG$. Additionally, encrypt each entry of the randomness R such that $U = \{U_{\alpha, \beta}\}_{\alpha, \beta=1, \dots, n\ell}$ where $U_{\alpha, \beta} = \text{Enc}(pk, r_{\alpha, \beta})$. Output the ciphertext as Output the ciphertext as the pair $\hat{C} = (U, C)$.
- MW.Extend($(pk_1, \dots, pk_K), \hat{C}$): Given a list of K public keys (pk_1, \dots, pk_K) and a ciphertext $\hat{C} = (U, C)$ encrypting a message m under pk_i , compute the auxiliary matrix $X_j = b_j - b_i$ for each $pk_{j \neq i}$. Construct a matrix $\bar{C} \in \mathbb{Z}_q^{nK \times n\ell K}$ such that each sub-matrix $\bar{C}_{a,b} \in \mathbb{Z}^{n \times n\ell}$ for $a, b \in 1, \dots, K$ is defined as follows.

$$\bar{C}_{a,b} = \begin{cases} C, & \text{if } a = b \\ X_j, & \text{if } a = i \neq j \text{ and } b = j \\ \mathbf{0}_{n \times n\ell}, & \text{otherwise} \end{cases}$$

Output \bar{C} as the extended ciphertext under the concatenated key $\bar{pk} = (pk_1, \dots, pk_K)$. For the corresponding secret key $\bar{sk} = (sk_1, \dots, sk_K)$, the property $(s_1, \dots, s_K)C \approx m(s_1, \dots, s_K)\bar{G}$ holds, where $\bar{G} = I_{nK} \otimes g$.

- MW.Eval($\bar{pk}, f, \bar{C}_1, \dots, \bar{C}_u$): Given a set of u extended ciphertexts $(\bar{C}_1, \dots, \bar{C}_u)$, directly perform the original GSW evaluation algorithms GSW.Add and GSW.Mult but in the extended dimensions $nK \times n\ell K$.
- MW.Dec(\bar{sk}, \bar{C}): Let \bar{C} be an extended ciphertext \bar{C} encrypted under K keys and consisting of K sub-matrices, such that V_i includes the ciphertext C and the auxiliary matrix X_i . Each party i performs locally the partial decryption using their secret sk_i and V_i and output $\rho_i = s_i V_i G^{-1}(w^T) + e_i$, where $w^T = (0, \dots, 0, \lceil q/2 \rceil)$ and e_i is a smudging noise. The message is retrieved as $m = \left\lfloor \left\lfloor \frac{\sum_{i=1}^K (\rho_i)}{q/2} \right\rfloor \right\rfloor$.

At the ciphertext extension step, we define a matrix $Z_{\alpha, \beta} = (\mathbf{0}_{n \times n\ell-1}, \mathbf{b}_2 - \mathbf{b}_1) \in \mathbb{Z}_q^{n \times n\ell}$, which means all entries are 0 except the last column is the vector $\mathbf{b}_2 - \mathbf{b}_1$. Note that $\mathbf{b}_1, \mathbf{b}_2$ are components of the public keys pk_1, pk_2 , respectively. Then we compute $X_2 = \sum_{\alpha, \beta} U_{\alpha, \beta} G^{-1}(Z_{\alpha, \beta})$, such that the property $s_1 X_{2, \alpha, \beta} \approx r_{\alpha, \beta} (\mathbf{b}_2 - \mathbf{b}_1)$ holds. Finally, output the extended ciphertext as $\bar{C}_1 = \begin{pmatrix} C_1 & X_2 \\ 0 & C_1 \end{pmatrix}$.

This process can be generalized to extending C_1 to K different keys. For each additional key pk_j , we compute the matrix X_j such that $s_1 X_j \approx R_1 (\mathbf{b}_j - \mathbf{b}_1)$. The new structure of the extended ciphertext is illustrated in Fig. 9a as follows.

$$\bar{C}_1 = \begin{pmatrix} C_1 & X_2 & \cdots & X_K \\ 0 & C_1 & \cdots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & \cdots & \cdots & C_1 \end{pmatrix} \in \mathbb{Z}^{nK \times n\ell K}$$

As observed, the dimension of the extended ciphertext matrix can increase quadratically with the number of involved keys because we have to add the auxiliary matrix and an additional copy of the ciphertext. With each extension, the dimension of the gadget matrix G is also changed in correspondence with the size of the extended key. Namely, we create a $(nK \times n\ell K)$ matrix where the diagonal is the vector $\mathbf{g} = (2^0, \dots, 2^\ell)$.

Homomorphic evaluations on extended ciphertexts performs the original GSW addition and multiplication. The only difference is performing on ciphertexts with higher dimensions based on the K involved keys.

Let \tilde{C}_{eval} be the result of some evaluations on \tilde{C}_1 with respect to the concatenated key (pk_1, pk_2) . Suppose that we want to extend the result to an additional key pk_3 . In this case, we must provide an encryption of the randomness R_{eval} , associated with the ciphertext \tilde{C}_{eval} , under the corresponding set of keys so it can be used to generate X_3 . However, it is not clear how to obtain such an encryption for R_{eval} . Note the randomness is affected by the homomorphic evaluation. For example, the product of \tilde{C}_1 with another ciphertext \tilde{C}_2 under the same keys (pk_1, pk_2) may contain randomness $R_{\text{eval}} \approx R_1 \cdot R_2$, where R_2 is a randomness associated with \tilde{C}_2 . Constructing such term from the encryptions is not trivial. The scheme does not describe either a method to dynamically extend the matrix X_1 to other keys. Hence, this renders a single-hop scheme.

Given a ciphertext \tilde{C} under (pk_1, \dots, pk_K) , previous MKHE schemes [37, 70] assumed the presence of a *trusted* party who holds all the corresponding secret keys (sk_1, \dots, sk_K) and can directly decrypt. To achieve a stronger security and avoid disclosing individual secret keys to anyone, this scheme performs a one-round distributed decryption protocol (similar to the ThHE technique). Let \tilde{C} be a ciphertext extended under K keys and composed of K sub-matrices such that $\tilde{C} = (\tilde{C}_1, \dots, \tilde{C}_K); \tilde{C}_i \in \mathbb{Z}_q^{n \times n\ell}$. All K participants will agree on a vector $\tilde{w} = (0, \dots, 0, \lceil q/2 \rceil) \in \mathbb{Z}^{nK}$. In the one round of the protocol, each participant i partially decrypts \tilde{C} by locally computing a masked decryption component $\rho_i = s_i \tilde{C}_i \tilde{G}^{-1}(\tilde{w}^T) + e_i$, where e_i is a smudging noise used to protect s_i under LWE according to the lemma in Sec. 3.3.3. After receiving the broadcast components (ρ_1, \dots, ρ_K) , the intended participant performs the final decryption by aggregating components as $\rho = \sum_{i=1}^K (\rho_i)$ and computing $m = \left\lfloor \left\lceil \frac{\rho}{q/2} \right\rceil \right\rfloor$ to retrieve the message.

Based on this scheme, Mukherjee and Wichs [76] constructed an MPC protocol with two rounds to enable secure computation with multiple keys. In the first round, users encrypt their inputs with their individual keys and output the ciphertexts and encryption of the randomness. Then, the evaluator can dynamically extend a ciphertext to K different keys without knowing any secret components, such as the encrypted randomness, and perform the homomorphic evaluations. In the second round, each user helps in a distributed decryption protocol by partially decrypting a part of the extended ciphertext such that the message is retrieved by combining these partial decryptions.

As mentioned, there is no limit on the number of involved keys K . Yet, the scheme is still not robust because no additional keys, or even further homomorphic evaluations, can be supported on evaluated ciphertexts. Moreover, the ciphertext size expansion is significant, which affects the overall practicality of the scheme. We discuss techniques for realizing multi-hop MKHE schemes in the following section.

5.2.3 Multi-hop MKHE. Recent MKHE schemes adopt new design that overcomes the limitations of earlier MKHE schemes and allows ciphertext extension after homomorphic evaluation. We review the state-of-the-art MKHE schemes and show how these schemes adopt the new design to support multi-hop.

Multi-hop MK-variant of GSW. Building on the single-hop Mukherjee-Wichs scheme [76], two concurrent works [23, 81] proposed techniques to extend it to the multi-hop setting.

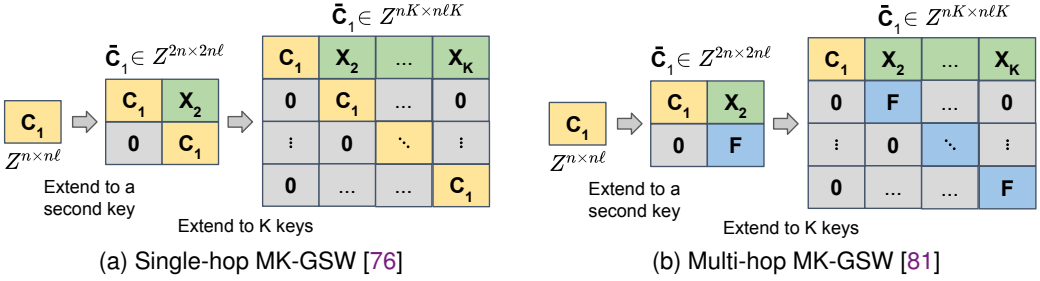


Fig. 9. A comparison between single-hop and multi-hop MK-variants of the GSW scheme

Peikert and Shiehian [81] proposed a *leveled* scheme that is based on the Mukhenrjee-Wichs scheme but has a new ciphertext structure and extension function. In the setup step, a public parameter in the form of uniformly LWE matrix $B \leftarrow \mathbb{Z}_q^{n \times 2n\ell}$ is chosen. The ciphertext in the new scheme consists of three components (C, F, D) compared to the former ciphertext tuple (C, U) . The component $C \in \mathbb{Z}_q^{n \times n\ell}$ remains the same as an original GSW ciphertext of the message m . The new component $F = BR + mG \in \mathbb{Z}_q^{n \times n\ell}$ is the commitment of the message m under the randomized public parameter B . Note the message is protected since the randomness R is not disclosed. The randomness is encrypted and provided as a ciphertext $D \in \mathbb{Z}_q^{2n^2\ell^2 \times n\ell}$ such that $sD \approx R$ holds.

Given a ciphertext (C, F, D) under s_1 , we can extend it to a second key s_2 as $(\bar{C}, \bar{F}, \bar{D})$, where $\bar{F} = F$ and \bar{D} is extended by padding it with rows and columns of zeros such that the new padded component is $\bar{D} = I_{2n\ell} \otimes \begin{pmatrix} I_n \\ 0_{n \times n} \end{pmatrix} \cdot D \in \mathbb{Z}_q^{4n^2\ell^2 \times n\ell}$. Note that we denote all extended elements with a

bar on the top, such as \bar{D} . The new structure of the extended ciphertext is $\bar{C} = \begin{pmatrix} C & X \\ 0 & F \end{pmatrix}$. Note that decrypting $s_2\bar{C} = s_2BR + mG = b_2R + mG$ where b_2R is a lingering term. The component X encrypts information of the inverse of this lingering term such that $s_1X \approx -b_2R$ which cancels the resulted lingering term.

With this new design, as illustrated in Fig. 9b, it is possible to further extend the ciphertext \bar{C} to K additional keys. Note in the Fig. 9, some elements in the extended ciphertexts have been substituted from C_1 to F . The new scheme is multi-hop because the randomness used in F, D is tied to the same public parameter rather than a new key at every key extension process. Hence, there is no need to provide an encryption of some randomness under a specific key to construct the cancelling term, as discussed in Sec. 5.2.2.

The size of the ciphertext grows *quadratically* with the number of involved keys. Hence, the scheme is limited by a bounded number of keys. An alternative scheme is introduced by authors to obtain smaller ciphertexts. Specifically, the extended ciphertext remains as a GSW encryption but the additional extension information is embedded in its corresponding public key instead. This scheme yields smaller ciphertexts but larger public keys, which reduces the space overhead since the number of evaluated ciphertexts is likely to be more than the number of extended public keys. Both schemes are leveled, which support homomorphic evaluations up to a predefined level, but they can be made fully through the bootstrapping technique.

Brakerski and Perlman [23] proposed a *fully* MK-variant of GSW scheme to overcome the single-hop limitation in [76]. Ciphertexts in the single-hop scheme cannot be further extended without being decrypted and encrypted again. This process can be done homomorphically via bootstrapping, as

discussed in Sec. 3.3.2. Hence, the Brakerski-Perlman method depends on the use of bootstrappable NAND gates for evaluation. Note that a NAND gate has a function completeness property, which means we can support any other operation by combining a set of NAND gates. Let C_1 and C_2 be two ciphertexts encrypted under two different keys pk_1 and pk_2 . Evaluating a bootstrappable NAND on the two ciphertexts homomorphically performs $\text{NAND}(\text{Dec}(sk_1, C_1), \text{Dec}(sk_2, C_2))$ and yields the extended ciphertext \tilde{C} that is encrypted under the concatenated keys (pk_1, pk_2) . As observed, we must provide as inputs the secret keys sk_1, sk_2 encrypted under their corresponding public keys to enable homomorphic decryption. Therefore, a circular-security (see Sec. 2.3) assumption is required. Moreover, the size of the ciphertexts can be significantly reduced in this scheme. As discussed in Sec. 4.1.3, the decryption process can be sufficiently done with one single column vector of the GSW ciphertext. Since we need to perform decryption first in the evaluation, the scheme proposes to keep the last column of the ciphertext and discard the rest. Hence, the size of the ciphertexts grows linearly, instead of quadratically, with the number of involved keys. There is no bound on the number of because ciphertexts are refreshed after each evaluation which keeps the noise level reduced. However, the bootstrapping can be slow and the efficiency of the scheme may degrade as the number of bootstrappable gates increases.

MK-variant of BGV. The BGV scheme [22] is one among the first RLWE-based schemes to enable ciphertext packing, where multiple messages can be packed in one ciphertext to support efficient SIMD computation. The first attempt of extending the BGV scheme [22] to multi-key setting was by López-Alt et al. [70]. Unfortunately, this initial MKHE scheme did not support key switching technique (see Sec. 3.3.1) or bootstrapping which prevented extending it to a leveled or FHE scheme.

Later, Chen et al. [32] proposed a multi-key variant of the BGV scheme that is multi-hop and supports extending a ciphertext to a bounded number of keys. We describe this new multi-key BGV (MKBGV) scheme in Scheme 11. Similar to other LWE-based MKHE schemes, the MKBGV scheme is designed in the CRS model. Participants use a pre-shared LWE element $a \leftarrow R_q$ to generate their key pairs. Messages are initially encrypted under a single key following the base BGV encryption function discussed in Sec. 4.1.1.

The scheme supports extending the ciphertexts to at most \mathcal{K} keys. Let $K \leq \mathcal{K}$ be the number of participants who wish to compute with their keys. Each participant is assigned a unique index $i \in \{1, \dots, K\}$ such their index is recorded in a special ordered set \mathcal{S} for each ciphertext under the corresponding key. Let $c_1 \in R_q^2$ be a BGV ciphertext encrypting m_1 under the public key pk_1 . The set for the ciphertext c_1 is populated with the index 1 as $\mathcal{S} = (1)$ since the encryption is under the first participant's key pk_1 . Extending the ciphertext to a second key pk_2 is done by constructing a new ciphertext with two slots as $\tilde{c}_1 = (c'_1, c'_2) \in R_q^4$. Each slot holds a sub-ciphertext corresponding to the index set. In particular, we define $c'_1 = c_1 \in R_q^2$ as the original ciphertext since the index $1 \in \mathcal{S}$. The second sub-ciphertext is initiated as zero $c'_2 = (0, 0) \in R_q^2$ because the index $2 \notin \mathcal{S}$. The index set is updated after the extension to include the new index $\mathcal{S} = (1, 2)$. In a similar manner, we can further extend this ciphertext to additional K keys as illustrated in Fig. 10a. The size of the ciphertext increases *linearly* with the number of keys such that for a ciphertext encrypted under K keys, we have $\tilde{c} \in R_q^{2K}$.

Homomorphic evaluations can be performed on the extended ciphertexts. Given two ciphertexts $\tilde{c}_1, \tilde{c}_2 \in R_q^4$ encrypted under the same set of keys, their sum is obtained through slot-wise homomorphic addition $\tilde{c}_{\text{add}} = \tilde{c}_1 + \tilde{c}_2 \in R_q^4$. Note that sub-ciphertexts in different slots do not interact directly with each other. Rather, we perform slot-wise operations and the final result can be aggregated at the decryption step as shown in the example in Fig. 10b. On the other hand, homomorphic multiplication is done by performing tensor-product of the two ciphertexts. The initial product is a long ciphertext $\tilde{c}_{\text{mult}} = \tilde{c}_1 \otimes \tilde{c}_2 \in R_q^{16}$ that is encrypted under $\tilde{s} \otimes \tilde{s}$ instead of \tilde{s} . We need to apply key switching to

Scheme 11: An MK-variant of BGV scheme [32]

- MKBGV.Setup($1^\lambda, 1^L, 1^K$): Given the security parameter λ , a multiplicative depth L , and a bound K on the number of keys, run the BGV.Setup function. Output the public parameters $pp = (R, \chi, \mathcal{B}, q, a, t)$ where $a \leftarrow R_q$ is the CRS vector.
- MKBGV.KeyGen(pp): Given the public parameters pp , run BGV.KeyGen to generate a key pair (sk, pk) as $sk = s \leftarrow \psi$ and $pk = (b = -as + te, a)$.
- MKBGV.Enc(pk, m): Given a message $m \leftarrow R_p$ and a public key pk , perform the BGV.Enc function to obtain the ciphertext as $c = (c_0, c_1) \in R_q^2$ where $c_0 = rb + m + te_0$ and $c_1 = ra + te_1$. Assume each participant has a unique index i and initiate an ordered set $\mathcal{S} = (i)$ for the ciphertext. Output the fresh BGV ciphertext as the tuple $\hat{c} = (c, \mathcal{S}, L)$, where L is the initial level.
- MKBGV.Extend($((pk_1, \dots, pk_K), \hat{c})$): To extend a BGV ciphertext to one encrypted under a set of K users' keys, simply set the ciphertext c as a concatenated K sub-vectors $\bar{c} = (c'_1, \dots, c'_K) \in R_{q_l}^{2N}$, such that $c'_i = c_i$ if the index $i \in \mathcal{S}$, and $c'_i = 0$ otherwise. Update the index set \mathcal{S} and the level correspondingly.
- MKBGV.Eval($(\bar{pk}, f, \bar{c}, \bar{c}')$): Given two extended ciphertexts $\bar{c}, \bar{c}' \in R_{q_l}^{2K}$ encrypted under the same concatenated key \bar{pk} , perform homomorphic addition as $\bar{c}_{add} = \bar{c} + \bar{c}' \pmod{q}$ and homomorphic multiplication as $\bar{c}_{mult} = \bar{c} \otimes \bar{c}' \pmod{q}$. Then perform the KeySwitch technique using the generated evaluation key \bar{ek} to reduce the ciphertext dimension and the ModulusSwitch technique to decrease the resultant noise by switching to a smaller ciphertext modulus.
- MKBGV.Dec($(sk_1, \dots, sk_K), \bar{c}$): Given an extended ciphertext \bar{c} , and the secret key \bar{s} concatenating all the secret keys of users in the set \mathcal{S} . Decrypt as $\langle \langle \bar{c}, \bar{s} \rangle \pmod{q} \rangle \pmod{t} = \langle \sum_{i=1}^K \langle c'_i, s_i \rangle \pmod{q} \rangle \pmod{t}$.

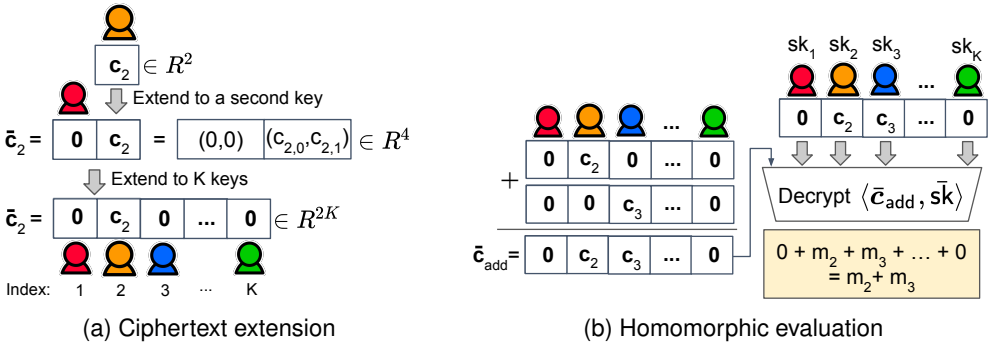


Fig. 10. MK-variant of the BGV scheme

convert \bar{c}_{mult} to an encryption under \bar{s} and reduce its dimension to R_q^4 . As mentioned in Sec. 3.3.1, this process requires providing an evaluation key \bar{ek} , which encrypts the long secret key $\bar{s} \otimes \bar{s}$. To facilitate the generation of the evaluation key corresponding to the extended secret key, the scheme proposes a ring-GSW scheme, which allows encrypting ring elements. Specifically, each participant

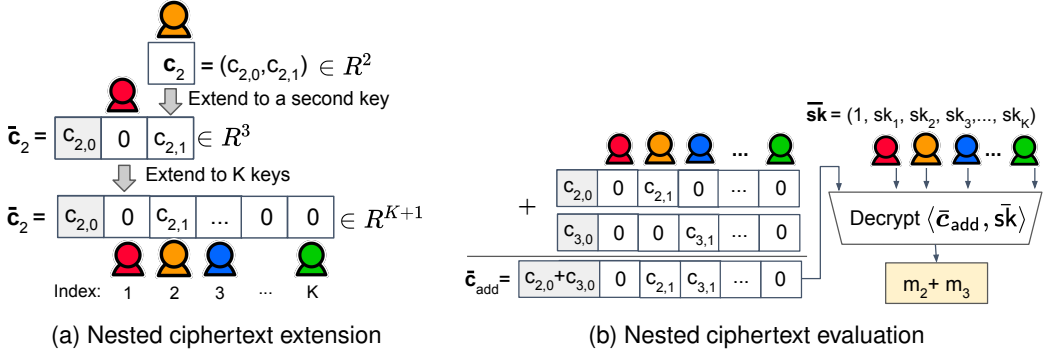


Fig. 11. Improved MK-variant of the BGV scheme with reduced ciphertexts

provides a GSW encryption of their secret key s_i such that each row can be viewed as a BGV ciphertext. Then, these ciphertexts can be extended to their concatenated keys. Lastly, one homomorphic multiplication is performed to obtain the encryption of $\bar{s} \otimes \bar{s}$ under the extended key that is used in the key switching procedure.

The decryption process can be done with a two-round distributed decryption protocol similar to previous schemes [23, 76, 81]. It requires all participants, whose indexes are in \mathcal{S} , to collaborate in a distributed decryption protocol. Each participant uses their secret key sk_i to decrypt the corresponding sub-ciphertext c'_i . Observe that decryptions of zero sub-ciphertexts are cancelled and the final result can be obtained through the aggregation of the decrypted BGV sub-ciphertexts.

Li et al. [67] proposed an alternative design for MKBGV scheme that reduced the size of extended ciphertexts roughly by half. In particular, the dimension (number of ring elements) of ciphertext extended under K keys will be $K + 1$ instead of $2K$. Recall that a BGV ciphertext has two components $c = (c_0, c_1) \in R_q^2$, where c_0 includes the message and c_1 encodes the randomness used in encryption. In the initial MKBGV scheme, the extended ciphertext under K keys has each slot holding two ring elements as a full BGV ciphertext or padded with zeros $(0, 0)$. Sub-ciphertexts still need to be combined at decryption step. However, the new scheme follows a nested ciphertext design which allocates the first slot of the extended ciphertext to the first ciphertext component c_0 which holds the message. Subsequent K slots are allocated to the second ciphertext component which holds the randomness or padded with zero 0 as illustrated in Fig. 11a. For example, we can extend a ciphertext c_1 , with a corresponding index set $\mathcal{S} = (1)$, to a second key pk_2 as $\bar{c}_1 = (c'_0, c'_1, c'_2) \in R_q^3$ such that $c'_0 = c_0$ and $c'_i = c_i$ if $i \in \mathcal{S}$ or $c'_i = 0$ otherwise.

Homomorphic operations in this scheme are performed similar to the initial MKBGV scheme. An example of homomorphic addition is shown in Fig. 11b. Homomorphic multiplication is performed via tensor product. The generation of evaluation key is modified by combining operations between GSW and BGV ciphertexts instead of between pair of GSW ciphertexts to reduce the space overhead. The scheme also suggested a *directed* distributed decryption protocol to prevent all participants from learning the decrypted result. In a nutshell, each participant sends their partial decryption to the designated participant, who is intended to receive the result. This participant does not share their partial decryption but collects all partial decryptions and combine them locally to retrieve the message. This design enforces the dishonest-majority assumption since none of the participants is able retrieve the message without having the partial decryption of the designated participant.

MK-variant of TFHE. The MKHE primitive remained abstract constructions that are not practical for years. Chen, Chillotti, and Song [29] designed an MK variant of the TFHE scheme [36] and provided the first proof-of-concept implementation of MKHE.

TFHE is an FHE over the torus $\mathbb{T} = \mathbb{R} \pmod{1}$ which supports the evaluation of a binary gate followed by a fast bootstrapping procedure, which takes about 13ms. Its multi-key variant is multi-hop and has the same functionality as the single-key TFHE. We provide a description in Scheme 12.

In the original TFHE, the bootstrapping procedure relied on the *external product* which takes a pair of ring-GSW and RLWE ciphertexts as input and returns an RLWE ciphertext. In [29], the authors proposed two MK variants of the external product: the first approach is an improvement of the previous ciphertext extension technique of [76] and the other one is a new solution which does not require the generation of an extended evaluation key. Their implementation is based on the second method which has advantages in terms of noise growth and complexity. A single party's evaluation key is directly used on the corresponding element from extended ciphertext.

- UniEnc(\mathbf{a}, s_i): For the CRS $\mathbf{a} \in R_q^\ell$ and the secret s_i , sample $r_i \leftarrow \psi$. Sample $\mathbf{d}_{i,1}$ uniformly at random from R_q^ℓ and errors $\mathbf{e}_{i,1}, \mathbf{e}_{i,2}$ from χ^ℓ . Return the relinearization key $\text{rlk}_i = (\mathbf{d}_{i,0}, \mathbf{d}_{i,1}, \mathbf{d}_{i,2})$ where $\mathbf{d}_{i,0} = -s_i \cdot \mathbf{d}_{i,1} + \mathbf{e}_{i,1} + r_i \cdot \mathbf{g} \pmod{q}$ and $\mathbf{d}_{i,2} = r_i \cdot \mathbf{a} + \mathbf{e}_{i,2} + s_i \cdot \mathbf{g} \pmod{q}$.
- ExtProd($\{\text{rlk}_i\}_{1 \leq i \leq k}, \hat{\mathbf{c}}$): Given a ciphertext $\hat{\mathbf{c}} = (c_{i,j})_{0 \leq i, j \leq K}$ and the set of relinearization keys $\text{rlk}_i = (\mathbf{d}_{i,0}, \mathbf{d}_{i,1}, \mathbf{d}_{i,2})$, compute and return the ciphertext $\mathbf{c}' = (c'_0, \dots, c'_K) \in R_q^{K+1}$ as follows:
 - (1) Initialize the ciphertext \mathbf{c}' as $c'_0 \leftarrow c_{0,0}$ and $c_i \leftarrow c_{i,0} + c_{0,i} \pmod{q}$ for $1 \leq i \leq K$.
 - (2) For $1 \leq i, j \leq K$, do $c'_{i,j} \leftarrow \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{b}_j \rangle \pmod{q}$, $(c'_0, c'_i) \leftarrow (c'_0, c'_i) + \mathbf{g}^{-1}(c'_{i,j}) \cdot [\mathbf{d}_{i,0} | \mathbf{d}_{i,1}] \pmod{q}$, and $c'_j \leftarrow c'_j + \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{d}_{i,2} \rangle \pmod{q}$.

Scheme 12: An MK-variant of TFHE scheme [29]

- MKTFHE.Setup(1^λ): Given a security parameter λ , choose the RLWE dimension n , the ciphertext module q , and the plaintext module t . Set the error distribution χ_e and the key distribution χ_k over R . Uniformly choose a random element $a \leftarrow R_q$. Output the public parameters as $\text{pp} = (n, q, t, \chi_e, \chi_k, a)$.
- MKTFHE.KeyGen(pp):
- MKTFHE.Enc(pk, m): Given a public key $\text{pk} = -sa + e \in R_q$ and a plaintext message $m \in R_t$. Sample $r \leftarrow \chi_k$ and $e_0, e_1 \leftarrow \chi_e$. Compute the ciphertext as $\mathbf{c} = (c_0, c_1) = (-ras + \Delta m + e_0, ra + e_1) \in R_q$, where $\Delta = \lfloor q/t \rfloor$.
- MKTFHE.Eval($\bar{\text{pk}}, f, \bar{\mathbf{c}}_1, \bar{\mathbf{c}}_2$): For given two extended ciphertexts under the same extended secret key $\bar{\text{sk}} = (s_1, \dots, s_K)$, perform homomorphic evaluation as follows. For addition, $\bar{\mathbf{c}}_{\text{add}} = \bar{\mathbf{c}}_1 + \bar{\mathbf{c}}_2 \in R_q^{K+1}$. For multiplication, compute $\hat{\mathbf{c}}_{\text{mult}} = \lfloor t/q \rfloor (\bar{\mathbf{c}}_1 \otimes \bar{\mathbf{c}}_2) \in R_q^{(K+1)^2}$ and return the ciphertext $\bar{\mathbf{c}}_{\text{mult}} = \text{Relinearize}(\hat{\mathbf{c}}_{\text{mult}}) \in R_q^{K+1}$.
- MKTFHE.Dec($(\text{sk}_1, \dots, \text{sk}_K), \bar{\mathbf{c}}$): Given a set of secret keys and an extended ciphertext $\bar{\mathbf{c}} = (c_0, c_1, \dots, c_K)$, set the extended secret key as $\bar{\text{sk}} = (1, \text{sk}_1, \dots, \text{sk}_K)$ and compute $\lfloor \lfloor t/q \rfloor \langle \bar{\mathbf{c}}, \bar{\text{sk}} \rangle \rfloor \pmod{t}$ to retrieve the underlying message.

MK-variant of BFV/CKKS. Chen et al. [30] proposed MK variants of the BFV and CKKS schemes. Different from the MKBGV scheme [32], the authors proposed a new linearization algorithm which does not require any precomputation on the evaluation keys. We first provide a description of the relinearization procedure which will be commonly used in two MKHE schemes. This method is also

based on the CRS model and each party independently generates a relinearization key using a CRS $\mathbf{a} \in R_q^\ell$. It allows us to directly relinearize a product of MK ciphertexts using individual keys from the involved parties. As a result, both the space and time complexity grow linearly with K . Then, we present MKBFV and MKCKKS in Schemes 13 and 14, respectively.

- $\text{RelinKeyGen}(\mathbf{a}, s_i)$: For the CRS $\mathbf{a} \in R_q^\ell$ and the secret s_i , sample $r_i \leftarrow \psi$. Sample $\mathbf{d}_{i,1}$ uniformly at random from R_q^ℓ and errors $\mathbf{e}_{i,1}, \mathbf{e}_{i,2}$ from χ^ℓ . Return the relinearization key $\text{rlk}_i = (\mathbf{d}_{i,0}, \mathbf{d}_{i,1}, \mathbf{d}_{i,2})$ where $\mathbf{d}_{i,0} = -s_i \cdot \mathbf{d}_{i,1} + \mathbf{e}_{i,1} + r_i \cdot \mathbf{g} \pmod{q}$ and $\mathbf{d}_{i,2} = r_i \cdot \mathbf{a} + \mathbf{e}_{i,2} + s_i \cdot \mathbf{g} \pmod{q}$.
- $\text{Relinearize}(\{\text{rlk}_i\}_{1 \leq i \leq k}, \hat{\mathbf{c}})$: Given a ciphertext $\hat{\mathbf{c}} = (c_{i,j})_{0 \leq i,j \leq K}$ and the set of relinearization keys $\text{rlk}_i = (\mathbf{d}_{i,0}, \mathbf{d}_{i,1}, \mathbf{d}_{i,2})$, compute and return the ciphertext $\mathbf{c}' = (c'_0, \dots, c'_K) \in R_q^{K+1}$ as follows:
 - (1) Initialize the ciphertext \mathbf{c}' as $c'_0 \leftarrow c_{0,0}$ and $c_i \leftarrow c_{i,0} + c_{0,i} \pmod{q}$ for $1 \leq i \leq K$.
 - (2) For $1 \leq i, j \leq K$, do $c'_{i,j} \leftarrow \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{b}_j \rangle \pmod{q}$, $(c'_0, c'_i) \leftarrow (c'_0, c'_i) + \mathbf{g}^{-1}(c'_{i,j}) \cdot [\mathbf{d}_{i,0} | \mathbf{d}_{i,1}] \pmod{q}$, and $c'_j \leftarrow c'_j + \langle \mathbf{g}^{-1}(c_{i,j}), \mathbf{d}_{i,2} \rangle \pmod{q}$.

Scheme 13: An MK-variant of BFV scheme [30]

- $\text{MKBFV.Setup}(1^\lambda)$: Given a security parameter λ , choose the RLWE dimension n , the ciphertext module q , and the plaintext module t . Set the error distribution χ_e and the key distribution χ_k over R . Uniformly choose a random element $a \leftarrow R_q$. Output the public parameters as $\text{pp} = (n, q, t, \chi_e, \chi_k, a)$.
- $\text{MKBFV.KeyGen}(\text{pp})$: Given the public parameters, sample a secret element $s \leftarrow \chi_k$ and an error $e \leftarrow \chi_e$. Output the key pair as (pk, sk) , where $\text{pk} = -as + e \in R_q$ and $\text{sk} = s$.
- $\text{MKBFV.Enc}(\text{pk}, m)$: Given a public key $\text{pk} = -sa + e \in R_q$ and a plaintext message $m \in R_t$. Sample $r \leftarrow \chi_k$ and $e_0, e_1 \leftarrow \chi_e$. Compute the ciphertext as $\mathbf{c} = (c_0, c_1) = (-ras + \Delta m + e_0, ra + e_1) \in R_q$, where $\Delta = \lfloor q/t \rfloor$.
- $\text{MKBFV.Eval}(\overline{\text{pk}}, f, \bar{\mathbf{c}}_1, \bar{\mathbf{c}}_2)$: For given two extended ciphertexts under the same extended secret key $\bar{\text{sk}} = (s_1, \dots, s_K)$, perform homomorphic evaluation as follows. For addition, $\bar{\mathbf{c}}_{\text{add}} = \bar{\mathbf{c}}_1 + \bar{\mathbf{c}}_2 \in R_q^{K+1}$. For multiplication, compute $\hat{\mathbf{c}}_{\text{mult}} = \lfloor t/q \rfloor (\bar{\mathbf{c}}_1 \otimes \bar{\mathbf{c}}_2) \in R_q^{(K+1)^2}$ and return the ciphertext $\bar{\mathbf{c}}_{\text{mult}} = \text{Relinearize}(\hat{\mathbf{c}}_{\text{mult}}) \in R_q^{K+1}$.
- $\text{MKBFV.Dec}(\{\text{sk}_1, \dots, \text{sk}_K\}, \bar{\mathbf{c}})$: Given a set of secret keys and an extended ciphertext $\bar{\mathbf{c}} = (c_0, c_1, \dots, c_K)$, set the extended secret key as $\bar{\text{sk}} = (1, \text{sk}_1, \dots, \text{sk}_K)$ and compute $\lfloor \lfloor t/q \rfloor \langle \bar{\mathbf{c}}, \bar{\text{sk}} \rangle \rfloor \pmod{t}$ to retrieve the underlying message.

5.3 Hybrid Multi-key Homomorphic Encryption

The proportional increase in ciphertext size remains a bottleneck for efficient MKHE schemes. The size of a ciphertext grows correspondingly to the number of involved participants' keys. This may pose an efficiency limitation in many outsourced computation scenarios with a large number of participants. Suppose we have a system with N model owners and P clients. The model owners want collaborate and jointly compute on their trained models or functions and evaluate their client's requests. The model owners want to keep their sensitive models private, so they encrypt them under their individual keys and delegate them to a Cloud. The Cloud evaluates these encrypted models but should not learn anything from them. In this system, both model owners and clients want to protect their inputs under their keys.

Scheme 14: An MK-variant of CKKS scheme [30]

- MKCKKS.Setup(1^λ): Given a security parameter λ , choose the RLWE dimension n and a chain of ciphertext moduli $q_0 < q_1 < \dots < q_L$. Assume $q = \prod_{i=0}^L p_i$ for some integers p_i , such that $q_\ell = \prod_{i=0}^\ell p_i$. Set the error distribution χ_e and the key distribution χ_k over R . Uniformly choose a random element $a \leftarrow R_q$. Output the public parameters as $\text{pp} = (n, q, \chi_e, \chi_k, a)$.
- MKCKKS.KeyGen(pp): Given the public parameters, sample a secret element $s \leftarrow \chi_k$ and an error $e \leftarrow \chi_e$. Output the key pair as (pk, sk) , where $\text{pk} = -sa + e \in R_q$ and $\text{sk} = s$.
- MKCKKS.Enc(pk, m): Given a public key $\text{pk} = -sa + e \in R_q$ and a plaintext message $m \in R$. Sample $r \leftarrow \chi_k$ and $e_0, e_1 \leftarrow \chi_e$. Compute the ciphertext as $c = (c_0, c_1) = r \cdot \text{pk} + (m + e_0, e_1) \in R_q^2$.
- MKCKKS.Eval($\bar{\text{pk}}, f, \bar{c}_1, \bar{c}_2$): Given two extended ciphertexts under the same extended secret key $\bar{\text{sk}} = (s_1, \dots, s_K)$ at level ℓ for K the number of involved parties, perform homomorphic evaluation as follows. For addition, $\bar{c}_{add} = \bar{c}_1 + \bar{c}_2 \in R_{q_\ell}^{K+1}$. For multiplication, compute $\hat{c}_{\text{mult}} = \bar{c}_1 \otimes \bar{c}_2 \in R_{q_\ell}^{(K+1)^2}$. Perform the relinearization and return the ciphertext $\bar{c}_{\text{mult}} = \text{Relinearize}(\hat{c}_{\text{mult}}) \in R_{q_\ell}^{K+1}$.
- MKCKKS.Rescale(\bar{c}): Given a ciphertext $\bar{c} = (c_0, \dots, c_K) \in R_{q_\ell}^{K+1}$ at level ℓ , compute $c'_i = \lfloor p_\ell^{-1} c_i \rfloor$ for $0 \leq i \leq K$ and return the re-scaled ciphertext $\bar{c}' = (c'_0, \dots, c'_K) \in R_{q_{\ell-1}}^{K+1}$.
- MKCKKS.Dec($(\text{sk}_1, \dots, \text{sk}_K), \bar{c}$): Given a set of secret keys and an extended ciphertext $\bar{c} = (c_0, c_1, \dots, c_K) \in R_{q_\ell}$, set the extended secret key as $\bar{\text{sk}} = (1, \text{sk}_1, \dots, \text{sk}_K)$ and compute $\langle \bar{c}, \bar{\text{sk}} \rangle \pmod{q_\ell}$ to retrieve the underlying message.

Leveraging MKHE schemes for this scenario will require each ciphertext to be extended to $N + 1$ different keys (i.e., the model owners keys plus the key of the requesting client) before any computation. The efficiency of the system is affected, especially if the number of model owners is large because it proportionally increases the ciphertext size. On the other hand, one may leverage ThHE schemes which compute under a joint key generated from participants keys. However, this approach is also not practical because we need to generate a joint key for every client and the group of N model owners; that is, we will need to produce and maintain P joint keys. This requirement also means that each model owner has to provide P copies of the model, each encrypted under one of the P joint keys, to the Cloud. Moreover, if there is any change (adding or removing) in the participants requires generating a new joint key and re-encrypting all ciphertexts with this new joint key, which is inefficient.

To address this issue, Aloufi and Hu [4] proposed a hybrid approach (MKHE+) based on the BGV scheme [22]. The scheme supports homomorphic computation over ciphertexts encrypted under multi-key and produces small ciphertexts. The approach combines the advantages of both ThHE and MKHE techniques to reduce computation complexity and ciphertext size. We present the MKHE+ scheme in Scheme 5.3.

The fundamental observation in MKHE+ is that the group of model owners do not often change and is likely to remain the same during the evaluation for each client. Hence, it is better to generate one joint key pk_M , using the ThHE key setup, for the model owners group to encrypt their models under this joint key. On the other hand, client's request can be dynamic as clients can come and go; therefore, a client can encrypt their data under their own key pk_C . This way the number of keys we are computing with becomes two keys – the model owner's joint key pk_M and the client's key pk_C .

Scheme 15: A Threshold MKHE scheme [4]

- MKHE + .KeyGen(1^λ): Given a security parameter λ , generate a key pair (pk_{M_i}, sk_{M_i}) for each model owner M_i where $i \in \{1, \dots, N\}$ and (pk_C, sk_C) for each client C . Then for the set of model owners, generate a joint key in a threshold manner, as the sum of their individual public keys $pk_M = \sum_{i=1}^N pk_{M_i}$.
- MKHE + .Enc(pk, m): Each model owner encrypts their model under the joint key pk_M and delegate them to the evaluator (if in outsourced system model, see Fig. 1c). The client encrypts their input under their public key pk_C .
- MKHE + .Extend($(pk_M, pk_C), c$): When evaluation is requested, the evaluator first extends each ciphertext under the set of the two keys $\bar{pk} = (pk_M, pk_C)$ using the algorithm MKBGV.Extend($(pk_1, \dots, pk_N), c$) described in the MKBGV scheme.
- MKHE + .Eval((\bar{c}_1, \bar{c}_2)): Computations now can be done on extended ciphertexts \bar{c}_1, \bar{c}_2 that are encrypted under the same extended key \bar{pk} . The homomorphic addition of two extended ciphertexts is performed as element-wise addition, and the homomorphic multiplication is performed as the tensor product. Multiplication increases the dimension exponentially. The key switching technique is then applied with the help of the evaluation key ek to reduce the dimension and allow further homomorphic operations.
- MKHE + .Dec($(sk_M, sk_C), \bar{c}$): Given an extended ciphertext that is encrypted under the extended key $\bar{pk} = (pk_M, pk_C)$, invoke the distributed decryption protocol between the model owners and the client to perform the decryption as the following algorithm, $Dec(\bar{s}, \bar{c}) = (\langle c_C, s_C \rangle + \sum_{i=1}^N \langle c_{M_i}, s_{M_i} \rangle \pmod{q}) \pmod{t}$.

At evaluation, the Cloud dynamically extends the ciphertexts to the concatenated key $\bar{pk} = (pk_M, pk_C)$. More specifically, each encrypted model is transformed from a ciphertext under one joint key into one encrypted under two keys, i.e., the joint key and the client's key. Similarly, the encrypted client's request is extended to the model owners' joint key. Homomorphic evaluations are done in a similar manner as described in Sec. 5.2.3. One core difference is that model owners also need to collaborate to jointly generate the evaluation key corresponding to the joint key pk_M to be used in the homomorphic multiplication. This can be done in an additional round as proposed by Asharov et al. [8] which we discuss in Sec. 5.1. Similar to ThHE and MKHE schemes, the hybrid scheme requires invoking a distributed decryption protocol so participants can jointly decrypt the encrypted result.

This hybrid approach combines ThHE and MKHE techniques to remove the requirement of generating a joint key for every client with the model owners. It also produces short extended ciphertexts since the number of involved keys is two instead of $K + 1$, which is adequate in scenarios where $K > 2$, the number of model owners is large.

6 OPEN RESEARCH PROBLEMS

In this survey, we reviewed different techniques that support homomorphic computation on data contributed by different owners and encrypted using different encryption keys. We categorized these techniques into the single-key and multi-key approaches, as illustrated in Fig. 3.

Selecting the right scheme for an application depends on the system design and requirements. For example, if the application goal is to share encrypted input between a set of authorized users, it is better to leverage PRE schemes, or IBE/ABE schemes if fine-grained access control is required. For outsourced computations, such as training an ML model based on data contributed by multiple

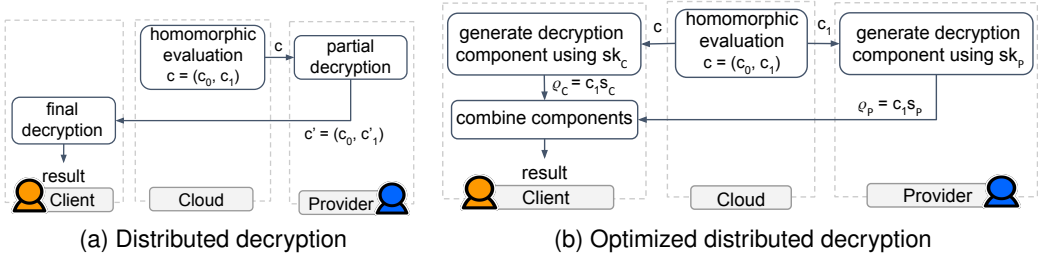


Fig. 12. Overview of different approaches for distributed decryption protocol

parties without leaking them, multi-key approaches is more suitable. It may be a better choice to leverage ThHE schemes in this case *if* participants do not change often. If they do, the efficiency quickly degrades because the joint key needs to be updated accordingly. In this case, MKHE schemes is more flexible to enable *on-the-fly* computations. Based on our observations and lesson-learned, we share open research problems and our view on methods that may overcome these problems.

6.1 Distributed decryption

In multi-key approaches discussed in Sec. 5, ciphertexts are encrypted under multiple keys – a joint key generated from multiple keys in ThHE or a concatenation of those keys in MKHE. Both ThHE and MKHE schemes share a similar requirement where all involved parties must collaborate for decryption. A trusted party can perform centralized decryption on behalf of involved parties, but it must have access to all secret keys. This setting may not be secure, especially with the increase of security breaches. Alternatively, a distributed decryption based on MPC protocol (see Sec. 3.3.3) can be invoked such that each participant partially decrypts the ciphertext with their own secret key. All participants then combine the partial decryptions to retrieve the message. In this setting, all participants learn the final decrypted result. To add an extra layer of security, the distributed decryption protocol can be made directed. This means only one *intended* participant can learn the result by keeping their partial decryption private and being the last one to perform the decryption step. One way to render a non-interactive decryption protocol is to key switch the ciphertext’s key to the intended participant’s key before returning the result. But, the key switching step is interactive, as further elaborated in Sec. 6.2.

In the outsourced system model (in Fig. 1c), the Cloud evaluator may return the resulting ciphertext to the client, who then invokes the distributed decryption protocol with other participants. This setting does not involve the Cloud in the decryption step. On the other hand, the Cloud can help and invoke the distributed decryption by sending the ciphertext directly to the other participants to perform partial decryption and return the partially decrypted ciphertext to the client, as illustrated in Fig. 12a. The client then completes the decryption with their own secret key to retrieve the result. This design involves moving a large ciphertext, containing two large components, among participants. The design can be improved, as illustrated in Fig. 12b. The Cloud returns the full ciphertext to the clients and sends only the randomness component to the participants who use their secret keys to generate decryption components and send them to the client who completes the decryption step by combining the received components. Note the improved method may have an additional round of communication, but it suits scenarios with a large number of participants because it avoids forwarding large partially decrypted ciphertexts between participants.

6.2 A combination of multi-key techniques

Most of our surveyed multi-key approaches are designed by adding extra protocols on top of some base HE schemes. These extra protocols include PRE which enables switching the key of a given ciphertext to another key without revealing the plaintext, IBE/ABE which provides access control for the homomorphic computation. ThHE and MKHE support computation with ciphertexts encrypted, respectively, under joint and concatenated keys. An interesting observation is that some of these features, such as PRE and ThHE, can be combined to obtain an encryption scheme with a more efficient design.

As mentioned earlier, PRE can be integrated with multi-key schemes such as ThHE and MKHE to avoid distributed decryption. The idea is to key switch the ciphertext from a join or concatenated key to the intended client's key. Although this method removes the interactivity within the decryption step, it still requires all participants to collaboratively generate a re-encryption key in advance either based on their secret or public key. Mouchet et al. [75] proposed methods for performing a distributed re-encryption key generation and distributed key-switching techniques.

Alternatively, we can develop hybrid MKHE approach [4] based on ThHE and MKHE techniques to obtain a more practical design with shorter extended ciphertexts. More specifically, this approach is more suitable for computation outsourcing scenarios where a fixed group of data owners wanting to offer data-driven services to customers. In this system model, the group membership of data owners does not change often hence ThHE can be used to create a joint key, whereas MKHE can support dynamic extension of this joint key to a customer's key, as we discussed in Sec. 5.3. Unfortunately, the research work on this approach is still scarce. More work is needed to address other system models with one group of users. Another open research question is how to achieve more generalized and dynamic hybrid schemes, specifically in cases where participants often change. One may investigate the generation of joint keys for subsets of participants and allow the cloud to use PRE and switch *on-the-fly* to a less number of concatenated keys to obtain shorter extended ciphertexts.

6.3 Public parameters

To facilitate computing with multiple keys in LWE-based schemes, it is common to design them in the CRS model. That is, participants use a publicly shared parameter in the form of a lattice (or ring in RLWE) element and use it in generating their public keys. This is to ensure that all keys are related, which is a requirement for correct elimination of large lattice elements during decryption. One disadvantage of this model is that it often relies on a trusted party to uniformly choose the shared element before any computation is done. Even with removing this trusted party and generating independent public keys [65], we still need to perform a *linking algorithm* on the keys to make them related. This linking algorithm is held at run time and is interactive, which increases both computation and communication overheads. Hence, a research question still stands as if we can compute efficiently with multiple keys without relying on public parameters.

Another form of public parameters that must be provided is encrypted randomness in GSW-based MKHE schemes and evaluation keys in BV-based MKHE schemes. In the MK-variants of GSW scheme, there is no need to provide evaluation keys. However, decrypting an extended ciphertext with concatenated keys fails if no information about the encryption randomness was provided. Hence, it is crucial to include additional encrypted randomness components within the extended ciphertext. Because of this requirement, the size of the ciphertexts increases quadratically at worst case [76, 81], which leads to compute on significantly high-dimensional matrices. Even with proposed optimizations using bootstrapping [23] to obtain linearly increased size, we need to bootstrap at each evaluation and the initial evaluated ciphertexts have quadratic growth.

On the other hand, MK-variants of BV schemes do not require encrypting the randomness, but they need providing evaluation keys to apply relinearization after each homomorphic multiplication as discussed in Sec. 3.3.1. Generating the evaluation keys corresponding to the concatenated key is not a trivial task. In earlier works proposed by Chen et al. [32] and Li et al. [67], this task requires combining different schemes, namely BGV and ring-GSW. However, Chen et al. [30] proposed that relinearization can be applied directly on extended ciphertexts with individually generated evaluation keys.

7 APPLICATIONS

Many outsourced computations involve sensitive data contributed by individual data owners. HE protects this sensitive data in-use without requiring decryption first. Multi-key HE extends the protection of private data to individual owners, with each possesses a private key for decrypting his/her inputs and results but nothing else from others. There are many applications that can benefit from multi-key HE. We classify these applications based on the two system models described in Section 1.1.

7.1 Computation outsourced to a cloud evaluator

Nowadays, large-scale data analysis and machine learning tasks are outsourced to the Cloud for its compelling economic efficiency and wide accessibility. Outsourcing these computational-intensive tasks to the Cloud where the homomorphically encrypted data resides can reduce the overheads of moving large ciphertext around. Many existing work proposed homomorphic algorithms based on this system model.

CryptoNets [58] and CryptoDL [63] are earlier works on supporting secure inference of encrypted neural networks that are outsourced to the Cloud. These works assumed a system model in which both network parameters and feature inputs are contributed by the same owner who just wants to offload data storage and computations to the cloud, hence these initial works assumed private data is encrypted under a single key. However, this setup is not suitable in a collaborative setting where each data owner wants to protect their private data using their own keys, as illustrated in Fig. 1c.

Wang et al. [99] proposed a framework with two non-colluding servers \mathcal{A} , \mathcal{B} to support homomorphic computation on ciphertexts encrypted under different keys. The core idea is based on proxy re-encryption (PRE) which has been discussed in Section 4.2. The framework leverages one of two proposed ElGamal variant schemes, Vitamin⁺ and Vitamin^{*}, to support partial homomorphic evaluation (addition or multiplication) on ciphertexts. In the Vitamin⁺ scheme, server \mathcal{A} can independently perform addition on ciphertexts but needs to interact with server \mathcal{B} to perform multiplication on the blinded messages. In particular, users encrypt their messages with their keys and send ciphertexts to the server \mathcal{A} . The server \mathcal{A} converts ciphertexts from under users' public keys to ones encrypted under the public key of server \mathcal{B} and locally perform homomorphic addition on ciphertexts under the same key. To evaluate multiplication, ciphertexts are first blinded and sent to the server \mathcal{B} who decrypts and perform multiplication of blinded messages in the plain. Then, the server \mathcal{B} encrypts the product with its key and returns it. The server \mathcal{A} can then remove the blinding factor and convert ciphertext results back to ones encrypted under users' keys. The same process follows in the Vitamin^{*} scheme but with changing the operations. The re-encryption and blinding operations are required due to the use of ElGamal variant schemes which cannot support homomorphic addition and multiplication on ciphertexts. These operations add significant overhead impacting the practicality of this framework. Also, many existing works explore the use of partial HE schemes to support homomorphic computation in a two-party setting (Fig. 1a). These solutions typically have limitations making them not straightforward to be generalized.

Nevertheless, the idea of using PRE to transform ciphertexts encrypted under one key to another has been adopted in later works [84, 105] to avoid the need of a trusted crypto server for decryption or invoking a distributed decryption. For example, the work of [105] leverages an MK-variant of BGV scheme to extend ciphertexts to multiple keys before evaluation. The ciphertext result, which is encrypted under concatenated keys, can be then converted via key switching to the receiver's public key so it is decryptable directly with the corresponding secret key.

Hu et al. [64] proposed homomorphic algorithms for evaluating geosocial query with user-controlled privacy on the Cloud. The authors proposed a system model that allows users to submit periodically encrypted geo-location data to a cloud evaluator (i.e., service provider). The cloud evaluator performs homomorphic evaluation of location queries upon receiving users' requests. In this case, the data privacy of individual users needs to be protected with multi-key approaches. The authors leveraged the idea of ThHE (Sec. 5.1) to support homomorphic evaluation with multiple keys and proposed the use of a semi-trusted crypto server to create joint public key using individual public key from each pair of users. Users encrypt their location data using this joint key. The homomorphically evaluated results are then sent to the crypto server for decryption, or passed around all involved parties for a distributed decryption. This work inherits the limitation of ThHE, and it is not salable to a large number of users.

Aloufi et al. [5] proposed homomorphic protocols for blindfolded evaluation of random forests that are assembled from collaborating model owners. Each model owner encrypts their decision trees under their keys and outsource them to the Cloud evaluator. The authors proposed the use of the hybrid MKHE (Sec. 5.3) to reduce the number of joint keys needed for encrypting the models while preserving the ability to dynamically extend the encrypted models to a user's public key just before evaluation. Similar design has been proposed by Chen et al. [30] for oblivious neural network inference based on encrypted inputs under multiple owners' keys. Both work require the semi-honest cloud to work with individual model owners to perform a MPC-based distributed decryption protocol.

7.2 Computation among multi-party

Earlier schemes, namely LTV [70] and Clear-McGoldrick [37], extended HE schemes to the multi-key setting. However, they assumed the existence of a trusted party that has access to all secret keys and can decrypt the evaluated result at the end. This setting is not feasible when multiple participants do not want to disclose their secret keys. Mukherjee and Wicks [76] laid the first construction of a two-round MPC protocol that leverages MKHE evaluations and enable distributed decryption. In the first round, each participant encrypts their inputs under their individually generated keys and broadcasts the ciphertexts. In the second round, each participant locally extends the ciphertexts to the additional keys (Sec. 5.2) and homomorphically evaluates the algorithm. Then, each participant uses their secret key to decrypt the locally evaluated result and broadcasts this partially decrypted result to other participants. At the end, all participants aggregate all partial decrypted result and obtain the final result. The protocol avoids delegating secret keys to a trusted party, but it requires all the parties to have access to the evaluated algorithm in order to concurrently compute the same algorithm on ciphertexts. Building on top of this construction, Gavin and Bonneval [54] proposed a system to securely aggregate testimonies from different parties in which some parties may be colluded to perturb the results. Another secure data aggregation protocol were proposed for Vehicular ad-hoc networks (VANETs) [74].

Technically, it is possible to extended most (R)LWE based HE schemes to a multiparty setting using a *universal thresholdizer* [16]. The construction is based on the linear secret sharing of key as in ThHE (Sec. 5.1). For example, Mouchet et al. [75] proposed the construction of the multiparty variant of recent (R)LWE-based schemes such as BFV [21, 49] and CKKS [34]. Fundamentally, this work shares many similarities with other ThHE scheme. Hence, it requires the collaboration

of all parties to decrypt the evaluated results or performs a collaborative key switching protocol to transform the evaluation results encrypted under a joint key to a ciphertext that is decryptable by the user's secret key.

Clinical and Genomic data is typically locked up on secure servers of individual data owners such as hospitals, biobanks. Due to privacy regulations (e.g., HIPAA, GDPR), these data owners are reluctant to outsource any private data to a third-party. Medco [84] combined HE with MPC protocol to facilitate federated processing of homomorphic encrypted data that is on-premises of individual data owners. Medco adopted ThHE to support multi-key HE, assuming the membership of these data owners does not change often. Hence, they can setup a joint public key for clients to encrypt their inputs. Note, the client inputs are protected since no single data owner can decrypt without the joint secret key which are generated using partial secret keys owned by each data owner. The encrypted client queries are sent to each data owner for homomorphic evaluation. The evaluated results are verified and passed through a distributed decryption protocol which converts the final results to be encrypted under only the client's key.

8 CONCLUSION

Many secure computation applications requires stronger security models where data are protected under different keys. In this survey, we reviewed existing techniques that support this model of computation. We categorized the techniques, based on the number of keys involved in the encryption, to single-key and multi-key approaches. Single-key approaches encrypt and process data then enable an authorized user to independently decrypt with their key. On the other hand, multi-key approaches enable a set of users to construct one joint key based on their individual keys as in ThHE schemes, or dynamically extends their individually encrypted data to multiple keys as in MKHE schemes. More recent techniques support combining different techniques for specific system models for efficiency. We also analyzed the security and complexity of different existing schemes and discussed open problems in this emerging research area.

REFERENCES

- [1] Coopetition. In *Oxford Dictionaries*. <https://www.lexico.com/en/definition/coopetition>
- [2] 2020. PALISADE Lattice Cryptography Library (release 1.7.4). <https://palisade-crypto.org/>. (Jan. 2020).
- [3] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. 2018. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 79.
- [4] Asma Aloufi and Peizhao Hu. 2019. Collaborative Homomorphic Computation on Data Encrypted under Multiple Keys. In *the International Workshop on Privacy Engineering (IWPE'19) co-located with S&P'19*.
- [5] Asma Aloufi, Peizhao Hu, Harry W. H. Wong, and Sherman S. M. Chow. 2019. Blindfolded Evaluation of Random Forests with Multi-Key Homomorphic Encryption. *IEEE Transactions on Dependable and Secure Computing* (2019), 1–1. DOI : <http://dx.doi.org/10.1109/TDSC.2019.2940020>
- [6] Jacob Alperin-Sheriff and Chris Peikert. 2014. Faster bootstrapping with polynomial error. In *Annual Cryptology Conference*. Springer, 297–314.
- [7] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. 2009. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *Advances in Cryptology – CRYPTO*. Springer, 595–618.
- [8] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. 2012. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Advances in Cryptology – EUROCRYPT*. Springer, 483–501.
- [9] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. Cryptology ePrint Archive, Report 2013/507. (2013).
- [10] Josh Benaloh. 1994. Dense probabilistic encryption. In *Proceedings of the workshop on selected areas of cryptography*. 120–128.
- [11] John Bethencourt, Amit Sahai, and Brent Waters. 2007. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE, 321–334.
- [12] Matt Blaze, Gerrit Bleumer, and Martin Strauss. 1998. Divertible protocols and atomic proxy cryptography. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 127–144.

- [13] Dan Boneh. 1998. The decision Diffie-Hellman problem. In *International Algorithmic Number Theory Symposium*. Springer, 48–63.
- [14] Dan Boneh, Xavier Boyen, and Shai Halevi. 2006. Chosen ciphertext secure public key threshold encryption without random oracles. In *Cryptographers’ Track at the RSA Conference*. Springer, 226–243.
- [15] Dan Boneh and Matt Franklin. 2001. Identity-based encryption from the Weil pairing. In *Annual international cryptology conference*. Springer, 213–229.
- [16] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. 2018. Threshold Cryptosystems from Threshold Fully Homomorphic Encryption. In *Advances in Cryptology – CRYPTO 2018 (Lecture Notes in Computer Science)*, Vol. 10991. Springer, 565–596. DOI : http://dx.doi.org/10.1007/978-3-319-96884-1_19
- [17] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. 2005. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography Conference*. Springer, 325–341.
- [18] Elena Fuentes Bongenaar. 2016. Multi-key fully homomorphic encryption report. (2016).
- [19] Joppe W Bos, Kristin Lauter, and Michael Naehrig. 2014. Private predictive analysis on encrypted medical data. *Journal of biomedical informatics* 50 (2014), 234–243.
- [20] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine Learning Classification over Encrypted Data. In *Network and Distributed System Security Symposium (NDSS)*. <https://www.ndss-symposium.org/ndss2015/machine-learning-classification-over-encrypted-data>
- [21] Zvika Brakerski. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual Cryptology Conference*. Springer, 868–886.
- [22] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science Conference (ITCS)*. ACM, 309–325.
- [23] Zvika Brakerski and Renen Perlman. 2016. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In *Annual Cryptology Conference*. Springer, 190–213.
- [24] Zvika Brakerski and Vinod Vaikuntanathan. 2011. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. IEEE, 97–106.
- [25] Zvika Brakerski and Vinod Vaikuntanathan. 2014. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.* 43, 2 (2014), 831–871.
- [26] Ran Canetti and Marc Fischlin. 2001. Universally composable commitments. In *Annual International Cryptology Conference*. Springer, 19–40.
- [27] Ran Canetti and Shafi Goldwasser. 1999. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 90–106.
- [28] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Improved bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 34–54.
- [29] Hao Chen, Ilaria Chillotti, and Yongsoo Song. 2019. Multi-Key Homomorphic Encryption from TFHE. Cryptology ePrint Archive, Report 2019/116. (2019). <https://eprint.iacr.org/2019/116>.
- [30] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2019. Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS ’19)*. ACM, New York, NY, USA, 395–412. DOI : <http://dx.doi.org/10.1145/3319535.3363207>
- [31] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from Fully Homomorphic Encryption with Malicious Security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1223–1237.
- [32] Long Chen, Zhenfeng Zhang, and Xueqing Wang. 2017. Batched Multi-hop Multi-key FHE from Ring-LWE with Compact Ciphertext Extension. In *Theory of Cryptography Conference*. Springer, 597–627.
- [33] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 360–384.
- [34] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. A full RNS variant of approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*. Springer, 347–368.
- [35] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT*. Springer, 409–437.
- [36] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. 2016. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 3–33.

- [37] Michael Clear and Ciaran McGoldrick. 2015. Multi-identity and multi-key leveled FHE from learning with errors. In *Annual Cryptology Conference*. Springer, 630–656.
- [38] Ronald Cramer, Ivan Damgård, and Jesper B Nielsen. 2001. Multiparty computation from threshold homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 280–300.
- [39] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. 1997. A secure and optimally efficient multi-authority election scheme. *Transactions on Emerging Telecommunications Technologies* 8, 5 (1997), 481–490.
- [40] Ivan Damgård. 2000. Efficient concurrent zero-knowledge in the auxiliary string model. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 418–430.
- [41] Ivan Damgård and Maciej Koprowski. 2001. Practical threshold RSA signatures without a trusted dealer. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 152–165.
- [42] Vanesa Daza, Javier Herranz, Paz Morillo, and Carla Rafols. 2007. CCA2-secure threshold broadcast encryption with shorter ciphertexts. In *International Conference on Provable Security*. Springer, 35–50.
- [43] Cécile Delerablée and David Pointcheval. 2008. Dynamic threshold public-key encryption. In *Annual International Cryptology Conference*. Springer, 317–334.
- [44] Yvo Desmedt and Yair Frankel. 1989. Threshold cryptosystems. In *Conference on the Theory and Application of Cryptology*. Springer, 307–315.
- [45] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router*. Technical Report. Naval Research Lab Washington DC.
- [46] Yevgeniy Dodis, Shai Halevi, Ron D Rothblum, and Daniel Wichs. 2016. Spooky encryption and its applications. In *Annual Cryptology Conference*. Springer, 93–122.
- [47] Léo Ducas and Daniele Micciancio. 2015. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 617–640.
- [48] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory* 31, 4 (1985), 469–472.
- [49] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. <https://eprint.iacr.org/2012/144/20120322:031216>. (March 2012).
- [50] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. 2010. Cryptography engineering. *Design Princi* (2010).
- [51] Marc Fischlin and Roger Fischlin. 2000. Efficient non-malleable commitment schemes. In *Annual International Cryptology Conference*. Springer, 413–431.
- [52] Caroline Fontaine and Fabien Galand. 2007. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security* 2007 (2007), 15.
- [53] Tan Soo Fun and Azman Samsudin. 2016. A survey of homomorphic encryption for outsourced big data computation. *KSI Transactions on Internet and Information Systems (TIIS)* 10, 8 (2016), 3826–3851.
- [54] Gerald Gavin and Stephane Bonneval. 2019. Securely Aggregating Testimonies with Threshold Multi-key FHE. In *International Conference on Codes, Cryptology, and Information Security*. Springer, 325–348.
- [55] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *STOC*, Vol. 9. 169–178.
- [56] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology—CRYPTO 2013*. Springer, 75–92.
- [57] Hossein Ghodosi, Josef Pieprzyk, and Rei Safavi-Naini. Dynamic threshold cryptosystems.
- [58] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning (ICML)*. 201–210.
- [59] David Goldschlag, Michael Reed, and Paul Syverson. 1999. *Onion routing for anonymous and private internet connections*. Technical Report. NAVAL RESEARCH LAB WASHINGTON DC CENTER FOR HIGH ASSURANCE COMPUTING SYSTEMS.
- [60] Shafi Goldwasser and Silvio Micali. 1984. Probabilistic encryption. *J. of Computer and System Sciences* 28, 2 (1984), 270–299.
- [61] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. 2015. Attribute-based encryption for circuits. *Journal of the ACM (JACM)* 62, 6 (2015), 45.
- [62] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. 2006. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*. Acm, 89–98.
- [63] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. CryptoDL: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189* (2017).
- [64] Peizhao Hu, Sherman SM Chow, and Asma Aloufi. 2017. Geosocial query with user-controlled privacy. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 163–172.

- [65] Eunkyung Kim, Hyang-Sook Lee, and Jeongeun Park. 2018. Towards round-optimal secure multiparty computations: Multikey FHE without a CRS. In *Australasian Conference on Information Security and Privacy*. Springer, 101–113.
- [66] Hyang-Sook Lee and Jeongeun Park. 2019. On the Security of Multikey Homomorphic Encryption. Cryptology ePrint Archive, Report 2019/1082. (2019). <https://eprint.iacr.org/2019/1082>.
- [67] Ningbo Li, Tanping Zhou, Xiaoyuan Yang, Yiliang Han, Wenchao Liu, and Guangsheng Tu. 2019. Efficient Multi-Key FHE With Short Extended Ciphertexts and Directed Decryption Protocol. *IEEE Access* 7 (2019), 56724–56732.
- [68] Benoît Libert and Damien Vergnaud. 2008. Unidirectional chosen-ciphertext secure proxy re-encryption. In *International Workshop on Public Key Cryptography*. Springer, 360–379.
- [69] Yehuda Lindell and Benny Pinkas. 2009. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality* 1, 1 (2009), 5.
- [70] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. 2012. 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*. ACM, 1219–1234.
- [71] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *Advances in Cryptology – EUROCRYPT*. Springer, 1–23.
- [72] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2013. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)* 60, 6 (2013), 43.
- [73] Paulo Martins, Leonel Sousa, and Artur Mariano. 2017. A Survey on Fully Homomorphic Encryption: An Engineering Perspective. *ACM Comput. Surv.* 50, 6 (Dec. 2017), 83:1–83:33. <http://doi.acm.org/10.1145/3124441>
- [74] Bo Mi, Hongyang Pan, Darong Huang, Tiancheng Wei, and Xingfeng Wang. 2019. A Secure Data Aggregation Protocol in VANETs Based on Multi-key FHE. In *International Conference on Artificial Intelligence and Security*. Springer, 178–190.
- [75] Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2020. Multiparty Homomorphic Encryption: From Theory to Practice. Cryptology ePrint Archive, Report 2020/304. (2020). <https://eprint.iacr.org/2020/304>.
- [76] Pratyay Mukherjee and Daniel Wichs. 2016. Two round multiparty computation via multi-key FHE. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 735–763.
- [77] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. 2011. Can homomorphic encryption be practical?. In *ACM workshop on Cloud Computing Security Workshop*. 113–124.
- [78] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT*. Springer, 223–238.
- [79] Rafael Pass and others. 2005. Unconditional characterizations of non-interactive zero-knowledge. In *Annual International Cryptology Conference*. Springer, 118–134.
- [80] Torben Pryds Pedersen. 1991. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 522–526.
- [81] Chris Peikert and Sina Shiehian. 2016. Multi-key FHE from LWE, revisited. In *Theory of Cryptography Conference*. Springer, 217–238.
- [82] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. 2010. Secure attribute-based systems. *Journal of Computer Security* 18, 5 (2010), 799–837.
- [83] Tal Rabin. 1998. A simplified approach to threshold and proactive RSA. In *Annual International Cryptology Conference*. Springer, 89–104.
- [84] Jean Louis Raisaro, Juan Ramón Troncoso-Pastoriza, Mickaël Misbach, João Sá Sousa, Sylvain Pradervand, Edoardo Missiaglia, Olivier Michielin, Bryan Ford, and Jean-Pierre Hubaux. 2018. MedCo: Enabling Secure and Privacy-Preserving Exploration of Distributed Clinical and Genomic Data. *IEEE/ACM transactions on computational biology and bioinformatics* 16, 4 (2018), 1328–1341.
- [85] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)* 56, 6 (2009), 34.
- [86] Amit Sahai and Brent Waters. 2005. Fuzzy identity-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 457–473.
- [87] Bruce Schneier. 2007. *Applied cryptography: protocols, algorithms, and source code in C*. John Wiley & Sons.
- [88] Berry Schoenmakers. 2011. *Threshold Homomorphic Cryptosystems*. Springer US, Boston, MA, 1293–1294. https://doi.org/10.1007/978-1-4419-5906-5_13
- [89] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
- [90] Adi Shamir. 1984. Identity-based cryptosystems and signature schemes. In *Workshop on the theory and application of cryptographic techniques*. Springer, 47–53.
- [91] Zihao Shan, Kui Ren, Marina Blanton, and Cong Wang. 2018. Practical Secure Computation Outsourcing: A Survey. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 31.

- [92] Jun Shao and Zhenfu Cao. 2009. CCA-secure proxy re-encryption without pairings. In *International Workshop on Public Key Cryptography*. Springer, 357–376.
- [93] Victor Shoup and Rosario Gennaro. 2002. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology* 15, 2 (2002), 75–96.
- [94] Nigel P Smart and Frederik Vercauteren. 2014. Fully homomorphic SIMD operations. *Designs, codes and cryptography* 71, 1 (2014), 57–81.
- [95] William Stallings. 2006. *Cryptography and network security: principles and practices*. Pearson Education India.
- [96] Damien Stehlé and Ron Steinfeld. 2010. Faster fully homomorphic encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 377–394.
- [97] Jun Tang, Yong Cui, Qi Li, Kui Ren, Jiangchuan Liu, and Rajkumar Buyya. 2016. Ensuring security and privacy preservation for cloud data services. *ACM Computing Surveys (CSUR)* 49, 1 (2016), 13.
- [98] Vinod Vaikuntanathan. 2011. Computing blindfolded: New developments in fully homomorphic encryption. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*. 5–16.
- [99] Boyang Wang, Ming Li, Sherman SM Chow, and Hui Li. 2013. Computing encrypted cloud data efficiently under multiple keys. In *2013 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 504–513.
- [100] Shuang Wang, Xiaoqian Jiang, Haixu Tang, Xiaofeng Wang, Diyue Bu, Knox Carey, Stephanie OM Dyke, Dov Fox, Chao Jiang, Kristin Lauter, and others. 2017. A community effort to protect genomic data sharing, collaboration and outsourcing. *NPJ genomic medicine* 2, 1 (2017), 1–6.
- [101] David J. Wu, Tony Feng, Michael Naehrig, and Kristin Lauter. 2016. Privately evaluating decision trees and random forests. *Proceedings on Privacy Enhancing Technologies* 4 (2016), 1–21.
- [102] Yang Yang, Xindi Huang, XiMeng Liu, Hongju Cheng, Jian Weng, Xiangyang Luo, and Victor Chang. 2019. A Comprehensive Survey on Secure Outsourced Computation and its Applications. *IEEE Access* (2019).
- [103] Yanjiang Yang, Haiyan Zhu, Haibing Lu, Jian Weng, Youcheng Zhang, and Kim-Kwang Raymond Choo. 2016. Cloud based data sharing with fine-grained proxy re-encryption. *Pervasive and Mobile computing* 28 (2016), 122–134.
- [104] Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshihara. 2013. Secure pattern matching using somewhat homomorphic encryption. In *ACM workshop on Cloud computing security workshop*. ACM, 65–76.
- [105] Satoshi Yasuda, Yoshihiro Koseki, Ryo Hiromasa, and Yutaka Kawai. 2018. Multi-key Homomorphic Proxy Re-Encryption. In *International Conference on Information Security*. Springer, 328–346.
- [106] Wenting Zheng, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2019. Helen: Maliciously secure cooperative learning for linear models. *arXiv preprint arXiv:1907.07212* (2019).