Arbitrary-Threshold Fully Homomorphic Encryption with Lower Complexity

Yijia Chang[†] and Songze Li[*,**]

[†]The Hong Kong University of Science and Technology (Guangzhou) [*]Southeast University
[**]Engineering Research Center of Blockchain Application, Supervision and Management (Southeast University), Ministry of Education
ychang847@connect.hkust-gz.edu.cn, songzeli@seu.edu.cn

Abstract

Threshold fully homomorphic encryption (ThFHE) enables multiple parties to compute functions over their sensitive data without leaking data privacy. Most of existing ThFHE schemes are restricted to full threshold and require the participation of *all* parties to output computing results. Compared with these full-threshold schemes, arbitrary threshold (ATh)-FHE schemes are robust to non-participants and can be a promising solution to many real-world applications. However, existing AThFHE schemes are either inefficient to be applied with a large number of parties N and a large data size K, or insufficient to tolerate all types of non-participants. In this paper, we propose an AThFHE scheme to handle all types of non-participants with lower complexity over existing schemes. At the core of our scheme is the reduction from AThFHE construction to the design of a new primitive called *approximate secret sharing* (ApproxSS). Particularly, we formulate ApproxSS and prove the correctness and security of AThFHE on top of arbitrary-threshold (ATh)-ApproxSS's properties. Such a reduction reveals that existing AThFHE schemes implicitly design ATh-ApproxSS following a similar idea called "noisy share". Nonetheless, their ATh-ApproxSS design has high complexity and become the performance bottleneck. By developing ATASSES, an ATh-ApproxSS scheme based on a novel "encrypted share" idea, we reduce the computation (resp. communication) complexity from $O(N^2K)$ to $O(N^2+K)$ (resp. from $O(NK)$ to $O(N+K)$). We not only theoretically prove the (approximate) correctness and security of ATASSES, but also empirically evaluate its efficiency against existing baselines. Particularly, when applying to a system with one thousand parties, ATASSES achieves a speedup of $3.83\times$ – $15.4\times$ over baselines.

1 Introduction

Threshold fully homomorphic encryption (ThFHE) allows arbitrary computations over encrypted data from *multiple* parties, without decrypting them [3, 7, 22, 35, 37, 42]. Different from traditional single-party FHE, threshold FHE supports to distribute the data-controlling authority to multiple parties, i.e., the decryption can not succeed unless enough parties participate. This provides more flexibility and induces a family of privacy-preserving computing schemes in the secure multiparty computation (MPC) setting, where multiple parties wish to evaluate a function over their joint inputs while ensuring the privacy of inputs [35]. Compared with classical MPC methods, these ThFHE-based solutions are distinguished by their low communication complexity [3] and compatibility with cloud-assisted setting such as multiple-client-single-server architecture [37]. To date, ThFHE has been implemented by open-source libraries [1, 4] and shown various promising real-world applications, including privacy-preserving machine learning [28, 29, 32, 39, 42] and medical analytics [14, 19, 21, 30, 38].

Along with the flexible controlling authority comes a new factor that should be taken into consideration: that is, how to set the threshold of participants required for decryption. Let N denote the number of parties and T denote such a threshold ($T \leq N$). Most of existing ThFHE schemes [3, 22, 37] are designed in the *full-threshold* case with $T = N$, i.e., the decryption can succeed only if *all* parties participate. Nonetheless, achieving full-threshold is often challenging in practice due to the following reasons.

- *Uninterested Parties.* The decryption process would incur high computation and communication costs of participants. Faced with these costs, a party may lose interest in the decryption result and quit the decryption process, because the result is less valuable than those costs or this party needs to handle tasks with higher priority.
- *Dropout Devices.* In practice, parties are usually some devices (e.g., smart phones and sensors) that are interconnected as a network. These devices may drop out of network due to unexpected random factors (e.g., device powering-off and loss of network connectivity) and are unable to participate in the decryption.

[*]Corresponding author: Songze Li.

- *Denial-of-Service Attacks.* The aim of denial-of-service (DoS) attack is to make some "service" unavailable to its intended users. For ThFHE, a common goal of DoS attack is to thwart parties from obtaining decryption results. To launch DoS attack against full-threshold FHE schemes, an attacker needs to invade only one party and prevent it from participating in the decryption process.

These reasons result in various types of non-participating parties, which may fail the decryption process and degrade the practicality of full-threshold FHE schemes [7, 35]. To improve the practicality of full-threshold FHE, a natural idea is to design *arbitrary*-threshold FHE (AThFHE) that allows elastic choices of threshold T from 1 to $N-1$. In this way, AThFHE can still guarantee the output delivery of decryption protocol even when $N-T$ parties do not participate.

To this end, recent works have proposed several AThFHE schemes [7,21,23,35,42] based on linear secret sharing (LSS). Notably, all of them follow the same idea called "noisy share" when devising the decryption protocols. Specifically, to prevent the secret key from leakage during the decryption process, each party needs to compute a decryption share and add a small noise to it. However, to guarantee that the aggregation of these noisy decryption shares leads to correct decryption results, the decryption requires well-crafted design that usually incurs high complexity. For example, Boneh et al. use LSS with small recovery coefficients [7, 21] to devise a AThFHE. Nonetheless, compared with single-party FHE, the decryption key share has the size of order $O(N^{4.2})$ on average. They also construct another AThFHE based on Shamir LSS, which has the optimal share-size. Still, this scheme has a huge ciphertext space with modulus $O(N \cdot (N!)^3)$ [7], incurring high complexity and difficulty for parameter instantiation. To improve the efficiency, some works propose decryption protocols with two rounds [23, 42]. The computation (resp. communication) complexity of decryption protocol is optimized to $O(N^2 \cdot K)$ (resp. $O(N \cdot K)$), where K is the size of decrypted message. Still, such a complexity is relatively high, especially when the number of parties and the length of message are both large. Mouchet et al. propose an idea to further reduce the complexity in [35] and extend this idea in Helium framework [36]. Nonetheless, this idea assumes that the set of participants is known prior to decryption, which may not hold when the non-participants are caused by random dropout devices or denial-of-service attacks.

In a word, following the "noisy share" idea, existing AThFHE schemes are either inefficient to be applied on large systems and large data size, or insufficient to handle all types of non-participants. These deficiencies severely restrict the application of AThFHE in the real world. Based on the above investigation, we are motivated to ask the questions: *Is this "noisy share" idea the only and the optimal way to design AThFHE? If not, how can we further reduce its complexity?*

In this work, we answer these questions by reducing the AThFHE design to the construction of *approximate* secret sharing (ApproxSS), a novel cryptographic primitive formulated in this paper. The "noisy share" idea followed by existing AThFHE schemes can be regarded as a specific type of construction for ApproxSS. We propose a novel idea called "encrypted share" to construct ApproxSS with computation complexity of $O(N^2+NK)$ and communication complexity of $O(N+K)$, which achieves order-wise improvements over existing AThFHE schemes. Below we summarize our main results and key contributions, and illustrate them in Figure 1.

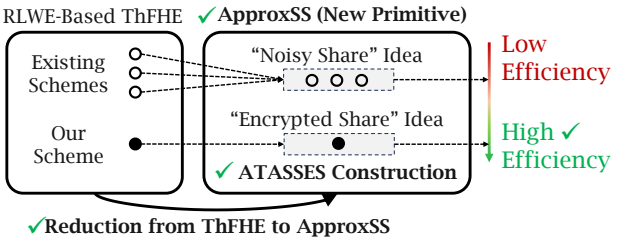


Figure 1: An overview of our main results.

- **Formulation of Approximate Secret Sharing**. Beyond vanilla secret sharing, a T-out-of-N ApproxSS has an additional operation called approximate recovery, through which any T shares can recover an approximation of the secret (a.k.a., approximate correctness), while the adversary corrupting up to $T-1$ parties can NOT learn the exact value of secret through approximate recovery (a.k.a., approximate security). Although existing works implicitly construct several ApproxSS schemes, to the best of our knowledge, this is the first work to formulate its definition and security. We believe this primitive can be of independent interest for other applications, such as the intersection of secure multi-party computation and differential privacy.

- **Reduction from RLWE-Based ThFHE to ApproxSS.** We propose a generic construction of ring-learning-with-errors (RLWE)-based ThFHE on top of ApproxSS and prove its security and correctness based on ApproxSS's properties. Particularly, on input decryption shares from any T parties, the approximate recovery in ApproxSS can output an approximate message. This approximate message can not only yield the correct plaintext by the feature of RLWE-based HE, but also protect the secret keys with the well-known smudging technique [3]. Notably, existing AThFHE schemes are special cases of this generic construction. Nonetheless, their ApproxSS design is less efficient and becomes the performance bottleneck of ThFHE.

- **An Efficient Arbitrary-Threshold ApproxSS.** In light of the above generic construction, we propose ATASSES, an Arbitrary-Threshold ApproxSS based on the idea of "Encrypted Share". This idea protects decryption shares using crafted encryption methods rather than adding noise, which helps to boost Shamir SS to ApproxSS in an efficient manner. We prove the approximate correctness and security of

ATASSES, and show that ATASSES achieves the computation complexity of $O(N^2+NK)$ and the communication complexity of $O(N+K)$.

- **Performance Evaluation through Experiments.** We implement ATASSES and existing baselines on top of Lattigo library [1]. Experimental results validate the superior performance of ATASSES. Particularly, when applying to a system with one thousand parties, ATASSES achieves a speedup of $3.83\times$ – $15.4\times$ than state-of-the-art ApproxSS. Such a speedup can be even more significant with more parties or larger data size.

The rest of this paper is organized as follows. We describe our system and threat models in Section 2. Then we provide our technical intuition in Section 3 and necessary preliminaries in Section 4. Based on these knowledge, Section 5 formulates ApproxSS and shows the generic construction of ThFHE on top of ApproxSS. Next, Section 6 proposes ATASSES and Section 7 empirically evaluates its efficiency. At last, Section 8 reviews related work and Section 9 concludes this paper.

2 System and Threat Model

System Model. As illustrated in Figure 2, we consider a group of N parties who wish to employ threshold FHE to jointly compute a function $f(\cdot)$ over their sensitive data. To facilitate the execution of threshold FHE, we assume an *aggregator* who combines cryptographic transcripts from parties and homomorphically evaluates the function $f(\cdot)$ over ciphertexts. This role can be played by any of available parties or an external server, such as the cloud. The parties (and the aggregator) are interconnected via authenticated channels. The aggregator would output one or more FHE ciphertexts, so that decrypting them leads to the desired output of function $f(\cdot)$. Throughout this paper, we use K to denote the length of function $f(\cdot)$'s output. We also assume the setting of *common reference string* (CRS), where parties can generate the same random value a from some distribution. We use $a \leftarrow\!\$\ \mathsf{CRS}$ to denote this case.

In practice, some parties may fail to participate in the decryption protocol due to losing interests, dropout devices, and/or denial-of-service attacks. We call these parties "non-participants" and other parties "participants". Nonetheless, we assume that there are at least T participants at the same time. More formally, when the decryption protocol asks parties to send some message to other parties, there are at least T parties who can upload the message within the required time. In addition, we require that a party can NOT know the set of participants until it receives the demanded messages from participants. The reason is that the non-participants may be unable to inform other parties due to random factors or even intentional attacks. In these cases, no one can assure which parties will succeed to upload the demanded messages. We remark that this is one key difference between our setting and existing work [35], which makes our work more practical.

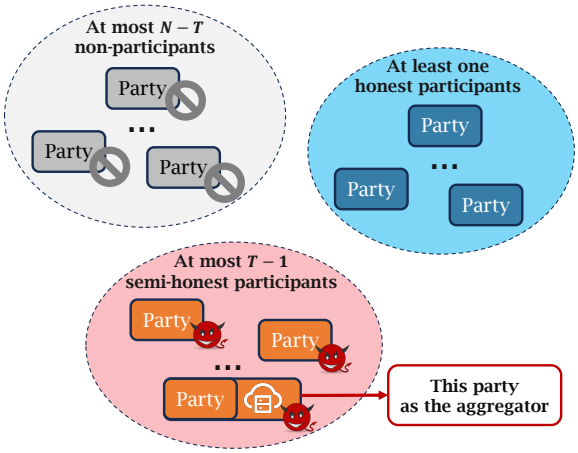


Figure 2: System model with three types of parties.

Threat Model. We consider a static semi-honest adversary who can corrupt a fixed set of up to $T-1$ parties, *i.e.*, these $T-1$ parties faithfully run the algorithms and protocols in ThFHE schemes, but the adversary can see their internal states to infer private messages. Such a setting implies that there exists at least one honest participant. In addition, we do not assume a trustworthy aggregator, i.e., the aggregator is corrupted by the adversary.

We note that our threat model, combined with the non-participants setting in the system model, is in fact a mixed adversary setting. Particularly, those non-participants can be regarded as being corrupted by a partly-malicious adversary, who can only abort at most $N-T$ parties simultaneously. We also note that recent work [13] makes the first move to shield the ThFHE schemes against malicious adversaries. Despite of its great success, they assume that the parties do not refuse to participate in the protocol execution. Since our work considers the non-participant problem, we believe our work can help to remove this assumption and boost ThFHE schemes with fully-malicious security. We provide a discussion on how to extend our scheme to fully-malicious security in Appendix E.

3 Technical Intuition

This section introduces the intuition behind our techniques on a high level. For brevity's sake, the notation in this section are highly intuitive but somewhat informal.

Problem Description. At the core of T-out-of-N ThFHE construction is to design a decryption protocol that allows any T participants to decrypt without disclosing the secret key. Suppose that the secret key is sk and the ciphertext is ct, the decryption algorithm in existing FHE schemes is usually to compute a linear function $b = \mathsf{ct}\cdot\mathsf{sk}$. In current lattice-based FHE schemes, the ct contains an error e_{ct}. The plaintext can be successfully decoded from b by rounding and/or modular reduction if the value of e_{ct} is less than some bound B.

Since the decryption is to compute a linear operation, existing ThFHE construction usually applies *linear secret sharing* technique to sk. This will generate N key shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$, one for each party. To decrypt ct, each party i computes a decryption share $b_i = \mathsf{ct} \cdot \mathsf{sk}_i$. By the linearity of secret sharing, decryption shares b_is are the shares of b, and the aggregator can recover b from any T decryption shares. For example, the full-threshold ThFHE adopts *additive* secret sharing so that $\mathsf{sk} = \sum_{i=1}^{N} \mathsf{sk}_i$. After receiving N decryption shares ($T = N$), the aggregator can compute $\sum_{i=1}^{N} b_i = \mathsf{ct} \cdot \sum_{i=1}^{N} \mathsf{sk}_i = \mathsf{ct} \cdot \mathsf{sk} = b$. However, this construction is insecure, as b leaks information of sk to the aggregator. To solve this issue, a common solution is noise smudging technique [3]. Specifically, the decryption protocol can recover $b' = b + n_{sm}$ for some random smudging noise n_{sm} rather than b. As long as the value of $n_{sm} + e_{ct}$ is bounded by B, the decryption can still succeed.

To do so, existing ThFHE constructions ask each party i to sample a local noise $n_{sm,i}$ and compute a noisy share $b'_i = b_i + n_{sm,i}$. We call this idea "noisy share". By the linearity of secret sharing, noisy shares $b'_i = b_i + n_{sm,i}$ can be regarded as the shares of $b + n_{sm}$ for some n_{sm} that is determined by $n_{sm,i}$s. In existing ThFHE constructions, the aggregator recovers $b + n_{sm}$ as b' from any T noisy shares. For example, in the full-threshold case, the recovery algorithm is simply summing all shares up. Then the aggregator will compute $b' = \sum_{i=1}^{N} b'_i = b + \sum_{i=1}^{N} n_{sm,i} = b + n_{sm}$, where $n_{sm} = \sum_{i=1}^{N} n_{sm,i}$.

Still, this "noisy share" idea faces a challenge. Recall that the value of $n_{sm} + e_{ct}$ should be bounded by B for successful decryption. Hence, this idea needs to guarantee that the value of n_{sm} is bounded by B_{sm} with $B_{sm} + \|e_{ct}\| < B$. For full-threshold FHE, this challenge can be easily tackled by setting the value of each local noise $n_{sm,i}$ bounded by B_{sm}/N. Under this setting, we have $\|n_{sm}\| < \sum_{i=1}^{N} \|n_{sm,i}\| < B_{sm}$. Nonetheless, this challenge becomes particularly hard for arbitrary-threshold FHE. Note that $n_{sm,i}$s are locally sampled by each party i and have no natural correlation. Without crafted design on $n_{sm,i}$s, the corresponding n_{sm} may have random values without bound. For example, when using the well-known Shamir secret sharing, we have $n_{sm} = \sum_{i \in \mathcal{T}} L_i^{(\mathcal{T})} n_{sm,i}$, where $\mathcal{T}$ is a set of at least T parties and $L_i^{(\mathcal{T})}$s are Lagrange coefficients associated with set $\mathcal{T}$. Since Lagrange coefficients have arbitrarily-large value, they may blow the noise n_{sm} up and fail the decryption.

Following the "noisy share" idea, existing ThFHE schemes try to solve this "blown-up noise" problem by restricting the value of $n_{sm,i}$ in various ways. Unfortunately, these solutions either suffer from high complexity or rely on an additional assumption. Particularly, without assuming that the knowledge of participant set $\mathcal{T}$ is available before decryption, the state-of-the-art solution [23, 42] has the computation complexity of $O(N^2K)$ and communication complexity of $O(NK)$. Readers can find more detailed discussion in Section 6.1.

Our Solution. Our first observation is that "noisy share" is not the only way to apply the noise smudging technique for ThFHE construction. Basically, the noise smudging technique inspires us that recovering the exact value of b is neither secure nor necessary. Instead, the aggregator only needs to recover an approximate value $b' = b + n_{sm}$ so that 1) n_{sm} has a bounded value and 2) n_{sm} is random that cannot be determined by less than T parties. We realize that vanilla secret sharing is insufficient to meet these two requirements and thus call for a novel cryptographgic primitive. As shown in Figure 3, vanilla SS only enables the trustworthy dealer to exactly recover the secret (i.e., b), while now an untrusted aggregator needs to recover the secret in an approximate manner (i.e., b').

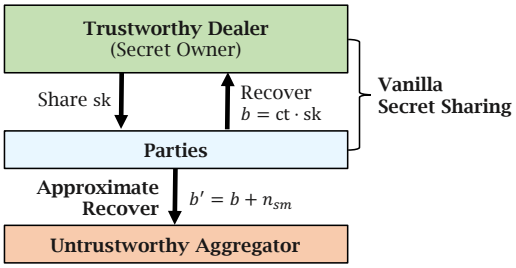


Figure 3: An illustration of approximate secret sharing.

Hence, beyond the original functions of vanilla secret sharing, the new primitive should support a novel function called "approximate recovery". This function should enable the aggregator to learn the value of b' from any T shares of b, such that 1) b' is "close" to b in the sense that $\|b' - b\|$ is bounded and 2) the adversary corrupting less than T parties cannot determine the value of $b' - b$. We term this primitive *approximate secret sharing* (ApproxSS) and formulate it in Section 5.1. We further show that a T-out-of-N ThFHE scheme can be constructed based on a T-out-of-N ApproxSS scheme in Section 5.2. Notably, the noisy share idea is an implicit construction of this primitive. However, their ApproxSS constructions has become the performance bottleneck of ThFHE scheme due to the high complexity. This inspires us to study more efficient ApproxSS construction.

Our second observation is another way to boost Shamir secret sharing to ApproxSS. This idea relies on an encryption scheme that can enable the following three steps.

Step 1: Each party i locally encrypts b_i and $n_{sm,i}$, respectively, and sends the ciphertexts to the aggregator.

Step 2: After receiving the ciphertexts of b_is and $n_{sm,i}$s from at least T parties in $\mathcal{T}$, the aggregator will learn the set $\mathcal{T}$ and further compute the corresponding $L_i^{(\mathcal{T})}$s. The encryption scheme should be somewhat linearly-homomorphic, so that the aggregator can obtain a new ciphertext of $b' = \sum_{i \in \mathcal{T}} (L_i^{(\mathcal{T})} b_i + n_{sm,i})$ from the ciphertexts of b_is and $n_{sm,i}$s.

Step 3: Any T parties collaboratively decrypt the ciphertext of b', without leaking individual b_i or $n_{sm,i}$. Notably, these T parties can be totally different from the set $\mathcal{T}$ in the first step.

We call this idea "encrypted share" and provide a specific

construction based on BFV secret-key encryption in Section 6. We argue that this idea has at least the following three advantages, especially against the noisy share idea.

- Our idea avoids the "blown-up noise". By encrypting b_i and $n_{sm,i}$, respectively, the aggregator can compute their linear combination with different coefficients, e.g., $L_i^{(\mathcal{T})} \cdot b_i + n_{sm,i}$. For comparison, in the noisy share idea, the $b_i + n_{sm,i}$ is uploaded to the aggregator as a whole. If the aggregator multiplies it with $L_i^{(\mathcal{T})}$ to recover b, the $n_{sm,i}$ is also multiplied by $L_i^{(\mathcal{T})}$, resulting in the blown-up noise. In addition, we note that in our idea, the Lagrange coefficients will blow up the errors in BFV ciphertexts. If the error exceeds some bound, the decryption also fails. Nonetheless, as we will show in Section 6.2, this problem can be solved by selecting a slightly larger ciphertext space with modulus $O(N)$.
- Although our idea also requires to share and recover (encryption and decryption) keys with $O(N^2)$ computation complexity and $O(N)$ communication complexity, these expensive operations only involve the keys whose size is a *constant* independent from the length of ciphertexts K. As a result, the total computation (resp. communication) complexity is $O(N^2+NK)$ (resp. $O(N+K)$). For comparison, the state-of-the-art solution [23,42] following the noisy share idea requires to share and recover noises whose length is the same as ciphertexts and thus its computation (resp. communication) complexity is $O(N^2K)$ (resp. $O(NK)$).
- Our idea does not require to learn the set of participants in advance. In Step 2, only *after* the aggregator has received the ciphertexts from the parties in $\mathcal{T}$, the aggregator needs to know the set $\mathcal{T}$ and $L_i^{(\mathcal{T})}$s. Moreover, the participants in Step 1 are allowed not to participate Step 3. These settings match our system model on (non)-participants. Despite an existing work [35, 36] based on the noisy share idea has lower complexity, it relies on an extra assumption than our idea, i.e., the prior knowledge of participant set is available.

By the theoretical analysis in Section 6.3 and empirical evaluation in Section 7, we verify the superior performance of our BFV-based construction. Still, our construction may not be the optimal one following the encrypted share idea. We expect future works to continuously improve its efficiency by using more ingenious encryption scheme.

4 Preliminaries

4.1 Notation

For a number M of two-power, this paper considers polynomial rings $\mathcal{R}_M = \mathbb{Z}[x]/(x^M+1)$ and $\mathcal{R}_{M,Q} = \mathcal{R}_M/Q\mathcal{R}_M$, and adopts bold letters to denote polynomials (e.g., $\mathbf{s}$). For a polynomial $\mathbf{s}$, its coefficient is denoted by s_i and its infinity norm is defined as $\|\mathbf{s}\| = \max_i |s_i|$. The sets are denoted by calligraphic letters (e.g., $\mathcal{S}$) except two exceptions. One is $[N] = \{1, \cdots, N\}$ for integer N. The other is negl that denotes the set of negligible functions (with respect to some security parameter). In addition, $a \leftarrow\!\$\, \chi$ and $a \leftarrow\!\$\, \mathcal{M}$ denote that a is sampled from a distribution χ or the uniform distribution over the set $\mathcal{M}$, respectively. We also use $\mathsf{Uniform}(\mathcal{M})$ to denote the uniform distribution over the set $\mathcal{M}$.

4.2 Vanilla Secret Sharing

Secret sharing (SS) enables a dealer to distribute its secret message m to other parties in a secure manner [41]. In this work, we consider T-out-of-N secret sharing. Its goal is to enable any set of at least T shares to recover the message m (i.e., correctness), while prevent the set of less than T shares from leaking anything about m (i.e., security). The definition of secret sharing is provided as follows.

Definition 1 (Vanilla Secret Sharing) *Given a message space $\mathcal{M}$, a T-out-of-N secret sharing scheme is a pair of PPT algorithms $(\mathtt{Share}, \mathtt{Rec})$ defined as follows:*
***Share**: $\{s_i\}_{i\in[N]} \leftarrow \mathtt{Share}(m)$. On input a message $m \in \mathcal{M}$, this algorithm outputs N shares $\{s_i\}_{i\in[N]}$;*
***Recovery**: $m \leftarrow \mathtt{Rec}(\{s_i\}_{i\in\mathcal{T}})$. On input shares $\{s_i\}_{i\in\mathcal{T}}$ from a set $\mathcal{T}$, this algorithm outputs a message $m \in \mathcal{M}$.*

In addition, some SS schemes satisfy a special property called linearity [24]. This property allows each share to contain multiple pieces and enables the message to be recovered as the linear combination of pieces from at least T shares. More formally, the recovery of message m is done as $m = \sum_{i\in\mathcal{T}} \sum_{l\in\mathcal{L}_i} w_{i,l} \cdot s_{i,l}$, where $\mathcal{T}$ is the set of at least T shares, $\mathcal{L}_i$ is the set of pieces in s_i, $s_{i,l}$ is the l-th piece of s_i, and $w_{i,l}$ is called the recovery coefficients. Linearity enables a convenient method to compute the share of the linear combination of messages. Particularly, given that the i-th share of messages $\{m_1, \ldots, m_K\}$ is $\{s_{1,i}, \ldots, s_{K,i}\}$, the i-th share of another message $m' = \sum_{k\in[K]} b_k m_k$ can be computed as $s'_i = \sum_{k\in[K]} b_k s_{k,i}$.

This work mainly uses Shamir SS [41], in which each share has only one piece. Suppose that each party i owns a share s_i, then the message m is recovered as $m = \sum_{i\in\mathcal{T}} L_i \cdot s_i$. Here, $\mathcal{T}$ can be any set of at least T parties and $L_i^{(\mathcal{T})}$s are called Lagrange coefficients that are computed as $L_i^{(\mathcal{T})} = \prod_{j\in\mathcal{T}, j\neq i} [x_j/(x_j - x_i)]$, where x_i is a number corresponding to party i.

4.3 RLWE-Based Homomorphic Encryption

Homomorphic encryption (HE) is a special type of encryption that permits computations over ciphertexts without decryption. In this work, we focus on the HE schemes whose security is based on the hardness of ring-learning-with-errors (RLWE)

problem, such as BGV [10], BFV [27], and CKKS [15–17]. These schemes can support additions and a bounded number of multiplications with high parallelized efficiency. By using the bootstrapping technique, these schemes can even evaluate an unbounded number of multiplications. Below we present BFV scheme. This scheme will be used as a concrete instance throughout this paper to introduce our scheme. Nonetheless, our work can be applied to other RLWE-based schemes, or even be naturally extended to other lattice-based HE schemes, such as Torus-FHE [18] and FHEW [25].

In practice, the plaintext space and ciphertext space of BFV are polynomial rings $\mathcal{R}_{M,P}$ and $\mathcal{R}_{M,Q}$, respectively. BFV involves an error distribution $\chi_{e,B}$, which is usually a discretized Gaussian distribution with norm bounded by B. Next, we describe the concrete algorithms of BFV scheme.

Key Generation: $\mathsf{sk}, \mathsf{pk}, \mathsf{evk} \leftarrow \mathtt{KGen}()$. This algorithm outputs secret key sk, public key pk, and evaluation key evk.

Secret-Key Encryption: $\mathsf{CT} \leftarrow \mathtt{SKEnc}(\mathsf{sk}, \mathbf{m}, \mathbf{a})$. On input sk, a message $\mathbf{m} \in \mathcal{R}_{M,P}$, and a random polynomial $\mathbf{a} \leftarrow\!\!\$\ \mathcal{R}_{M,Q}$, this algorithm outputs a pair $\mathsf{CT} = (\mathsf{CT}[0], \mathsf{CT}[1])$ as the ciphertext.

Public-Key Encryption: $\mathsf{CT} \leftarrow \mathtt{PKEnc}(\mathsf{pk}, \mathbf{m})$. On input pk and a message $m \in \mathcal{R}_{M,P}$, this algorithm outputs a pair $\mathsf{CT} = (\mathsf{CT}[0], \mathsf{CT}[1])$ as the ciphertext.

Homomorphic Evaluation: $\mathsf{CT}' \leftarrow \mathtt{Eval}(\mathsf{evk}, \{\mathsf{CT}_i\}_{i\in[N]}, f(\cdot))$. On input evk, N ciphertexts $\{\mathsf{CT}_i\}_{i\in[N]}$, and an N-input function $f(\cdot)$, this algorithm outputs a new ciphertext CT'.

Decryption: $\mathbf{m} \leftarrow \mathtt{Dec}(\mathsf{sk}, \mathsf{CT})$. On input sk and ciphertext $\mathsf{CT} \in \mathcal{R}_{M,Q}$, this algorithm outputs a message $\mathbf{m} \in \mathcal{R}_{M,P}$.

In the BFV scheme, the ciphertext $\mathsf{CT} = (\mathsf{CT}[0], \mathsf{CT}[1])$ of a message $\mathbf{m}$ under secret key sk has the structure of $\mathsf{CT}[0] + \mathsf{CT}[1] \cdot \mathsf{sk} = \Delta \cdot \mathbf{m} + \mathbf{e}_{\mathsf{CT}}$, where $\Delta = \lfloor Q/P \rfloor$ and $\mathbf{e}_{\mathsf{CT}}$ is the error in ciphertext. For example, the secret key encryption $\mathtt{SKEnc}(\mathsf{sk}, \mathbf{m}, \mathbf{a})$ is performed by sampling an error $\mathbf{e} \leftarrow\!\!\$\ \chi_{e,B}$ and computing $\mathsf{CT}[0] = \mathbf{a} \cdot \mathsf{sk} + \Delta \cdot \mathbf{m} + \mathbf{e}$ and $\mathsf{CT}[1] = -\mathbf{a}$.

Decryption and Its Requirement. By the ciphertext structure, the decryption is done in two steps. The first step is to compute $\mathbf{b} = \mathsf{CT}[0] + \mathsf{CT}[1] \cdot \mathsf{sk}$, which is equal to $\Delta \cdot \mathbf{m} + \mathbf{e}_{\mathsf{CT}}$. The second step is to decode $\mathbf{m}$ from $\mathbf{b}$ by rounding and modular reduction. Notably, the decryption, and in particular, the second step, can succeed if and only if the error has a bounded value, i.e., $\|\mathbf{e}_{\mathsf{CT}}\| < \Delta/2$.

4.4 (Public-Key) Threshold FHE

Traditional FHE is typically used in the single-server-single-client architecture. The client is the owner of private data and the server is used to evaluate some function over the ciphertexts of private data. In this architecture, the client owns the secret key, along with the full authority to decrypt all data. Nonetheless, in some scenario, the data may come from multiple clients and the decryption authority should be distributed among them. Threshold fully homomorphic encryption (ThFHE) is proposed for such a scenario. Particularly, almost all ThFHE schemes require public-key encryption, as the clients are not allowed to own the secret key and can only use public key for encryption. Although ThFHE has the same pipeline as traditional FHE (i.e., secret/public/evaluation key generation, encryption, homomorphic evaluation, and decryption), these operations are modified as follows to distribute the decryption authority.

For *key generation*, ThFHE asks each party to sample a local secret key and the (implicit) global secret key is the summation of all local secret keys. In addition, to support *public-key* encryption and evaluation, all parties need to cooperate to generate global public/evaluation keys from their local secret keys. To this end, the key generation algorithms are modified to multi-party protocols. These key generation protocols are executed only once in the setup stage. Once these global public encryption and evaluation keys are generated, they are fixed and will be reused for encryption and evaluation in the subsequent computation stage. Since then, we follow the setting of existing work [7, 37] and assume that all parties remain available in the setup stage. For *decryption*, ThFHE also modifies it to a multi-party protocol. This protocol involves a threshold parameter T ($T \leq N$): the decryption can succeed if and only if at least T parties participate. We call such a scheme T-out-of-N ThFHE. Correspondingly, the ThFHE schemes whose decryption relies on the participation of *all* parties are called *full*-threshold schemes with $T = N$.

We note that the decryption protocol can be re-executed to decrypt multiple ciphertexts. When decrypting, the parties who participate in the key generation protocol may fail to participate in decryption. Hence, we prefer those ThFHE schemes who can successfully decrypt even if only *part* of parties participate. These preferred schemes should allow the elastic choice of T according to the system's characteristics. Particularly, if there are at most N' non-participants in the system, then the setting of $T = N - N'$ can support to decrypt. We call such a scheme arbitrary-threshold FHE (AThFHE). How to design AThFHE, and, in particular, its decryption protocol, is the core problem studied in this paper.

5 ThFHE via ApproxSS

In this section, we formulate approximate secret sharing (ApproxSS) in Section 5.1 and establish a reduction from ThFHE construction to the design of ApproxSS in Section 5.2.

5.1 Formulation of ApproxSS

Below we formalize a novel primitive called approximate secret sharing (ApproxSS). Compared with vanilla SS, a T-out-of-N ApproxSS has an additional operation called *approximate recovery*. Informally, this operation recovers an approximate message from any T shares of the message, such that 1) the approximate message is "close" to the original message and 2) the adversary corrupting $T-1$ parties can

not learn "too much information" about the original message via this operation. To formulate this operation, a direct way is to restrict it as a one-shot algorithm like $\texttt{Share}$ and $\texttt{Rec}$ in vanilla SS. Still, to thoroughly explore all possible constructions, we allow this operation to take multiple rounds, during which parties can interact with each other. Hence, we formulate the approximate recovery as a multi-party protocol and define ApproxSS as follows.

Definition 2 (Approximate Secret Sharing) *Given a message space $\mathcal{M}$, a T-out-of-N approximate secret sharing scheme is a pair $(\texttt{Share}, \Pi_{\texttt{ApproxRec}})$ defined as follows:*
***Share**: $\{s_i\}_{i\in[N]} \leftarrow \texttt{Share}(m)$. On input a message $m \in \mathcal{M}$, this algorithm outputs N shares $\{s_i\}_{i\in[N]}$;*
***Approximate Recovery**: $m' \leftarrow \Pi_{\texttt{ApproxRec}}(\{s_i\}_{i\in\mathcal{T}}, \chi)$. On input shares $\{s_i\}_{i\in\mathcal{T}}$ from a set $\mathcal{T}$ and a distribution χ on $\mathcal{M}$, this multi-party protocol outputs a common approximate message $m' \in \mathcal{M}$ to all parties.*

In the above definition, the distribution χ is used to characterize how much information about m is allowed to be leaked via the execution of approximate recovery. In other words, an adversary who corrupts $T-1$ parties cannot learn anything about m other than a random value $m_\chi = m + x$ with $x \leftarrow\!\$\ \chi$ and what is implied by this value. For example, if $m = 8$ and χ is a uniform distribution over $\{-1, 0, 1\}$, then the adversary could learn $m_\chi = 9$ and further infer $m = 8, 9, 10$ with equal probability, but nothing else. We term this property of ApproxSS *approximate security*. To formulate this property, we introduce two experiments $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}(\chi)$ and $\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}(\chi)$, as described in Figure 4. Particularly, the $\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}(\chi)$ is the execution of ApproxSS over message m and distribution χ. In contrast, the view of adversary in $\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}(\chi)$ is simulated from either public parameters (i.e., $N, T, \mathcal{M}, \mathcal{T}_r$), the internal states of corrupted parties (i.e., $\{s_i\}_{i\in\mathcal{N}_\mathcal{A}}$), or a random value $m_\chi = m + x$ with $x \leftarrow\!\$\ \chi$. Hence, if the adversary fails to distinguish between two experiments, then it can learn nothing about m except that m can be a random value $m_\chi - x$ with $x \leftarrow\!\$\ \chi$, implying the approximate security of ApproxSS. We formalize the above discussion as follows.

Definition 3 (Approximate Security) *For an ApproxSS whose $\Pi_{\texttt{ApproxRec}}$ consists of R rounds, it satisfies χ-approximate security if the following claim holds: There exists $R+1$ PPT simulator algorithms $(S_0, S_1, \ldots, S_R)$ such that for any PPT adversary $\mathcal{A}$, the experiments $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}(\chi)$ and $\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}(\chi)$ in Figure 4 are indistinguishable.*

We note that there exists a trivial solution for approximate recovery construction: a party directly outputs the approximate message as a uniformly-random value over $\mathcal{M}$. In this way, no information about message m is leaked. Nonetheless, this construction is also somewhat meaningless. When applying ApproxSS in the real-world applications (e.g., the ThFHE construction in this work), we usually need the approximate message to be "close" to the original message.

$\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}(\chi)$: Real-World Experiment

1 : $\mathcal{A}$ outputs $N, T, \mathcal{M}$, a message $m \in \mathcal{M}$, and a set $\mathcal{N}_\mathcal{A}$ of $T-1$ corrupted clients
2 : The challenger runs $\{s_i\}_{i\in[N]} \leftarrow \texttt{Share}(m)$ and sends $\{s_i\}_{i\in\mathcal{N}_\mathcal{A}}$ to $\mathcal{A}$
3 : The $\Pi_{\texttt{ApproxRec}}$ protocol is performed as follows:
In each round r, $\mathcal{A}$ selects a set $\mathcal{T}_r$ consisting of T participants, then the challenger runs the r-th round of the protocol for parties in $\mathcal{T}_r$ and provides the transcript Trans_r that can be observed by parties in $\mathcal{N}_\mathcal{A}$ to $\mathcal{A}$
4 : At the end, $\mathcal{A}$ outputs a distinguishing bit b

$\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}(\chi)$: Ideal-World Experiment

1 : $\mathcal{A}$ outputs $N, T, \mathcal{M}$, a message $m \in \mathcal{M}$, and a set $\mathcal{N}_\mathcal{A}$ of $T-1$ corrupted clients
2 : The challenger runs $\{s_i\}_{i\in\mathcal{N}_\mathcal{A}} \leftarrow S_0(N, T, \mathcal{M})$ and sends $\{s_i\}_{i\in\mathcal{N}_\mathcal{A}}$ to $\mathcal{A}$
3 : The challenger samples $m_\chi \leftarrow m + x$ with $x \leftarrow\!\$\ \chi$ and then simulates the $\Pi_{\texttt{ApproxRec}}$ protocol as follows:
In each round r, $\mathcal{A}$ selects a set $\mathcal{T}_r$ consisting of T participants, then the challenger generates transcript Trans'_r by running $\mathsf{Trans}'_r \leftarrow S_r(m_\chi, \{s_i\}_{i\in\mathcal{N}_\mathcal{A}}, \mathcal{T}_r)$ and sends Trans'_r to $\mathcal{A}$
4 : At the end, $\mathcal{A}$ outputs a distinguishing bit b'

Figure 4: Description of $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}(\chi)$ and $\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}(\chi)$. Their differences are highlighted by red, underlined parts.

This property is called *approximate correctness*. To formulate this property, we use a subset $\mathcal{M}_B \subsetneqq \mathcal{M}$ to denote the range of allowed difference between approximate message m' and original message m. The common setting of $\mathcal{M}_B$ contains the elements whose value is less than or equal to a given number B. For example, if $m = 8$ and $B = 1$, then $\mathcal{M}_B = \{-1, 0, 1\}$ and m' should belong to $\{7, 8, 9\}$. With the notion of $\mathcal{M}_B$, the approximate correctness is formally defined as follows.

Definition 4 (Approximate Correctness) *An ApproxSS satisfies $\mathcal{M}_B$-approximate correctness if and only if for the shares of any message $\{s_1, \ldots, s_N\} \leftarrow \texttt{Share}(m)$ and any set $\mathcal{T}$ of at least T parties, the approximate message $m' \leftarrow \Pi_{\texttt{ApproxRec}}(\{s_i\}_{i\in\mathcal{T}}, \chi)$ satisfies $m' - m \in \mathcal{M}_B$.*

Notably, the support[1] of distribution χ is not necessarily the same as the set $\mathcal{M}_B$. For example, when χ is the distribution

[1] Roughly speaking, for a random variable, the support of its distribution is the set of its possible values with non-zero probability. In this paper, we mainly consider a discrete random variable $x \in \mathcal{M}$. In this case, the support of its distribution χ is defined as the set $\{\tilde{x} \in \mathcal{M} \mid P(x = \tilde{x}) > 0\}$.

over $\{0\}$ with $P(x=0)=1$, we can construct an ApproxSS scheme whose $\Pi_{\mathsf{ApproxRec}}$ protocol outputs $m+1$ by modifying a vanilla SS scheme: after the aggregator recovers m via the recovery algorithm, it randomly outputs m, $m+1$, and $m-1$ instead of m. This protocol can only satisfies $\mathcal{M}_B$-approximate security with $\mathcal{M}_B=\{-1,0,1\}$, while this $\mathcal{M}_B$ is not equal to the support of χ (i.e., $\{0\}$). Nonetheless, we also note that there could exist some relations between the support of χ and the set $\mathcal{M}_B$. For example, when $\mathcal{M}_B=\{0\}$, the approximate recovery protocol must output m. In this case, the adversary can learn the exact value of m and the best we can expect for χ-approximate security is χ over $\{0\}$ with $P(x=0)=1$. Since this paper focuses on the ApproxSS's efficient construction and application for ThFHE, we left the studies on relations between χ and $\mathcal{M}_B$ as an interesting problem for future research.

5.2 ApproxSS-Based ThFHE Construction

Next, we show that a T-out-of-N ThFHE scheme can be constructed based on a T-out-of-N ApproxSS scheme. Such a construction includes two modifications on existing full-threshold FHE schemes. One is to add a step called secret key sharing to the key generation protocol and the other is to re-devise the decryption protocol. Below we present these two modifications and illustrate them in Figure 5.

Key Generation Protocol. In this work, we consider the *public-key* ThFHE construction, in which the key generation protocol consists of two parts: local secret key sampling and global public key generation. For local secret key sampling, each party i produces its secret key $\mathsf{sk}_i \leftarrow \mathtt{BFV.SKGen}()$. For global public key generation, parties collaborate to generate a global public/evaluation key. These keys are published to all parties and will be used for encryption/evaluation later.

Recall that we assume all parties are available for key generation. Under this setting, we can employ the key generation protocol of existing full-threshold schemes in our construction, without requiring many modifications. In these schemes, by well-crafted design, the global public keys are almost equivalent to the output of single-party FHE's key generation algorithm, with the corresponding global secret key sk being the summation of local secret keys from *all* parties, i.e., $\mathsf{sk}=\sum_{i\in[N]}\mathsf{sk}_i$. Under this design, the encryption and evaluation algorithms are just the same as those of the single-party FHE. Meanwhile, the decryption protocol will rely on the local secret keys from all parties. If a local secret key is only known by its owner (say party i), then the decryption protocol can not decrypt once party i does not participate. To avoid this situation, we add a new step called `SKShare` as the third part of key generation protocol.

As specified in Figure 5, this step is executed by every party i after its local secret key sk_i is generated. Through `SKShare`, party i invokes the Share algorithms of the ApproxSS scheme to generate T-out-of-N shares of its local secret key sk_i (Line 1). We particularly note that the message space of ApproxSS should be the ciphertext space of BFV, as the secret key lies in this space. Then the party i sends its generated shares to other parties and receives the shares of local secret keys from other parties (Line 2-4). By the linearity of ApproxSS, the summation of the i-th share of local secret keys is the i-th share of the summation of local secret keys. Recall that the global secret key is exactly the summation of local secret keys. By summing all received shares up, party i generates its T-out-of-N share of global secret key (Line 5-6). Notably, any T shares contain enough information of global secret key and can enable the decryption to succeed. Therefore, through this secret key sharing step, the decryption protocol will not require all parties to participate.

Decryption Protocol. The decryption protocol can be used to decrypt a long message from multiple ciphertexts. To highlight our key idea, we only discuss the case with one ciphertext $\mathsf{CT}=(\mathbf{c}_0,\mathbf{c}_1)$ and put the full description with multiple ciphertexts in Figure 5. Besides the ciphertext, the decryption protocol needs two public parameters as input: the BFV parameter Δ and the norm bound of smudging noises B_{sm}. For ease of description, we decompose the decryption into three successive phases. Phases 1 and 3 are locally executed by every party and the aggregator, respectively, while Phase 2 is an interactive protocol between them.

Phase 1: Computation of Decryption Share. Recall that by the `SKShare` step in key generation protocol, each party i has learned a T-out-of-N share $\mathsf{skShare}_i$ of global secret key sk. With the ciphertext $\mathsf{CT}=(\mathbf{c}_0,\mathbf{c}_1)$, the knowledge of this share enables party i to compute $\mathbf{b}_i=\mathbf{c}_0+\mathbf{c}_1\cdot\mathsf{skShare}_i$. By the linearity of ApproxSS, $\mathbf{b}_i$ is a T-out-of-N share of $\mathbf{b}=\mathbf{c}_0+\mathbf{c}_1\cdot\mathsf{sk}$. Since $\mathbf{b}$ is a desired value for BFV decryption, we call $\mathbf{b}_i$ the decryption share of party i.

Phase 2: Recovery of Approximate Decryption. A natural idea for decryption is to recover $\mathbf{b}$ from its shares computed in Phase 1 and further decode message $\mathbf{m}$ from $\mathbf{b}$. However, such a recovery process may result in the leakage of secret key: since $\mathbf{b}=\Delta\cdot\mathbf{m}+\mathbf{e}_{\mathsf{CT}}$, the knowledge of $\mathbf{b}$ and decryption result $\mathbf{m}$ will leak the ciphertext noise $\mathbf{e}_{\mathsf{CT}}$, which helps to further reveal the secret key. To avoid the leakage, existing works [3, 22] propose *noise smudging* technique to "smudges out" any small noise $\mathbf{e}_{\mathsf{CT}}$ by adding a large noise. Formally, they prove the following lemma.

Lemma 1 (Noise Smudging) *Let B_1 and B_2 be positive integers, and let $e_1\in[-B_1,B_1]$ be a fixed integer. Let $e_2\leftarrow\!\!\$\ [-B_2,B_2]$ be chosen uniformly at random. Then the distribution of e_2 is statistically indistinguishable from that of e_2+e_1 as long as $B_1/B_2\in\mathsf{negl}$.*

Suppose that the error in ciphertext CT is bounded by B_{CT}, i.e., $\|\mathbf{e}_{\mathsf{CT}}\|\le B_{\mathsf{CT}}$. Then for some "sufficiently large" B_{sm} with $B_{\mathsf{CT}}/B_{sm}\in\mathsf{negl}$, we can use this lemma to prove that the value of $\mathbf{b}+\mathbf{e}_{sm}=\Delta\cdot\mathbf{m}+\mathbf{e}_{\mathsf{CT}}+\mathbf{e}_{sm}$ can be simulated as $\Delta\cdot\mathbf{m}+\mathbf{e}_{sm}$, where $\mathbf{e}_{sm}\leftarrow\!\!\$\ [-B_{sm},B_{sm}]$. This further indicates

ThFHE Construction via ApproxSS

$\mathtt{SKShare}(\mathsf{sk}_i)$// Executed after $\mathsf{sk}_i \leftarrow \mathtt{BFV.SKGen}()$

1 : $\{\mathsf{skShare}_{j,i}\}_{j\in[N]} \leftarrow \mathtt{ApproxSS.Share}(\mathsf{sk}_i)$

2 : **For** $j = 1,\ldots,N$**, do**

3 : Send $\mathsf{skShare}_{j,i}$ to party j

4 : Receive $\mathsf{skShare}_{i,j}$ from party j

5 : $\mathsf{skShare}_i \leftarrow \sum_{j\in[N]} \mathsf{skShare}_{j,i}$

6 : Output $\mathsf{skShare}_i$

..... Decryption Protocol Π_{Dec} .....

Private Input of Party i**:** global secret key share $\mathsf{skShare}_i$

Public Input: C ciphertexts and parameters Δ, B_{sm}

Aggregator Output: C plaintexts by decrypting ciphertexts

Phase 1 // Executed by Party $i \in \mathcal{T}$

7 : **For the** c**-th ciphertext** $\mathsf{CT}_c = (\mathbf{c}_{c,0}, \mathbf{c}_{c,1})$**, do**

8 : $\mathbf{b}_{c,i} \leftarrow \mathbf{c}_{c,1} \cdot \mathsf{skShare}_i + \mathbf{c}_{c,0}$

9 : Concatenate $\{\mathbf{b}_{1,i},\ldots,\mathbf{b}_{C,i}\}$ as $\mathbf{b}_i$

Phase 2 // Interaction between parties and the aggregator

10 : $\chi \leftarrow \mathsf{Uniform}(\{\mathbf{n} \mid \|\mathbf{n}\| \le B_{sm}\})$

11 : $\mathbf{b}' \leftarrow \mathtt{ApproxSS}.\Pi_{\mathsf{ApproxRec}}(\{\mathbf{b}_i\}_{i\in\mathcal{T}}, \chi)$

12 : Parse $\mathbf{b}'$ as $\{\mathbf{b}'_1,\ldots,\mathbf{b}'_C\}$

Phase 3 // Executed by the aggregator

13 : **For the** c**-th ciphertext** $\mathsf{CT}_c = (\mathbf{c}_{c,0}, \mathbf{c}_{c,1})$**, do**

14 : $\mathbf{m}'_c \leftarrow \mathbf{b}'_c + \mathbf{c}_{c,0}$

15 : Decode $\mathbf{m}_c \in \mathcal{R}_{M,P}$ from $\mathbf{m}'_c \in \mathcal{R}_{M,Q}$

16 : Output $\mathbf{m}_1,\ldots,\mathbf{m}_C$

Figure 5: Illustration of Our ThFHE construction

that the value of $\mathbf{b}+\mathbf{e}_{sm}$ is safe to be learned by the adversary, as the adversary can also generates the $\Delta\cdot\mathbf{m}+\mathbf{e}_{sm}$ by itself without using any secret information.

Inspired by this technique, we ask parties and the aggregator to conduct the approximate recovery protocol, with its input χ being the uniform distribution over $[-B_{sm}, B_{sm}]$. Based on the approximate security of ApproxSS, we can guarantee that the adversary cannot learn anything about $\mathbf{b}$ via the execution of approximate recovery protocol, except a random value $\mathbf{b}+\mathbf{e}_{sm}$ that is already safe to be disclosed according to the noise smudging technique. Therefore, we can further prove the security of our ThFHE construction.

Notably, researchers [9, 21, 33] further show that the noise smudging technique still works well with the B_{sm} in the polynomial order rather than $B_{\mathsf{CT}}/B_{sm} \in \mathsf{negl}$. We note that our construction on ThFHE as well as the ApproxSS scheme is compatible with this new conclusion. In fact, we just need to switch the parameter B_{sm} from exponential order to polynomial order, while the other parts can remain unchangeable.

Phase 3: Output of Final Plaintext. At the end of Phase 2, the aggregator learn an approximate message $\mathbf{b}'$ that is close to the original message $\mathbf{b}$. Suppose that the ApproxSS scheme satisfies the $\mathcal{M}_B$-approximate correctness for some $\mathcal{M}_B = [-B, B]$, then the norm of total error in $\mathbf{b}'$ is bounded by $B_{\mathsf{CT}} + B$. As long as $B < \Delta/2 - B_{\mathsf{CT}}$, the aggregator can successfully decode plaintext $\mathbf{m}$ from $\mathbf{b}'$.

The following theorems state the correctness and security of our ThFHE construction. Although the description of Phases 1 – 3 has provided some reasoning, we formally prove them in Appendix A and Appendix B, respectively.

Theorem 1 (Correctness of ThFHE) *Given a T-out-of-N ApproxSS scheme with linearity and $\mathcal{M}_B$-approximate correctness, the ThFHE construction in Figure 5 is a correct T-out-of-N scheme with guaranteed output delivery if $B < \Delta/2 - B_{\mathsf{CT}}$.*

Theorem 2 (Security of ThFHE) *Given a T-out-of-N ApproxSS scheme with vanilla security and χ-approximate security, the ThFHE construction in Figure 5 satisfies simulation-based security if $B_{\mathsf{CT}}/B_{sm} \in \mathsf{negl}$.*

In addition, we note that the decryption protocol consists of the approximate recovery protocol (in Phase 2) and simple operations (in Phases 1 and 3) with complexity $O(K)$, which is linear with respect to the length of ciphertext K but keeps constant with respect to the number of parties N. Hence, the efficiency of decryption protocol is dominated by the efficiency of approximate recovery protocol, unless the approximate recovery protocol has less complexity than $O(K)$, which is unlikely to happen. This inspires us to design efficient ApproxSS, especially under arbitrary threshold.

6 ATASSES: Construction and Analysis

This section starts with the challenges of ApproxSS constructions and the shortcomings of existing solutions in Section 6.1 and then presents how we address them in Section 6.2.

6.1 Challenges and Existing Solutions

Existing works design several arbitrary-threshold (ATh)-ApproxSS schemes based on two types of linear SS. One is linear SS with small recovery coefficients, and the other is Shamir SS. Below we describe the challenges of ATh-ApproxSS design under these two types of SS and highlight the shortcomings of existing solutions.

ApproxSS with Small Recovery Coefficients. Recall that for T-out-of-N linear SS, each share consists of L pieces and the message m can be recovered as a linear combination $m = \sum_{i\in\mathcal{T},l\in[L]} w_{i,l}^{(\mathcal{T})} \cdot s_{i,l}$, where $\mathcal{T}$ is a set of at least T parties, $s_{i,l}$ is

the l-th piece of the share of party i, and $w_{i,l}^{(\mathcal{T})}$ is the recovery coefficients. An existing ApproxSS adopts linear SS with $w_{i,l}^{(\mathcal{T})}$ being a small value like $0, 1$, a.k.a, $\{0,1\}$-linear SS [7].

This type of SS can be easily boosted to ApproxSS and is thus friendly to ThFHE design. To do so, the approximate recovery protocol only needs one round. In this round, every party i adds each of its piece $s_{i,l}$ ($l = 1, \ldots, L$) a noise $n_{i,l} \leftarrow\$ \chi_B$. After collecting these noisy shares from any T parties, the aggregator can recover the approximate message as follows

$$m' = \sum_{i\in\mathcal{T}, l\in[L]} w_{i,l}^{(\mathcal{T})} \cdot (s_{i,l} + n_{i,l}) = m + \sum_{i\in\mathcal{T}, l\in[L]} w_{i,l}^{(\mathcal{T})} \cdot n_{i,l}. \quad (1)$$

Since $w_{i,l}^{(\mathcal{T})}$ and $n_{i,l}$ both have bounded values, the difference between original message and approximate message also has a bounded value. Readers can easily check its approximate correctness and prove its approximate security.

However, this type of SS suffers from an extremely large size of each share in the arbitrary-threshold case. For example, in a T-out-of-N replicated SS, each share consists of $\binom{N-1}{T-1}$ pieces, which may increase exponentially with the number of parties N. Hence, this type of ApproxSS and its induced ThFHE can only be used in some special cases with $\binom{N-1}{T-1}$ being a small value, such as in the full-threshold case ($N = T$) or in a small system (e.g., $N = 2$ or $N = 3$) [23].

Shamir ApproxSS. Shamir SS enjoys an advantage that its share has the same size as the message. Suppose that each party i owns a share s_i, then the message m is recovered as $m = \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot s_i$, where $\mathcal{T}$ can be any set of at least T parties and $L_i^{(\mathcal{T})}$s are the Lagrange coefficients.

Although Shamir SS has the efficiency advantage, the Lagrange coefficients have two properties that make the design of Shamir ApproxSS very challenging. One is unbounded value, i.e., the Lagrange coefficients can be arbitrarily large in the message space. Another is unpredictablity. Lagrange coefficients are associated with the set $\mathcal{T}$ and their values can not be learned until the set $\mathcal{T}$ is determined. These two features imply that Lagrange coefficients can be arbitrarily-large unknown numbers. If each party i adds a noise n_i to its share s_i for approximate security, then the recovered approximate message will be

$$m' = \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot (s_i + n_i) = m + \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot n_i. \quad (2)$$

The difference between original message and approximate message is $\sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot n_i$, which may be blown up by $L_i^{(\mathcal{T})}$s and fail the approximate correctness. To overcome the "blown-up noise" challenge, existing works propose three ideas.

- *Type-I.* One idea is to adopt the "clearing out the denominators" technique [2] to scale n_i so that $L_i^{(\mathcal{T})} \cdot n_i$ has a value bounded by $(N!)^3$ [7]. However, this bound is extremely loose and grows rapidly with the number of parties N. Recall that the ApproxSS requires the bound of $\sum_{i\in\mathcal{T}} L_i \cdot n_i$ to be smaller than the modulus of message space ($\mathcal{M}_B \subsetneq \mathcal{M}$). Hence, this idea results in a large message space of ApproxSS with modulus being $O(N \cdot (N!)^3)$. This huge message space incurs high complexity and difficulty for parameter instantiation.

- *Type-II.* The second idea is to coordinate the noise n_is from all parties so that n_is are the shares of some bounded value n [42]. In other words, we have $n = \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot n_i$ for any set $\mathcal{T}$. By this way, the difference between original and approximate messages is in fact n, whose value is bounded. However, the noise coordination asks each party to share a random noise with the same length as message (say K) to all of other parties. This may result in relatively high cost, especially when the values of N and K are both large.

- *Type-III.* Another idea is to assume that the set of participants $\mathcal{T}$ can be known in advance [35, 36]. In this way, the set $\mathcal{T}$ as well as its associated Lagrange coefficients, can be learned and be used to generate a T-out-of-T decryption share. Then the design of approximate recovery protocol is reduced to its counterpart in the full-threshold case, which can be fairly easy and efficient. Nonetheless, the set $\mathcal{T}$ of participants can be determined by random factors or even the adversary in the real world. Hence, it is impractical to assume the knowledge of $\mathcal{T}$ in advance.

In a word, existing Shamir-based constructions are either inapplicable in the real world or inefficient with high complexity.

6.2 ATASSES for Efficient AThFHE

Faced with the above deficiencies, we use a novel idea called "encrypted share" and propose ATASSES, an Arbitrary-Threshold ApproxSS scheme based on Encrypted Share idea. Next, we first introduce this idea, then describe a concrete construction, and discuss its application for AThFHE at last.

Main Idea. Our goal is to build a Shamir ApproxSS with lower complexity. Suppose that the shares s_is have been generated from the message m by Shamir secret sharing, i.e., $m = \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot s_i$ for any set $\mathcal{T}$ of at least T parties. The key problem is how to devise the approximate recovery protocol. We note that the shares can not be sent to the aggregator without any protection. Otherwise, the aggregator can exactly recover the message and break the approximate security. Correspondingly, we rely on a novel idea to protect the shares.

Part 1: Share Protection. Different from existing constructions who directly add some noise to the share, we employ BFV secret-key encryption to protect shares. To distinguish from ThFHE's parameters, we use the superscript to denote the parameters of BFV secret-key encryption here. For example, the moduli for the plaintext space and ciphertext space are denoted by P' and Q', respectively, and the degree of BFV's

secret key is M'. In this case, one ciphertext can only encrypt a degree-M' polynomial. Nonetheless, the share s_i can be a long vector with length $K \gg M'$. In this case, party i breaks s_i down into C' sub-vectors with length M' so that $K \leq C'M'$ and encrypts each sub-vector, respectively. Formally, the k-th sub-vector $s_{i,k}$ is encrypted to $\mathsf{CTs}_{i,k}$ as

$$\mathsf{CTs}_{i,k} = (\mathbf{a}_k \cdot \mathsf{ek}_{i,1} + \mathbf{e}_{i,1,k} + \Delta' \cdot s_{i,k}, -\mathbf{a}_k), \tag{3}$$

where $\mathbf{a}_k$ is a uniformly-random polynomial, $\mathsf{ek}_{i,1}$ is the (secret) encryption key of party i, $\mathbf{e}_{i,1,k}$ is an error polynomial with norm bounded by B', and $\Delta' = \lfloor Q'/P' \rfloor$.

Part 2: Message Recovery. Our first observation is that BFV secret-key encryption is linearly-homomorphic in the key. Specifically, given $\mathsf{CTs}_{i,k}$s from multiple parties $i \in \mathcal{T}$ and Lagrange coefficients $\{L_i^{(\mathcal{T})}\}_{i\in\mathcal{T}}$, if $\mathsf{CTs}_{i,k}[1] = -\mathbf{a}_k$ for any i, the aggregator can compute a new ciphertext CTs_k as

$$\mathsf{CTs}_k = (\sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot \mathsf{CTs}_{i,k}[0], -\mathbf{a}_k). \tag{4}$$

Substituting Eq.(3) into Eq.(4) leads to the following:

$$\begin{aligned}\mathsf{CTs}_k[0] = \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot \mathsf{CTs}_{i,k}[0] = \mathbf{a}_k \cdot \sum_{i\in\mathcal{T}} (L_i^{(\mathcal{T})} \cdot \mathsf{ek}_{i,1}) \\ + \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot \mathbf{e}_{i,1,k} + \Delta' \cdot (\sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot s_{i,k}).\end{aligned} \tag{5}$$

By observing Eq.(5), we can tell that CTs_k is the encryption of $\sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot s_{i,k} = m_k$, with the key being $L_i^{(\mathcal{T})} \cdot \mathsf{ek}_{i,1}$. In other words, the aggregator can decrypt the original message m_k from CTs_k if it learns decryption key $\mathsf{dk} = \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot \mathsf{ek}_{i,1}$. Nonetheless, ApproxSS requires to recover an approximate message rather than the original message. To solve this problem, we simply ask each party i to encrypt a B_{sm}-bounded noise $n_{i,k}$ using another key $\mathsf{ek}_{i,2}$ but the same $\mathbf{a}_k$, i.e.,

$$\mathsf{CTn}_{i,k} = (\mathbf{a}_k \cdot \mathsf{ek}_{i,2} + \mathbf{e}_{i,2,k} + \Delta' \cdot n_{i,k}, -\mathbf{a}_k), \tag{6}$$

By using the linearly key-homomorphic property again, the aggregator combines $\mathsf{CTn}_{i,k}$s and $\mathsf{CTs}_{i,k}$s into an overall ciphertext CTall_k as following.

$$\mathsf{CTall}_k = (\sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot \mathsf{CTs}_{i,k}[0] + \mathsf{CTn}_{i,k}[0], -\mathbf{a}_k). \tag{7}$$

With the similar reasoning process, we can come to the conclusion: CTall_k is the encryption of $\sum_{i\in\mathcal{T}} (L_i^{(\mathcal{T})} \cdot s_{i,k} + n_{i,k})$, with decryption key being $\mathsf{dk} = \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot \mathsf{ek}_{i,1} + \mathsf{ek}_{i,2}$. Since $\sum_{i\in\mathcal{T}} (L_i^{(\mathcal{T})} \cdot s_{i,k} + n_{i,k}) = m_k + \sum_{i\in\mathcal{T}} n_{i,k}$, the decryption of CTall_k with dk is exactly the desired approximate message. Recall that the value of $n_{i,k}$ is bounded by B_{sm}. The difference between original and approximate messages is bounded by $N \cdot B_{sm}$, which satisfies approximate correctness.

Before presenting how the aggregator learns dk, we note that the decryption of overall ciphertext CTall_k may fail if its error exceeds $\Delta'/2$. Nonetheless, we only need to select a slightly larger ciphertext space to fix this problem. Particularly, the error in CTall_k is $\sum_{i\in\mathcal{T}} (L_i^{(\mathcal{T})} \mathbf{e}_{i,1,k} + \mathbf{e}_{i,2,k})$. Since $L_i^{(\mathcal{T})}$s are in the message space of Shamir SS as s_i, which is also the plaintext space of BFV secret-key encryption, their values are bounded by $P' - 1$. Recall that the values of original errors $\mathbf{e}_{i,1,k}$ and $\mathbf{e}_{i,2,k}$ are bounded by B'. Then the overall error is bounded by $N \cdot P' \cdot B'$. Hence, although Lagrange coefficients $L_i^{(\mathcal{T})}$s also blow the error up, if we set $N \cdot P' \cdot B' < \Delta'/2$ or more strictly $Q' > 2(P')^2 B' N + 2P' = O(N)$, the decryption will succeed to output correct approximate message. In other words, our scheme only requires $Q' = O(N)$ and has no requirement on P'. In contrast, Type-I Shamir ApproxSS [7] requires the message space with $P' = O(N \cdot (N!)^3)$ that is much larger than the ciphertext space of our scheme. This is one reason why our "encrypted share" idea performs better.

Part 3: Secure Computation of Decryption Key. Now, the only step left is to obtain the decryption key $\mathsf{dk} = \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot \mathsf{ek}_{i,1} + \mathsf{ek}_{i,2}$. We first note that dk is safe to disclose: following the security proof in [5], the ciphertexts are still indistinguishable from random values even when dk is revealed. Readers can find more details in Appendix C. To see how to compute the decryption key, we recall that $L_i^{(\mathcal{T})}$s are associated with the set $\mathcal{T}$ and can NOT be revealed before the set $\mathcal{T}$ is known. Since $L_i^{(\mathcal{T})}$s are necessary for computing dk, the aggregator needs an extra round after the set $\mathcal{T}$ is known. Nonetheless, the party $i \in \mathcal{T}$ who encrypts s_i and n_i may not participate in this extra round. If so, no one knows its encryption keys $\mathsf{ek}_{i,1}, \mathsf{ek}_{i,2}$ that are also necessary for the dk computation.

Our solution to this problem consists of two rounds. In the first round, each participant $i \in \mathcal{T}$ not only encrypts s_i and n_i, but also shares its encryption keys with other parties using T-out-of-N Shamir SS. At the end of the first round, the knowledge of the set $\mathcal{T}$ becomes available, as well as its associated $L_i^{(\mathcal{T})}$s. With the knowledge of $L_i^{(\mathcal{T})}$s, in the second round, each party j computes one decryption key share from its received shares of $\mathsf{ek}_{i,1}$ and $\mathsf{ek}_{i,2}$ by the linear property of Shamir SS. Suppose that the set of participants in the second round is $\mathcal{T}_2$. After collecting the shares of dk from $\mathcal{T}_2$, the aggregator recovers dk and further obtains the approximate message by decrypting CTall_k. Here, the set of participants $\mathcal{T}$ and $\mathcal{T}_2$ can be different. Therefore, our scheme does not require to learn the set of participants in advance, which is a key difference from the Type-III Shamir ApproxSS [35].

We note that our solution also asks each party i to share its encryption keys with other parties, which leads to quadratic computation complexity $O(N^2)$ with respect to the number of parties N. Nonetheless, the length of encryption key is the intrinsic parameter of BFV, which remains constant with respect to the length of message. Therefore, in our solution, the complexity of sharing does not grow with the message's length K. For comparison, the complexity of Type-II Shamir ApproxSS asks to share a noise whose length is the same

ATASSES Construction: $\Pi_{\mathsf{ApproxRec}}$

Private Input of Party i: a share s_i of message m

Public Input: $\chi = \mathsf{Uniform}(\{n \mid \|n\| \le B_{sm}\})$

Aggregator Output: an approximate message m'

$\mathtt{PartyR1}(s_i)$

1 : $\mathsf{ek}_{i,1} \leftarrow \mathtt{BFV.SKGen}(),\ \mathsf{ek}_{i,2} \leftarrow \mathtt{BFV.SKGen}()$

2 : $\{\mathsf{ekShare}_{j,i,1}\}_{j\in[N]} \leftarrow \mathtt{ShamirSS.Share}(\mathsf{ek}_{i,1})$

3 : $\{\mathsf{ekShare}_{j,i,2}\}_{j\in[N]} \leftarrow \mathtt{ShamirSS.Share}(\mathsf{ek}_{i,2})$

4 : Sends $\mathsf{ekShare}_{j,i,1}, \mathsf{ekShare}_{j,i,2}$ to party j

5 : Breaks s_i down into length-M' sub-vectors

6 : **For the k-th length-M' sub-vector $s_{i,k}$, do**

7 : $\quad \mathbf{a}_k \leftarrow\!\$\ \mathsf{CRS}, \quad n_{i,k} \leftarrow\!\$\ \chi$

8 : $\quad \mathsf{CTs}_{i,k} \leftarrow \mathtt{BFV.SKEnc}(\mathsf{ek}_{i,1}, s_{i,k}, \mathbf{a}_k)$

9 : $\quad \mathsf{CTn}_{i,k} \leftarrow \mathtt{BFV.SKEnc}(\mathsf{ek}_{i,2}, n_{i,k}, \mathbf{a}_k)$

10 : $\quad$ Sends $\mathsf{CTs}_{i,k}, \mathsf{CTn}_{i,k}$ to the aggregator

$\mathtt{AggregatorR1}()$

// Executed after collecting $\mathsf{CTs}_{i,k}, \mathsf{CTn}_{i,k}$ for all ks from set $\mathcal{T}$

11 : Computes $\{L_i^{(\mathcal{T})}\}_{i\in\mathcal{T}}$ and sends $\{L_i^{(\mathcal{T})}\}_{i\in\mathcal{T}}$ to all parties

$\mathtt{PartyR2}(\{L_i^{(\mathcal{T})}\}_{i\in\mathcal{T}}, \{\mathsf{ekShare}_{i,j,1}, \mathsf{ekShare}_{i,j,2}\}_{j\in[N]})$

12 : $\mathsf{dkShare}_j = \sum_{i\in\mathcal{T}} (L_i^{(\mathcal{T})} \cdot \mathsf{ekShare}_{i,j,1} + \mathsf{ekShare}_{i,j,2})$

13 : Sends $\mathsf{dkShare}_j$ to the aggregator

$\mathtt{AggregatorR2}(\{\mathsf{CTs}_{i,k}, \mathsf{CTn}_{i,k}\}_{i\in\mathcal{T}}, \{\mathsf{dkShare}_j\}_{j\in\mathcal{T}_2})$

// Executed after collecting $\mathsf{dkShare}_j$ from set $\mathcal{T}_2$

14 : $\mathsf{dk} \leftarrow \mathtt{ShamirSS.Rec}(\{\mathsf{dkShare}_j\}_{j\in\mathcal{T}_2})$

15 : **For the k-th ciphertext** $(\mathsf{CTs}_{i,k}, \mathsf{CTn}_{i,k})$**, do**

16 : $\quad \mathsf{CTall}_k \leftarrow (\sum_{i\in\mathcal{T}} (L_i^{(\mathcal{T})} \cdot \mathsf{CTs}_{i,k}[0] + \mathsf{CTn}_{i,k}[0]), -\mathbf{a}_k)$

17 : $\quad m'_k \leftarrow \mathtt{BFV.Dec}(\mathsf{dk}, \mathsf{CTall}_k)$

18 : Concatenate all m'_ks as m'

Figure 6: The approximate recovery protocol of ATASSES.

as that of message. As a result, its complexity of sharing is $O(N^2K)$ and grows with message's length. This is another reason why our solution performs better.

Concrete Construction. As an implementation of the above idea, the $\Pi_{\mathsf{ApproxRec}}$ protocol of ATASSES consists of two rounds, as illustrated in Figure 6.

- *Round 1: Encryption of Share and Noise.* Each party i executes algorithm $\mathtt{PartyR1}$ for two tasks. One is to generate and share two encryption keys with other parties (Line 1-4). The other is to encrypt its share s_i and noise n_i using BFV secret-key encryption and sends the ciphertexts to the aggregator (Line 5-10). After collecting at least T groups of ciphertexts of s_i and n_i from set $\mathcal{T}$, the aggregator executes $\mathtt{AggregatorR1}$ to compute the Lagrange coefficients associated with set $\mathcal{T}$ and send them to all parties. We note that the successful decryption of ATASSES relies on a condition, namely, the set of parties who send the ciphertexts should be identical to the set of parties who send the shares of encryption keys, which is both denoted by symbol $\mathcal{T}$. Our system model in Section 2.2 assumes that this condition is met for at least T parties. In practice, we can take the aggregator as the communication relay between parties, so that the aggregator can monitor the party-to-party message delivery and guarantee to satisfy this condition.
- *Round 2: Decryption of Approximate Message.* Each party j executes $\mathtt{PartyR2}$ to compute its decryption key share $\mathsf{dkShare}_j$. After collecting at least T shares from set $\mathcal{T}_2$, the aggregator executes $\mathtt{AggregatorR2}$ to recover the decryption key dk from $\{\mathsf{dkShare}_j\}_{j\in\mathcal{T}_2}$ (Line 14) and further decrypt every overall ciphertexts CTall_ks (Line 15-17). The concatenation of decryption results is the desired output m'.

Application to ThFHE. To construct ThFHE based on ATASSES, the ciphertext space of ThFHE should be set as the message space of ATASSES, so that b in ThFHE can be regarded as m in ATASSES. In addition, as we will show later, the ATASSES can achieve $\mathcal{M}_B$-approximate correctness with $B = T \cdot B_{sm}$. Recall that the correctness of ThFHE requires $B < \Delta/2 - B_{\mathsf{CT}}$. Hence, when applying ATASSES for ThFHE construction, we need to set $T \cdot B_{sm} < \Delta/2 - B_{\mathsf{CT}}$.

6.3 Performance Analysis of ATASSES

We analyze ATASSES in terms of its correctness, security, and efficiency. The approximate correctness and security are given by Theorem 3. The complexity of $\Pi_{\mathsf{ApproxRec}}$ protocol of ATASSES along with other ApproxSS schemes is listed in Table 1. Due to the page limit, the proof and analysis details is deferred to Appendix C and Appendix D, respectively. We note that Type-III Shamir ApproxSS has a different assumption with other schemes, i.e., the knowledge of participant set is known in advance. Compared with existing schemes that do not rely on this assumption, ATASSES reduces the computation (resp. communication) complexity from $O(N^2 \cdot K)$ to $O(N^2 + NK)$ (resp. $O(N \cdot K)$ to $O(N + K)$). Although Type-III scheme shows lower complexity, it may lose superiority when the knowledge of participant set is fault or even unavailable. In this case, Type-III scheme can only randomly search from $\binom{N}{T}$ possibilities until finding the correct set $\mathcal{T}$, which may yield a significantly higher (average) cost than ATASSES.

Theorem 3 (ATASSES's Properties) *ATASSES satisfies χ-approximate correctness and $\mathcal{M}_B$-approximate security under RLWE-hardness assumption for $\chi = \mathsf{Uniform}(\{n \mid \|n\| \le B_{sm}\})$ and $\mathcal{M}_B = \{n \mid \|n\| \le T \cdot B_{sm}\}$.*

Table 1: Comparison between ATASSES and existing ApproxSS schemes in terms of $\Pi_{\mathsf{ApproxRec}}$ protocol's communication complexity (Comm.), computation complexity (Comp.), and round number.

	$\{0,1\}$-ApproxSS	Shamir-based ApproxSS			
		Type-I [7]	Type-II [42]	Type-III [35, 36]	**ATASSES**
Comm.	$O(N^{4.2}K)$	$O(NK)$	$O(NK)$	$O(K)$	$O(N+K)$
Comp.	$O(N^{5.2}K)$	$O(N^2K)$	$O(N^2K)$	$O(NK)$	$O(N^2+NK)$
Round Number	1	1	2	1	2

7 Experiment Evaluation

We empirically evaluate ATASSES's efficiency with code in https://zenodo.org/records/14644655.

Setup. We implement the proposed ATASSES as well as three existing ApproxSS schemes for comparison, including two one-round schemes (i.e., replicated ApproxSS as a specific instance of $\{0,1\}$-ApproxSS and Type-I Shamir ApproxSS) and a two-round scheme (i.e., Type-II Shamir ApproxSS). Particularly, ATASSES relies on the BFV secret-key encryption and is hence implemented on top of Lattigo library [1]. We note that Type-III Shamir ApproxSS has a different setting with others, i.e., it allows the knowledge of participant set in advance. With this setting, Type-III Shamir ApproxSS has unfair advantage and is thus excluded from comparison. All experiments in this part are performed in a 14-inch MacBook with Apple M2 Pro CPU.

When implementing these ApproxSS schemes, we adopt the common parameter setting to make a fair comparison. Since ATASSES adopts the default parameter setting `PN12QP109` provided by Lattigo library, we set the message space of almost all ApproxSS schemes to be the polynomial ring with degree 4096 and modulus 65537. The only exception is Type-I Shamir ApproxSS. Since Type-I Shamir ApproxSS requires a much larger message space, we use Golang's "math/big" package to search the smallest modulus that can meet its requirement. As for the norm bound of noises B_{sm}, we set it to be 2^{16} for all schemes.[2]

In addition, we adopt multiple values of T, N, and K to observe their impacts on performance. Specifically, we set $T \in \{0.5N, 0.7N, 0.9N\}$. The value of N ranges from 10 to 100 in existing one-round schemes, while ranges from 100 to 1000 in existing two-round schemes. The reason behind such a difference is that those one-round schemes are not efficient enough to support a large number of parties. As for the value of K, we set it to be the multiples of key's size. Let M' denote the size of key used in ATASSES's encryption scheme, we consider $K \in \{5M', 10M', 15M', 20M'\}$.

Results. We measure the running time of approximate recovery protocols as the performance metric. The running time includes communication time and computation time. The communication time is set to be the size of transferred data divided by the network bandwidth Band. We set Band $=$ 98Mbps, which emulates the realistic bandwidth of 4G cellular networks [34, 40]. The experiment results are shown in Figure 7, from which we can come to the following conclusions.

Regarding the growth of running time with N, we can observe a slow growth trend of ATASSES. This aligns with our theoretical analysis and demonstrates that ATASSES can be applied to large-scale systems. In contrast, existing ApproxSS schemes have much longer time that grows rapidly with N. Particularly, those one-round ApproxSS schemes, despite their lower round-complexity, suffers from extremely-large running time when N exceeds 30. The two-round ApproxSS scheme performs better, but still worse than ATASSES. More specifically, ATASSES has comparable running time as Type-II Shamir ApproxSS when there are a small number of parties, while provides a speedup of $3.83\times - 15.4\times$ when $N = 1000$. Meanwhile, the above observations on comparison between different ApproxSS schemes hold for different values of T.

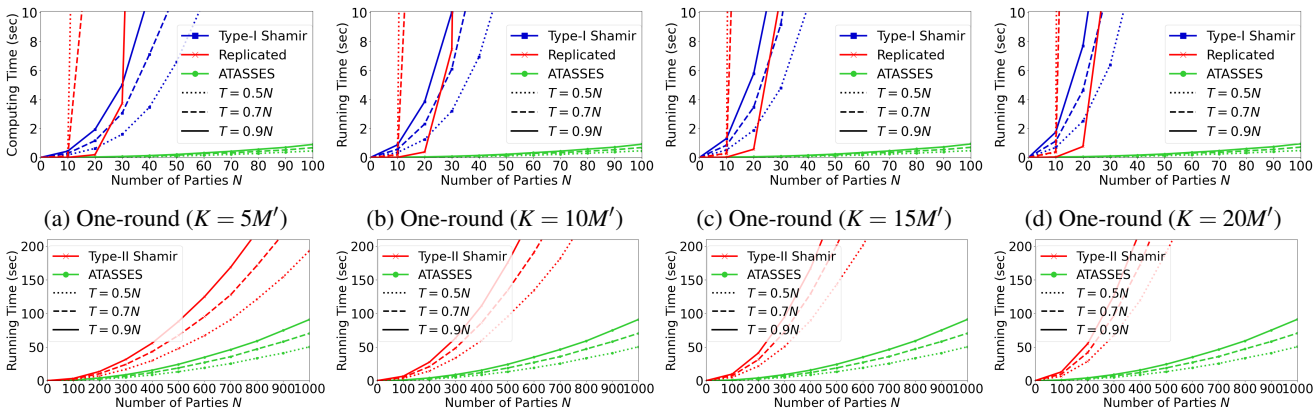
In addition, by observing the same row of sub-figures, the running time of ATASSES hardly changes with the growth of K. The reason is that the running time is dominated by the operations whose running time is independent with K. In contrast, the running time of existing ApproxSS schemes grows rapidly with different values of K. Hence, ATASSES's performance advantage becomes more significant with a larger value of K. This demonstrates that ATASSES has the advantage to be applied when the data size is large. Meanwhile, we also observe that ATASSES may lack superiority when K is smaller than the length of BFV's secret key M'. Nonetheless, this can be mitigated by choosing an adequate value of M'.

8 Related Work

In the interest of space, this section reviews related work solely on *arbitrary-threshold* FHE schemes that try to reduce the complexity. Particularly, we divide these schemes into the synchronous and asynchronous settings.

Most of existing schemes are designed in the asynchronous setting, as it can cover more real-world situations. In this setting, the set of participants can vary at any time, and a participant can NOT learn the set of other participants until receiving some message from them. Asharov et al. proposed a secret key resharing idea to extend a full-threshold ThHE to arbitrary-threshold ThHE [3]. Nonetheless, their idea would reveal the secret key of non-participants, resulting in a security vulnerability. Boneh et al. proposed two arbitrary-threshold ThHE schemes based on $\{0,1\}$-linear SS and Shamir SS, respectively [7]. Although these two schemes avoid the secret key leakage and only require one-round communication for decryption, they both suffer from significant overhead that grows rapidly with the number of parties N. Concurrently

[2] We note that in practice, this value should be set according to the requirements of applications. For example, the ThFHE may require the value of B_{sm} to be in the polynomial order or even super-polynomial order with respect to some security parameter. In the experiments, we fix the value of B_{sm} for all schemes to make a fair comparison.



(a) One-round ($K = 5M'$) (b) One-round ($K = 10M'$) (c) One-round ($K = 15M'$) (d) One-round ($K = 20M'$)

(e) Two-rounds ($K = 5M'$) (f) Two-rounds ($K = 10M'$) (g) Two-rounds ($K = 15M'$) (h) Two-rounds ($K = 20M'$)

Figure 7: Running time of ATASSES and existing (one-round and two-rounds) ApproxSS with different values of K, N, and T.

to these one-round schemes, Tian et al. design a two-round scheme based on Shamir secret sharing and apply them in the scenario of federated learning [42], which also occurs in [23]. This scheme asks each party to share a secret noise with every parties, resulting in relatively high complexity.

Different from the above work, Mouchet et al. note that the asynchronous setting might be an overkill for many applications and consider to design an efficient scheme in the synchronous setting [35]. In this setting, the participant set can be known to every parties as a prior knowledge. By utilizing this knowledge, they devise a decryption protocol with $O(NK)$ computation complexity and $O(K)$ communication complexity. Considering that parties in the expected participant set may also crash, Helium framework [36] extends [35] with a concrete *retry* mechanism. Specifically, if the expected participant set does not fully match the physical truth, Helium allows the parties to re-execute the decryption protocol with a modified set of expected participants. Once the modified set matches the physical truth of participant set, the decryption protocol can output the correct result. Nonetheless, the successful decryption still requires that the T participants be exactly as expected. Although this condition can be met when those non-participants become unavailable following *random process* as modeled in Helium's churn model, these works may fail to handle all types of non-participants. For example, a denial-of-service adversary may deliberately switch a party to non-participant once this party is identified as an expected participant. In this case, the decryption protocol cannot output the correct result even after multiple retries, because the set of expected participants will never match the physical truth.

In summary, existing schemes in the asynchronous setting suffer from low efficiency and existing schemes in the synchronous setting may fail to handle all types of non-participants. In this work, we aim to propose an AThFHE scheme with lower complexity that can successfully decrypt as long as there are at least T participants (could be *any* T parties) at the same time. Meanwhile, we note that some works also try to optimize AThFHE along two other lines. Along one line, recent work [13] considers *maliciously-secure* ThFHE design. Still, this work assume that there are no non-participants. Our work can help to remove this assumption and advance this work to fully-malicious security. Along the other line, a series of works [9, 21, 33] improve the efficiency by reducing the value of noises. Particularly, they show that the noise smudging technique can also be applied when the noise is in the polynomial order rather than exponential order. Since the ciphertext modulus needs to grow with a larger value of noises, a lower-order noise can reduce the ciphertext modulus from exponential order to polynomial order, which helps to decrease the communication and computation complexity. Notably, our work is compatible with noises of any order and thus can employ these works to reduce the ciphertext modulus.

9 Conclusion

We formulate a novel primitive called approximate secret sharing (ApproxSS), and establish the reduction of constructing ThFHE schemes to ApproxSS designs. We develop ATASSES, a Shamir-based ApproxSS scheme in the arbitrary-threshold case with lower complexity. We theoretically prove its security and correctness with guaranteed output delivery, and then empirically demonstrate its substantial efficiency advantages over baselines. ATASSES helps to induce an efficient arbitrary-threshold ThFHE scheme that outperforms existing schemes for higher efficiency, by which ThFHE can be truly applied in the real world, such as the secure aggregation in federated learning. Overall, we believe that the proposed primitive can not only improve the efficiency of ThFHE, but also be of independent interest with more applications.

10 Ethics Considerations

We have carefully considered the ethics following the conference guideline. Our research aims to reduce the complexity of cryptographic primitives by proposing novel techniques. The involved stakeholders could include the designers of secure multi-party protocols and the participants of computing services. Our work will have positive effects on 1) facilitating the designers with more tools and 2) protecting the privacy of participants. Meanwhile, both of the research process and our contributions have no negative impacts, including but not limited to breaking the security of computer systems, collecting private information, and violating human rights.

11 Open Science Policy

Regarding the open science policy, we fully obey it and have published our code via anonymous link (see Section 7).

Acknowledgments

We sincerely thank our shepherd and all anonymous reviewers, for their constructive comments and valuable suggestions on improving the quality of our manuscript. This work is in part supported by the Fundamental Research Funds for the Central Universities (Grant No. 2242024k30059).

References

[1] Lattigo v4. Online: https://github.com/tuneinsight/lattigo, August 2022. EPFL-LDS, Tune Insight SA.

[2] Shweta Agrawal, Xavier Boyen, Vinod Vaikuntanathan, Panagiotis Voulgaris, and Hoeteck Wee. Functional encryption for threshold functions (or fuzzy IBE) from lattices. In *Proceedings of Springer PKC*, pages 280–297, 2012.

[3] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Proceedings of Springer EUROCRYPT*, pages 483–501, 2012.

[4] Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915.

[5] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. ACORN: Input validation for secure aggregation. In *Proceedings of USENIX Security Symposium*, pages 4805–4822, 2023.

[6] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. In *Proceedings of Springer PKC*, pages 528–558, 2021.

[7] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *Proceedings of Springer CRYPTO*, pages 565–596, 2018.

[8] Cecilia Boschini, Jan Camenisch, Max Ovsiankin, and Nicholas Spooner. Efficient post-quantum SNARKs for RSIS and RLWE and their applications to privacy. In *Springer Post-Quantum Cryptography*, pages 247–267, 2020.

[9] Katharina Boudgoust and Peter Scholl. Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. In *Proceedings of Springer ASIACRYPT*, pages 371–404, 2023.

[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of ACM ITCS*, page 309–325, 2012.

[11] Ignacio Cascudo and Bernardo David. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In *Proceedings of Springer EUROCRYPT*, pages 216–248, 2024.

[12] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. Verifiable encodings for maliciously-secure homomorphic encryption evaluation. In *Proceedings of ACM CCS*, page 3525–3527, 2023.

[13] Sylvain Chatel, Christian Mouchet, Ali Utkan Sahin, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. PELTA - Shielding multiparty FHE against malicious adversaries. In *Proceedings of ACM CCS*, page 711–725, 2023.

[14] Jeffrey Chen, Manaswitha Edupalli, Bonnie Berger, and Hyunghoon Cho. Secure and federated linear mixed model association tests. *bioRxiv*, 2022.

[15] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *Proceedings of Springer SAC*, pages 347–368, 2018.

[16] Jung Hee Cheon, Seungwan Hong, and Duhyeong Kim. Remark on the security of CKKS scheme in practice. *Cryptology ePrint Archive*, 2020. https://eprint.iacr.org/2020/1581.

[17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Proceedings of Springer ASIACRYPT*, pages 409–437, 2017.

[18] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[19] Hyunghoon Cho, David Froelicher, Jeffrey Chen, Manaswitha Edupalli, Apostolos Pyrgelis, Juan R. Troncoso-Pastoriza, Jean-Pierre Hubaux, and Bonnie Berger. Secure and federated genome-wide association studies for biobank-scale datasets. *bioRxiv*, 2022.

[20] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of IEEE FOCS*, pages 383–395, 1985.

[21] Siddhartha Chowdhury, Sayani Sinha, Animesh Singh, Shubham Mishra, Chandan Chaudhary, Sikhar Patranabis, Pratyay Mukherjee, Ayantika Chatterjee, and Debdeep Mukhopadhyay. Efficient threshold FHE with application to real-time systems. Cryptology ePrint Archive, Paper 2022/1625, 2022. https://eprint.iacr.org/2022/1625.

[22] Ronald Cramer, Ivan Damgård, and Jesper B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Proceedings of Springer EUROCRYPT*, pages 280–300, 2001.

[23] Morten Dahl, Daniel Demmler, Sarah El Kazdadi, Arthur Meyre, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P. Smart, Samuel Tap, and Michael Walter. Noah's Ark: Efficient threshold-FHE using noise flooding. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, page 35–46. ACM, 2023.

[24] Ivan Damgård and Rune Thorbek. Linear integer secret sharing and distributed exponentiation. In *Proceedings of Springer PKC*, pages 75–90, 2006.

[25] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Proceedings of Springer EUROCRYPTO*, volume 617–640, 2015.

[26] Muhammed F. Esgin, Ngoc Khanh Nguyen, and Gregor Seiler. Practical exact proofs from lattices: New techniques to exploit fully-splitting rings. In *Proceedings of Springer ASIACRYPT*, page 259–288, 2020.

[27] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, 2012. https://ia.cr/2012/144.

[28] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, T. Schneider, Hossein Yalame, and Shaza Zeitouni. SAFELearn: Secure aggregation for private federated learning. *IEEE Security and Privacy Workshops (SPW)*, pages 56–62, 2021.

[29] David Froelicher, Juan R Troncoso-Pastoriza, Apostolos Pyrgelis, Sinem Sav, Joao Sa Sousa, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Scalable privacy-preserving distributed learning. In *Proceedings of Privacy Enhancing Technologies*, page 323–347, 2021.

[30] David Froelicher, Juan R. Troncoso-Pastoriza, Jean Louis Raisaro, Michel A. Cuendet, Joao Sa Sousa, Hyunghoon Cho, Bonnie Berger, Jacques Fellay, and Jean-Pierre Hubaux. Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption. *Nature Communications*, 12(1):5910, October 2021.

[31] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *Proceedings of Springer EUROCRYPT*, pages 458–487, 2022.

[32] Xiaohan Hao, Chao Lin, Wenhan Dong, Xinyi Huang, and Hui Xiong. Robust and secure federated learning against hybrid attacks: A generic architecture. *IEEE Transactions on Information Forensics and Security*, 19:1576–1588, 2024.

[33] Daniele Micciancio and Adam Suhl. Simulation-secure threshold PKE from LWE with polynomial modulus. *Cryptology ePrint Archive*, 2023. https://eprint.iacr.org/2023/1728.

[34] Dimitar Minovski, Niclas Ögren, Karan Mitra, and Christer Åhlund. Throughput prediction using machine learning in LTE and 5G networks. *IEEE Transactions on Mobile Computing*, 22(3):1825–1840, 2023.

[35] Christian Mouchet, Elliott Bertrand, and Jean-Pierre Hubaux. An efficient threshold access-structure for RLWE-based multiparty homomorphic encryption. *Journal of Cryptology*, 36(2):10, 2023.

[36] Christian Mouchet, Sylvain Chatel, Apostolos Pyrgelis, and Carmela Troncoso. Helium: Scalable MPC among lightweight participants and under churn. In *Proceedings of ACM CCS*, page 3038–3052, 2024.

[37] Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. In *Proceedings on Privacy Enhancing Technologies*, volume 2021, pages 291–311, 2021.

[38] Sinem Sav, Jean-Philippe Bossuat, Juan R. Troncoso-Pastoriza, Manfred Claassen, and Jean-Pierre Hubaux. Privacy-preserving federated neural network learning for disease-associated cell classification. *Patterns*, 3(5):100487, 2022.

[39] Sinem Sav, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. POSEIDON: privacy-preserving federated neural network learning. In *Proceedings of NDSS Symposium*, 2021.

[40] Joel Scheuner and Philipp Leitner. A cloud benchmark suite combining micro and applications benchmarks. In *Proceedings of ACM/SPEC International Conference on Performance Engineering*, pages 161–166, 2018.

[41] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[42] Haibo Tian, Yanchuan Wen, Fangguo Zhang, Yunfeng Shao, and Bingshuai Li. A distributed threshold additive homomorphic encryption for federated learning with dropout resiliency based on lattice. In *Proceedings of International Symposium on Cyberspace Safety and Security*, page 277–292. Springer, 2022.

A Proof of Theorem 1

Theorem 1 (Correctness of ThFHE, Restated) *Given a T-out-of-N ApproxSS scheme with linearity and $\mathcal{M}_B$-approximate correctness, the ThFHE construction in Figure 5 is a correct T-out-of-N scheme with guaranteed output delivery if $B < \Delta/2 - B_{\mathsf{CT}}$.*

Proof: To prove the correctness of ThFHE with guaranteed output delivery, we must show that given a ciphertext $\mathsf{CT} = (\mathbf{c}_0, \mathbf{c}_1)$ as the encryption of $\mathbf{m}$ under the global public key sk, the decryption protocol Π_{Dec} will output $\mathbf{m}$ with overwhelming probability.

In the Phase 1 of decryption protocol, we have $\mathbf{b}_i = \mathbf{c}_1 \cdot \mathsf{skShare}_i + \mathbf{c}_0$. By the linearity of ApproxSS, since $\mathsf{skShare}_i$ is a share of global secret key sk, $\mathbf{b}_i$ is a share of $\mathbf{c}_1 \cdot \mathsf{sk} + \mathbf{c}_0$. According to the definition of BFV encryption, the ciphertext has the structure $\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathsf{sk} = \Delta\mathbf{m} + \mathbf{e}_{\mathsf{CT}}$, with $\|\mathbf{e}_{\mathsf{CT}}\| < B_{\mathsf{CT}}$. Hence, we can learn that $\mathbf{b}_i$ is a share of $\Delta\mathbf{m} + \mathbf{e}_{\mathsf{CT}}$. Let $\mathbf{b}$ denote the $\Delta\mathbf{m} + \mathbf{e}_{\mathsf{CT}}$.

In the Phase 2 of decryption protocol, the approximate recovery protocol is executed to output $\mathbf{b}'$ with only T participants. By the $\mathcal{M}_B$-approximate correctness of ApproxSS, we have $\mathbf{b}' - \mathbf{b} \in \mathcal{M}_B$ and thus $\|\mathbf{b}' - \mathbf{b}\| < \Delta/2 - B_{\mathsf{CT}}$. Recall that $\mathbf{b} = \Delta\mathbf{m} + \mathbf{e}_{\mathsf{CT}}$ from the conclusion of Phase 1. Hence, we have $\mathbf{b}' = \Delta\mathbf{m} + \mathbf{e}_{\mathsf{CT}} + \mathbf{n}$ for some $\|\mathbf{n}\| < \Delta/2 - B_{\mathsf{CT}}$.

In the Phase 3 of decryption protocol, the plaintext $\mathbf{m}$ is decoded from $\mathbf{b}'$. By the decryption requirement of BFV, the decryption will succeed to output $\mathbf{m}$ if and only if $\|\mathbf{e}_{\mathsf{CT}} + \mathbf{n}\| < \Delta/2$. Recall that $\|\mathbf{e}_{\mathsf{CT}}\| < B_{\mathsf{CT}}$ and $\|\mathbf{n}\| < \frac{\Delta}{2} - B_{\mathsf{CT}}$ from the parameter setting and the conclusion of Phase 2. By triangle inequality $\|a+b\| < \|a\| + \|b\|$, we have $\|\mathbf{e}_{\mathsf{CT}} + \mathbf{n}\| < \|\mathbf{e}_{\mathsf{CT}}\| + \|\mathbf{n}\| < B_{\mathsf{CT}} + (\Delta/2 - B_{\mathsf{CT}}) = \Delta/2$. since the condition $\|\mathbf{e}_{\mathsf{CT}} + \mathbf{n}\| < \Delta/2$ holds, the decryption protocol will successfully output $\mathbf{m}$, which concludes this proof. ■

B Proof of Theorem 2

Theorem 2 (Security of ThFHE, Restated) *Given a T-out-of-N ApproxSS scheme with vanilla security and χ-approximate security, the ThFHE construction in Figure 5 satisfies simulation-based security if $B_{\mathsf{CT}}/B_{sm} \in \mathsf{negl}$.*

Proof: The proof is conducted via a sequence of hybrid experiments between an adversary $\mathcal{A}$ and a challenger.

H_0. This is the real-world execution of ThFHE protocols and algorithms. Specifically, the challenger executes the key generation protocol (including the full-threshold FHE's key generation protocol and our `SKShare` protocol). Then the challenger encrypts the data of participated parties and send the ciphertexts to the aggregator. Next, the challenger homomorphically evaluates over the ciphertexts from parties and outputs the ciphertext of computation result. Finally, the challenger runs the decryption protocol for parties and aggregator to output the computation result. During this process, the adversary can see the transcripts of every corrupted parties (and in particular, the aggregator) and assign the set of participants in each round.

H_1. This experiment is the same as H_0, except that the `SKShare` in key generation and approximate recovery protocol in decryption are simulated as specified in $\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}$. Specifically, the challenger samples a random value m_χ from χ and then runs several simulator algorithms to output the transcripts that can be seen by the adversary. By the approximate security of ApproxSS, H_1 and H_0 are indistinguishable.

H_2. This experiment is the same as H_1, except that the random value m_χ is simulated by sampling $\mathbf{e}' \leftarrow\!\!\$\, \{\mathbf{e} \mid \|\mathbf{e}\| < B_{sm}\}$

and computing $m_\chi = \Delta \cdot \mathbf{m} - \mathbf{c}_0 + \mathbf{e}'$, rather than sampling as $\mathbf{b} + \mathbf{x}$ with $\mathbf{x} \leftarrow\$ \chi$. Recall that we set $\chi = \mathsf{Uniform}(\mathcal{M}_{B_{sm}} = \{\mathbf{e} \mid \|\mathbf{e}\| < B_{sm}\})$. To demonstrate that H_1 and H_2 are indistinguishable, it suffices to show that $\mathbf{b} + \mathbf{x}$ is indistinguishable from $\Delta \cdot \mathbf{m} - \mathbf{c}_0 + \mathbf{e}'$ with $\mathbf{e}' \leftarrow\$ \{\mathbf{e} \mid \|\mathbf{e}\| < B_{sm}\}$. Note that $\mathbf{b} = \mathbf{c}_1 \cdot \mathsf{sk}$ and thus $\mathbf{b} + \mathbf{e} = \mathbf{c}_1 \cdot \mathsf{sk} + \mathbf{e} = \Delta \cdot \mathbf{m} - \mathbf{c}_0 + \mathbf{e}_{\mathsf{CT}} + \mathbf{e}$. When $B_{\mathsf{CT}}/B_{sm} \in \mathsf{negl}$, we have that $\mathbf{e}_{\mathsf{CT}} + \mathbf{e}$ is indistinguishable with $\mathbf{e}'$ by the smudging lemma [3]. Hence, χ is indistinguishable from $\Delta \cdot \mathbf{m} - \mathbf{c}_0 + \mathbf{e}'$ with $\mathbf{e}' \leftarrow\$ \{\mathbf{e} \mid \|\mathbf{e}\| < B_{sm}\}$ and thus H_1 and H_2 are indistinguishable.

H_3. This experiment is the same as H_2, except that the challenger does not generate the parties' ciphertext by encrypting their data. Instead, the challenger generates them by encrypting 0. By the CPA security of BFV encryption, H_3 is indistinguishable with H_2.

We note that H_3 is the ideal-world experiment. The reason is that H_3 simulates the view of adversary without using the private data of parties. Instead, the only used data is public parameters and the final output $\mathbf{m}$.

By the above arguments, the real-world experiment H_0 is indistinguishable from the ideal-world experiment H_3. Hence, ThFHE satisfies the simulation-based security. ■

C Proof of Theorem 3

Theorem 3 (Properties of ATASSES, Restated) *ATASSES satisfies χ-approximate correctness and $\mathcal{M}_B$-approximate security under RLWE-hardness assumption for $\chi = \mathsf{Uniform}(\{n \mid \|n\| \le B_{sm}\})$ and $\mathcal{M}_B = \{n \mid \|n\| \le T \cdot B_{sm}\}$.*

Proof: This proof consists of two parts, which prove the approximate correctness and the approximate security of ATASSES, respectively.

Approximate Correctness: We first prove the approximate correctness of ATASSES. By Definition 4, the approximate correctness says that the output of $\Pi_{\mathsf{ApproxRec}}$, i.e., m', should satisfy $m' = m + n$ for some $n \in \mathcal{M}_B$. Since ATASSES sets $\mathcal{M}_B = \{n \mid \|n\| \le T \cdot B_{sm}\}$, we just need to prove $m' = m + \sum_{i\in\mathcal{T}} n_i$ with $\|n_i\| \le B_{sm}, \forall i$. Since m' is obtained by decrypting CTall with key dk, it suffices to prove that CTall is truly the encryption of $m + \sum_{i\in\mathcal{T}} n_i$ using dk. Next, we prove this conclusion by proving two intermediate conclusions: 1) the CTall can be decrypted to $m + \sum_{i\in\mathcal{T}} n_i$ using the key $\sum_{i\in\mathcal{T}} L_i^{(\mathcal{T}_i)} \cdot \mathsf{ek}_{i,1} + \mathsf{ek}_{i,2}$ and 2) $\mathsf{dk} = \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T}_i)} \cdot \mathsf{ek}_{i,1} + \mathsf{ek}_{i,2}$.

Proof for intermediate conclusion 1. By the algorithm `PartyR1`, the ciphertexts are obtained as

$$\mathsf{CTs}_i = (\mathbf{a} \cdot \mathsf{ek}_{i,1} + \mathbf{e}_{i,1} + \Delta' \cdot s_i, -\mathbf{a}); \tag{8}$$

$$\mathsf{CTn}_i = (\mathbf{a} \cdot \mathsf{ek}_{i,2} + \mathbf{e}_{i,2} + \Delta' \cdot n_i, -\mathbf{a}); \tag{9}$$

$$\mathsf{CT}_{all} = (\sum_{i\in\mathcal{T}} (L_i \cdot \mathsf{CTs}_i[0] + \mathsf{CTn}_i[0]), -\mathbf{a}). \tag{10}$$

By substituting Eq.(8) and Eq.(9) to Eq.(10), decrypting CT_{all} using $\sum_{i\in\mathcal{T}} L_i^{(\mathcal{T}_i)} \cdot \mathsf{ek}_{i,1} + \mathsf{ek}_{i,2}$ leads to

$$\mathtt{BFV.Dec}(\sum_{i\in\mathcal{T}} L_i^{(\mathcal{T}_i)} \cdot \mathsf{ek}_{i,1} + \mathsf{ek}_{i,2}, \mathsf{CT}_{all}) \tag{11}$$

$$= \sum_{i\in\mathcal{T}} (L_i \cdot \mathsf{CTs}_i[0] + \mathsf{CTn}_i[0]) - \mathbf{a} \cdot (\sum_{i\in\mathcal{T}} L_i^{(\mathcal{T}_i)} \cdot \mathsf{ek}_{i,1} + \mathsf{ek}_{i,2}) \tag{12}$$

$$= \sum_{i\in\mathcal{T}} [\Delta' \cdot (L_i s_i + n_i) + (L_i \mathbf{e}_{i,1} + \mathbf{e}_{i,2})] \tag{13}$$

$$= \Delta' \cdot (m + \sum_{i\in\mathcal{T}} n_i) + \sum_{i\in\mathcal{T}} (L_i \mathbf{e}_{i,1} + \mathbf{e}_{i,2}). \tag{14}$$

Recall that we set $\Delta'/2 > N \cdot P' \cdot B'$. Hence, the norm of total error $\sum_{i\in\mathcal{T}} (L_i \mathbf{e}_{i,1} + \mathbf{e}_{i,2})$ must be bounded by $N \cdot P' \cdot B'$ as well as $\Delta'/2$. This aligns with the requirement of successful BFV decryption. Hence, decrypting CT_{all} using $\sum_{i\in\mathcal{T}} L_i^{(\mathcal{T}_i)} \cdot \mathsf{ek}_{i,1} + \mathsf{ek}_{i,2}$ leads to $m + \sum_{i\in\mathcal{T}} n_i$. The first intermediate conclusion is proved.

Proof for intermediate conclusion 2. Note that dk is generated by running the recovery algorithm of Shamir secret sharing over shares $\{\mathsf{dkShare}_j\}_{j\in\mathcal{T}_2}$. Each decryption key share is generated by $\mathsf{dkShare}_j = \sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot \mathsf{ekShare}_{i,j,1} + \mathsf{ekShare}_{i,j,2}$. By the linearity of Shamir secret sharing, dk is equal to $\sum_{i\in\mathcal{T}} L_i^{(\mathcal{T})} \cdot \mathsf{ek}_{i,1} + \mathsf{ek}_{i,2}$. The second intermediate conclusion is proved.

Based on the above two intermediate conclusions, CTall is truly the encryption of $m + \sum_{i\in\mathcal{T}} n_i$ using dk. Hence, $m' = m + n$ for some $n \in \mathcal{M}_B = \{n \mid \|n\| \le T \cdot B_{sm}\}$. The approximate correctness of ATASSES holds.

Approximate Security: Below, we first prove a property of RLWE secret-key encryption and then use it to prove the semi-honest security of ATASSES. Our proof relies on the following lemma from the full version of [5].

Lemma 2 (Lemma 4, Appendix A in [5]) *For any $\sigma_1 > 0$, for any $m, K, Q, P, \Delta, l \ge 1$, let $k = l/K$ and $x_1, \ldots, x_m \in \mathbb{Z}_P^l \equiv \mathcal{R}_P^k$. Assume $\mathsf{RLWE}_{K,q,\sigma}$ is hard for $\sigma = 1/\sqrt{2}\sigma_1$. Then, the following two distributions D_0 and D_1 are indistinguishable.*

$$D_0 = \left\{ \begin{array}{l} (A, As_1 + (e_1 + f_1) + \Delta x_1, \ldots, \\ \quad As_{m-1} + (e_{m-1} + f_{m-1}) + \Delta x_{m-1}, \\ \quad -A \sum_{i=1}^{m-1} s_i + (e_m + f_m) + \Delta x_m) \mod Q : \\ A \leftarrow \mathcal{R}_Q^k, s_1, \ldots, s_{m-1} \leftarrow\$ \chi_s, e_i, f_i \leftarrow\$ D_{\sigma_1}^k, \forall i \end{array} \right\}$$

$$D_1 = \left\{ \begin{array}{l} (A, u_1, \ldots, u_{m-1}, \\ \quad -\sum_{i=1}^{m-1} u_i + \sum_{i=1}^{m} (e_i + f_i) + \Delta \sum_{i=1}^{m} x_i) \mod Q : \\ A \leftarrow \mathcal{R}_Q^k, u_1, \ldots, u_{m-1} \leftarrow\$ \mathcal{R}_Q^k, e_i, f_i \leftarrow\$ D_{\sigma_1}^k, \forall i \end{array} \right\}$$

Intuitively, this lemma demonstrates that the joint distribution of the ciphertexts of x_is (D_0) is indistinguishable from that of random values (D_1), conditioned on the sum of these ciphertexts is the same as the sum of random values. To apply this lemma to our proof, we slightly modify it by considering the indistinguishability between two new distributions D'_0 and D'_1. The differences between D_0 and D'_0, as well as D_1 and D'_1, are highlighted in red color. Basically, D'_0 (resp. D'_1) contains an extra random value As and this As is added to the last ciphertext in D'_0 (resp. the last random value in D'_1). By Lemma 2, D'_0 and D'_1 are also indistinguishable.

$$D'_0 = \left\{ \begin{array}{l} (A, As, As_1 + (e_1 + f_1) + \Delta x_1, \ldots, \\ \quad As_{m-1} + (e_{m-1} + f_{m-1}) + \Delta x_{m-1}, \\ \quad As - A\sum_{i=1}^{m-1} s_i + (e_m + f_m) + \Delta x_m) \mod Q : \\ A \leftarrow\$ \mathcal{R}_Q^k, s, s_1, \ldots, s_{m-1} \leftarrow\$ \chi_s, e_i, f_i \leftarrow\$ D_{\sigma_1}^k, \forall i \end{array} \right\}$$

$$D'_1 = \left\{ \begin{array}{l} (A, As, u_1, \ldots, u_{m-1}, \\ \quad As - \sum_{i=1}^{m-1} u_i + \sum_{i=1}^{m} (e_i + f_i) + \Delta \sum_{i=1}^{m} x_i) \mod Q : \\ A \leftarrow\$ \mathcal{R}_Q^k, s \leftarrow\$ \chi_s, u_1, \ldots, u_{m-1} \leftarrow\$ \mathcal{R}_Q^k, e_i, f_i \leftarrow\$ D_{\sigma_1}^k, \forall i \end{array} \right\}$$

Intuitively, the indistinguishability between D'_0 and D'_1 states that the joint distribution of the ciphertexts of $x_1, \ldots, x_m$s encrypted by different encryption keys $s_1, \ldots, s_{m-1}, s - \sum_{i \in [m-1]} s_i$ is indistinguishable from that of random values, conditioned on 1) the sum of these ciphertexts is the same as the sum of random values and 2) the sum of encryption keys s is publicly known. In other words, the sum of encryption keys is safe to disclose without revealing the encryption keys and the encrypted messages.

Next, we prove the approximate security of ApproxSS based on this conclusion. Note that ATASSES consists of two rounds. By Definition 3, to prove the approximate security of ATASSES, we need to prove the existence of simulator algorithms $\{S_0, S_1, S_2\}$, so that the real-world experiment $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}$ and ideal-world experiment $\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}$ are indistinguishable. Next, the proof proceeds by first describing the simulator algorithms $\{S_0, S_1, S_2\}$ and then constructing successive hybrid experiments between $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}$ and $\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}$.

Before describing the simulator algorithms, we give some notations. Let $\mathcal{A}_r$ and $\mathcal{H}_r$ denote the set of corrupted participants and the set of honest participants in round r, respectively. Hence, the set $\mathcal{T}_r$ of participants in round r satisfies $\mathcal{T}_r = \mathcal{A}_r \cup \mathcal{H}_r$. Note that there exists at least one honest participant by our system model. We use h_r to denote this participant in round r. In addition, we use the same notation as Section 2.3 in [5] and rewrite an RLWE sample from $\mathbf{a} \cdot \mathsf{sk} + \mathbf{e}$ to $A \cdot sk + e$, where A, sk, e are matrices and vectors generated from the coefficient embedding of polynomials $\mathbf{a}, \mathsf{sk}, \mathbf{e}$. Such a rewritting helps to simplify the description with long message. Recall that in ATASSES, the message m_i can be too long to be encrypted into one ciphertext. Hence, party i needs to break down m_i into C' sub-vectors and encrypt each sub-vector $m_{i,k}$ using $\mathbf{a}_k$ and $\mathbf{e}_k$, respectively. By using the matrix-vector notation, we can rewrite the encryption of long message m_i as $A \cdot s + e + \Delta m_i$, where A and e are the concatenation of $A_1, \ldots, A_{C'}$ and $e_1, \ldots, e_{C'}$, respectively.

Simulator algorithm $S_0(N, T, \mathcal{M})$*:* This algorithm outputs $\{s_i\}_{i \in \mathcal{A}}$ by sampling s_is from $\mathsf{Uniform}(\mathcal{M})$.

Simulator algorithm $S_1(m_\chi, \mathcal{T}_1)$*:* Recall that algorithm `PartyR1` of ATASSES asks each party $i \in \mathcal{T}_1$ to generate two things: 1) the shares of encryption keys $\mathsf{ekShare}_{j,i,1}$ and $\mathsf{ekShare}_{j,i,2}$ and 2) the ciphertexts $\mathsf{CTs}_{i,k}$ and $\mathsf{CTn}_{i,k}$. The S_1 simulates these outputs as follows:

- If $i \in \mathcal{A}_1$, S_1 follows algorithm `PartyR1`() in the real world to generate those outputs.
- If $i \in \mathcal{H}_1 \setminus \{h_1\}$, S_1 samples its encryption key shares $\mathsf{ekShare}_{j,i,1}$ and $\mathsf{ekShare}_{j,i,2}$ for all $j \in [N]$ from the uniform distribution over the ciphertext space. As for the ciphertexts, S_1 uses the same $A \leftarrow\$ \mathsf{CRS}$ as $\mathsf{CTs}_i[1]$ and $\mathsf{CTn}_i[1]$ for all i, but samples uniformly-random elements from ciphertext space as $\mathsf{CTs}_i[0]$ and $\mathsf{CTn}_i[0]$.
- If $i = h_1$, S_1 samples its encryption key shares $\mathsf{ekShare}_{j,h_1,1}$ and $\mathsf{ekShare}_{j,h_1,2}$ for all $j \in [N]$ from the uniform distribution over the ciphertext space. As for the ciphertexts, S_1 uses the same $A \leftarrow\$ \mathsf{CRS}$ as $\mathsf{CTs}_{h_1}[1]$ and $\mathsf{CTn}_{h_1}[1]$ and samples uniformly-random elements from ciphertext space as $\mathsf{CTs}_{h_1}[0]$. As for $\mathsf{CTn}_{h_1}[0]$, S_1 generates it as follows:

$$\begin{aligned} \mathsf{CTn}_{h_1}[0] &= A \cdot s + \sum_{i \in \mathcal{T}_1} L_i^{(\mathcal{T}_1)} e_{i,1} + \sum_{i \in \mathcal{H}_1} e_{i,2} \\ &+ \Delta' \cdot (m_\chi - \sum_{i \in \mathcal{A}_1} L_i^{(\mathcal{T}_1)} s_i + \sum_{i \in \mathcal{H}_1 \setminus \{h_1\}} n_i) \\ &- \sum_{i \in \mathcal{T}_1} L_i^{(\mathcal{T}_1)} \cdot \mathsf{CTs}_i[0] - \sum_{i \in \mathcal{H}_1 \setminus \{h_1\}} \mathsf{CTn}_i[0] \end{aligned} \tag{15}$$

  where s is sampled from the key distribution, $e_{i,1}$, $e_{i,2}$ are the random errors sampled from the distribution as $e_i + f_i$, and n_is are the random noises sampled from $\{n \mid \|n\| < B_{sm}\}$.

Simulator algorithm $S_2(m_\chi, \mathcal{T}_2)$*:* Recall that algorithm `PartyR2` of ATASSES asks each party $j \in \mathcal{T}_2$ to generate a decryption share $\mathsf{dkShare}_j$. The S_2 simulates $\{\mathsf{dkShare}_j\}_{j \in \mathcal{T}_2}$ as follows:

- If $j \in \mathcal{A}_2$, S_2 follows algorithm `PartyR2`() in the real world to output $\mathsf{dkShare}_j$.
- If $j \in \mathcal{H}_2 \setminus \{h_2\}$, S_2 samples $\mathsf{dkShare}_j$ from the uniform distribution over the ciphertext space.

- If $j = h_2$, S_2 generates $\mathsf{dkShare}_{h_2}$ as

$$(L_{h_2}^{(\mathcal{T}_2)})^{-1} \cdot (s + \sum_{i \in \mathcal{A}_1} \mathsf{ek}_{i,2} - \sum_{j \in \mathcal{T}_2 \setminus \{h_2\}} L_j^{(\mathcal{T}_2)} \mathsf{dkShare}_j). \tag{16}$$

Based on the above simulator algorithms, we construct the following hybrid experiments by gradually replacing the real-world algorithms by the simulator algorithms.

H_1: this is same as $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}$, except that the generation of ciphertexts in algorithm PartyR1 and algorithm PartyR2 are replaced by simulator algorithms S_1 and S_2. Notably, the encryption key shares in algorithm PartyR1 remain the same as in $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}$.

The difference between $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}$ and H_1 can be summarized in the form of D'_0 and D'_1.

- For H_1: In S_1, we remark that $\{L_i^{\mathcal{T}_1} \cdot \mathsf{CTs}_{i,k}[0]\}_{i \in \mathcal{T}_1} \cup \{\mathsf{CTn}_{i,k}[0]\}_{i \in \mathcal{H}_1 \setminus \{h_1\}}$ plays the role of the former $m-1$ random values $u_1, \ldots, u_{m-1}$ in distribution D'_1. The $\mathsf{CTn}_{h_1}[0]$ corresponds to the last random value in D'_1. Particularly, $(m\chi - \sum_{i \in \mathcal{A}_1} L_i^{(\mathcal{T}_1)} s_i + \sum_{i \in \mathcal{H}_1 \setminus \{h_1\}} n_i)$ corresponds to $\sum_{i \in [m]} x_i$, and $\sum_{i \in \mathcal{T}_1} L_i^{(\mathcal{T}_1)} e_{i,1} + \sum_{i \in \mathcal{H}_1} e_{i,2}$ corresponds to $\sum_{i \in [m]} (e_i + f_i)$. In S_2, $\sum_{j \in \mathcal{T}_2} L_j^{(\mathcal{T}_2)} \cdot \mathsf{dkShare}_j - \sum_{i \in \mathcal{A}_1} \mathsf{ek}_{i,2}$ corresponds to s, and thus, $A \cdot (\sum_{j \in \mathcal{T}_2} L_j^{(\mathcal{T}_2)} \cdot \mathsf{dkShare}_j - \sum_{i \in \mathcal{A}_1} \mathsf{ek}_{i,2})$ corresponds to As.

- For $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}$, we remark that $\{L_i^{\mathcal{T}_1} \cdot \mathsf{CTs}_{i,k}[0]\}_{i \in \mathcal{T}_1} \cup \{\mathsf{CTn}_{i,k}[0]\}_{i \in \mathcal{H}_1 \setminus \{h_1\}}$ plays the role of the former $m-1$ ciphertexts. By the proof of approximate correctness, we have $\mathsf{dk} = \sum_{i \in \mathcal{T}_1} L_i^{(\mathcal{T}_1)} \cdot \mathsf{ek}_{i,1} + \mathsf{ek}_{i,2}$ and $\mathsf{dk} = \sum_{j \in \mathcal{T}_2} L_j^{(\mathcal{T}_2)} \cdot \mathsf{dkShare}_j$. By combining these two equations, we have $\mathsf{ek}_{h_2,2} = \sum_{j \in \mathcal{T}_2} L_j^{(\mathcal{T}_2)} \cdot \mathsf{dkShare}_j - \sum_{i \in \mathcal{T}_1} L_i^{(\mathcal{T}_1)} \cdot \mathsf{ek}_{i,1} - \sum_{i \in \mathcal{T}_1 \setminus \{h_1\}} \mathsf{ek}_{i,2}$, which corresponds to $s - \sum_{i \in \mathcal{T}_1} L_i^{(\mathcal{T}_1)} \cdot \mathsf{ek}_{i,1} - \sum_{i \in \mathcal{H}_1 \setminus \{h_1\}} \mathsf{ek}_{i,2}$ in D'_0. Hence, $\sum_{j \in \mathcal{T}_2} L_j^{(\mathcal{T}_2)} \cdot \mathsf{dkShare}_j - \sum_{i \in \mathcal{A}_1} \mathsf{ek}_{i,2}$ corresponds to s, and thus, $A \cdot (\sum_{j \in \mathcal{T}_2} L_j^{(\mathcal{T}_2)} \cdot \mathsf{dkShare}_j - \sum_{i \in \mathcal{A}_1} \mathsf{ek}_{i,2})$ corresponds to As.

By the above arguments on differences, to distinguish between $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}$ and H_1, the adversary is in fact to distinguish two distributions in the form of D'_0 and D'_1. Nonetheless, D'_0 and D'_1 are indistinguishable under RLWE-hardness assumption. Hence, $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}$ and H_1 are indistinguishable.

H_2: this is same as H_1, except that the encryption key shares is generated by simulator algorithm S_0 instead of PartyR1.

The only difference between H_2 and H_1 is the encryption key shares of parties corrupted by $\mathcal{A}$. We note that by the security of Shamir secret sharing, the T-out-of-N shares from $T-1$ parties are indistinguishable from uniformly-random values. Since $\mathcal{A}$ can corrupt at most $T-1$ parties by our threat model, H_1 and H_2 are indistinguishable.

H_3: this is same as H_2, except that the algorithm Share is replaced by simulator algorithm S_0. We note that H_3 is equivalent to $\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}$.

The only difference between H_3 and H_2 is the shares of parties corrupted by $\mathcal{A}$. We note that by the security of Shamir secret sharing, the T-out-of-N shares from $T-1$ parties are indistinguishable from uniformly-random values. Since $\mathcal{A}$ can corrupt at most $T-1$ parties by our threat model, H_1 and H_2 are indistinguishable.

Based on the above arguments, $\mathsf{Expt}_{\mathcal{A},\mathsf{Real}}$ and $\mathsf{Expt}_{\mathcal{A},\mathsf{Ideal}}$ are indistinguishable and thus ATASSES satisfies approximate security. ■

D Complexity Analysis

Below we analyze the computation and communication complexity of four ApproxSS schemes, namely $\{0,1\}$-ApproxSS, Type-I Shamir ApproxSS, Type-II Shamir ApproxSS, and ATASSES. In the analysis, we set T as the order of $O(N)$, as T is usually set to be the proportion of N, say $0.5N$ or $0.9N$. The computation complexity is set to be the summation of computation complexity of the aggregator and a single party. The reason is that the parties execute their algorithms in parallel. Hence, the computation time should be counted as the computation time of the slowest party, rather than the summation of all parties. The communication complexity is set to be the size of transferred message between a pair of parties (or between a party and an aggregator), rather than the size of transferred message between a party and every parties. The reason is that we assume a party has the communication channel with every parties (and the aggregator). Hence, the messages to N parties can be transferred in parallel, rather than to each party one by one.

$\{0,1\}$-ApproxSS: Existing work [7] shows that the size of each share of $\{0,1\}$-ApproxSS is $O(N^{4.2})$ on average. When the data has the size of K, the share size is $O(N^{4.2} \cdot K)$. The computation workload of each party is to sample a noise whose size is $O(N^{4.2} \cdot K)$ and to add the share and the noise together. Hence, the computation complexity of each party is $O(N^{4.2} \cdot K)$. The computation workload of the aggregator is to recover the approximate message from T noisy shares. Hence, the computation complexity of the party is $T \cdot O(N^{4.2} \cdot K) = O(N^{5.2} \cdot K)$. Therefore, the overall computation complexity is $O(N^{5.2} \cdot K)$. As for the communication complexity, each party needs to send the noisy share to the aggregator, with the complexity being $O(N^{4.2} \cdot K)$. Hence, the communication complexity is $O(N^{4.2} \cdot K)$.

Type-I Shamir ApproxSS: When applying Type-I Shamir ApproxSS, the message space has a modulus $O(N \cdot (N!)^3)$. Hence, the size of each element is $O(\log(N \cdot (N!)^3))$. By Stirling's formula, the element's size can be approximated to $O(N \log N)$, which is $O(N)$ times the element's size of other ApproxSS schemes. Hence, the size of each share is regarded as $O(NK)$. The computation workload is dominated by the

aggregator, who needs to compute Lagrange coefficients with complexity $O(N^2)$ and compute the linear combination of T shares with complexity $T \cdot O(NK) = O(N^2K)$. Hence, the computation complexity is $O(N^2K)$. As for the communication complexity, each party needs to send the noisy share to the aggregator, with the complexity being $O(NK)$. Hence, the communication complexity is $O(NK)$.

Type-II Shamir ApproxSS: Type-II Shamir ApproxSS consists of two rounds. In the first round, each party generates the shares of a length-K noise and sends a share to each party with computation complexity $O(N^2K)$ and communication complexity $O(NK)$. In the second round, each party adds T noises and the share together and sends the noisy share to the aggregator, resulting in computation complexity $O(N^2K)$ and communication complexity $O(NK)$. For the aggregator, to recover the approximate message, it needs to compute Lagrange coefficients with complexity $O(N^2)$ and compute the linear combination of T shares with complexity $T \cdot O(NK) = O(N^2K)$. Hence, the computation complexity is $O(N^2K)$. In total, the computation complexity is $O(N^2K)$ and the communication complexity is $O(NK)$.

Type-III Shamir ApproxSS: Type-III Shamir ApproxSS requires only one round. Each party computes a T-out-of-T secret key share with computation complexity $O(N)$ and then generates a decryption share using this secret key share with computation complexity $O(K)$. Hence, the total computation complexity is $O(N+K)$. Then the party needs to send this decryption share to the aggregator with communication complexity $O(K)$. Next, the aggregator can recover the message by aggregating T decryption shares with computation complexity $O(NK)$. In total, the computation complexity is $O(NK)$ and the communication complexity is $O(K)$.

ATASSES: ATASSES also consists of two rounds. In the first round, each party generates the shares of encryption keys and sends each share to each party with computation complexity $O(N^2)$ and communication complexity $O(N)$. In addition, each party also needs to encrypt the message's shares and noises and sends the ciphertexts to the aggregator. The computation complexity is $O(K)$ and communication complexity is $O(K)$. The aggregator needs to compute Lagrange coefficients with communication complexity $O(N^2)$ and computation complexity $O(N)$. In the second round, each party adds $2T$ encryption key shares together and sends the decryption key share to the aggregator, resulting in computation complexity $O(N^2)$ and communication complexity $O(N)$. For the aggregator, to recover the decryption key, it needs to compute the linear combination of T decryption key shares with complexity $T \cdot O(N) = O(N^2)$. Then it needs to compute the overall ciphertext by summing T ciphertexts up with computation complexity $O(NK)$ and decrypting it with computation complexity $O(K)$. In total, the computation complexity is $O(N^2+NK)$ and the communication complexity is $O(N+K)$.

E Extension to Fully-Malicious Security

Below we provide a brief discussion on how to extend our protocol from semi-honest security to malicious security. Note that the key difference between semi-honest security and malicious security is the adversary's capability. The semi-honest adversary must follow the protocol, while the malicious adversary can operate arbitrarily. Particularly, the actions of malicious adversary can be divided into two categories: one is to abort its corrupted parties and the other is to ask its corrupted parties to mistakenly execute the operations. Our work mainly discuss how to deal with the first category. Hence, to extend to fully-malicious security, we need to further enable parties to prove the correctness of operations they execute. Thanks to the previous work on maliciously secure ThFHE (e.g., PELTA [13]) and verifiable secret sharing [11, 31], we can adopt their techniques to achieve this goal. Next, we first recap the key techniques in existing works and then discuss how to apply their techniques in our work.

Recap. Below we recap the PELTA framework and verifiable secret sharing, respectively.

PELTA first categorizes the operations in ThFHE schemes into non-interactive operations and interactive operations. For non-interactive operations such as encryption and homomorphic evaluation, PELTA pointed out that their correctness verification has been addressed by previous works including [6, 8, 12, 26]. Hence, PELTA put the focus on those interaction operations and identify that these operations share common functionalities. Namely, the interactive operations in ThFHE schemes can be decomposed into multiple 2-step interactions. Each interaction consists of the following two steps.

Step 1: In this step, every party i uses its local secret s_i to locally generate a share that can be publicly disclosed. These shares have a common structure. Particularly, a share b_i is computed as a linear equation of the form:

$$b_i = a \cdot s_i + X + e_i, \tag{17}$$

where a is a publicly known value, s_i is the secret value of party i, e_i is a newly sampled error term from some distribution, and X is a placeholder that takes different forms in different operations.

Step 2: In this step, the aggregator computes the (weighted) summation of the shares from parties up and outputs a collective value b.

For such 2-step interactions, they further show how to verify the correctness of these 2-step interactions via zero-knowledge proof techniques, which can be used for the extension of our work.

Verifiable secret sharing (VSS), introduced by Chor et al. [20], aims to make the (vanilla) secret sharing technique robust against malicious parties. Particularly, it not only prevents a malicious dealer from distributing incorrect shares,

but also prevents malicious shareholders from submitting incorrect shares in the reconstruction protocol. VSS will be used when our work relies on secret sharing techniques.

Main Idea. Recall that the ATASSES construction (see Figure 6) mainly consists of two parts: one is *ciphertext generation* and the other is *decryption key generation*. The first part involves the BFV encryption (Lines 5–10) executed by each party and the ciphertext summation (Lines 15–16) by the aggregator. We note that these two steps exactly match the above 2-step interaction and thus can be verified by PELTA's techniques. The second part includes the secret key sharing (Lines 2–4), combination (Lines 12-13), and recovery (Line 14). This part is executed by Shamir secret sharing in our semi-honest version and can be boosted to maliciously-secure construction by verifiable secret sharing such as [11, 31].

Besides these two parts of ATASSES, there are still several simple operations in ATASSES, including BFV secret key generation (Line 1), Lagrange coefficients computation (Line 11), and BFV decryption (Line 17). The BFV secret key generation is a non-interactive operation and its verification has been discussed in PELTA. As for the Lagrange coefficients computation and BFV decryption, these operations do not involve any secret information and thus every party can check its correctness by simply re-executing these operations.

In addition, our ThFHE construction has several extra operations beyond the ATASSES. One is secret key sharing (Lines 1–6) in the key generation stage. Similarly, it can be verified by verifiable secret sharing. The other is Phase 1 and Phase 3 in the decryption stage. The Phase 1 asks each party i to output a share $b_i = c_1 \cdot \mathsf{skShare}_i + c_0$, which is exactly the Step 1 in the above 2-step interaction and can be verified by utilizing PELTA's techniques. The Phase 3 is to decode the plaintext m from b'. This phase also does not involve any secret information. Hence, it can be verified by asking every party to re-execute this phase.