






Provably Secure Approximate Computation Protocols from CKKS

Intak Hwang¹ , Yisol Hwang¹ , Miran Kim² , Dongwon Lee¹ , and Yongsoo Song¹ 

¹ Seoul National University

{intak.hwang, yisol.hwang, dongwonlee95, y.song}@snu.ac.kr

² Hanyang University, Republic of Korea

miran@hanyang.ac.kr

Abstract. Secure multi-party computation (MPC) enables collaborative, privacy-preserving computation over private inputs. Advances in homomorphic encryption (HE), particularly the CKKS scheme, have made secure computation practical, making it well-suited for real-world applications involving approximate computations. However, the inherent approximation errors in CKKS present significant challenges in developing MPC protocols.

This paper investigates the problem of secure approximate MPC from CKKS. We first analyze CKKS-based protocols in two-party setting. When only one party holds a private input and the other party acts as an evaluator, a simple protocol with the noise smudging technique on the encryptor’s side achieves security in the standard manner. When both parties have private inputs, we demonstrate that the protocol incorporating independent errors from each party achieves a relaxed standard security notion, referred to as a liberal security. Nevertheless, such a protocol fails to satisfy the standard security definition. To address this limitation, we propose a novel protocol that employs a distributed sampling approach to generate smudging noise in a secure manner, which satisfies the standard security definition.

Finally, we extend the two-party protocols to the multi-party setting. Since the existing threshold CKKS-based MPC protocol only satisfies the liberal security, we present a novel multi-party protocol achieving the standard security by applying multi-party distributed sampling of a smudging error. For all the proposed protocols, we formally define the functionalities and provide rigorous security analysis within the simulation-based security framework. To the best of our knowledge, this is the first work to explicitly define the functionality of CKKS-based approximate MPC and achieve formal security guarantees.

Keywords: Secure Approximations, Homomorphic Encryption, Multi-party Computation, CKKS

1 Introduction

Homomorphic Encryption (HE) is a cryptographic scheme that allows computations to be performed directly on encrypted data without requiring decryption. It has emerged as one of the privacy-enhancing techniques due to its simplicity, versatility, and communication-efficiency. This capability enables secure outsourcing of computation, allowing clients to leverage the computational resources of an untrusted third party while preserving data privacy.

In a typical two-party HE-based protocol, the client acts as the key owner, while the server as the evaluator. This framework can be applied to any two-party computation tasks, such as genomic analysis [20], secure inference [15, 21] and privacy-preserving machine learning [26]. In the multi-party setting, HE-based protocols have been widely adopted across various domains, including privacy-preserving federated learning [13, 18, 30].

Among various HE schemes, the CKKS scheme [8] has garnered significant attention due to its support for fixed-point arithmetic in an approximate manner, making it particularly suitable for privacy-preserving computations involving real-number operations. However, an inherent characteristic of CKKS poses major challenges in constructing CKKS-based secure multi-party computation (MPC) protocols for approximate computation. Specifically, in CKKS, the error from approximate computation is entangled with the error introduced for security. So, it is not straightforward to apply cryptographic techniques such as circuit

privacy or noise smudging [10, 19, 31, 33, 35], which are commonly used in MPC protocols based on traditional exact HE schemes. Since these techniques can change plaintext values, they can directly impact the functionality of the protocol, making it non-trivial to define functionalities and address security concerns that do not arise in other HE schemes.

Most of all, the security of an underlying HE scheme is typically analyzed in terms of indistinguishability-based security notion (e.g., indistinguishability under chosen plaintext attacks or its stronger notion with a decryption oracle [27]). However, achieving security in approximate MPC protocols requires us to investigate simulation-based security framework beyond the indistinguishability security notion. As a result, to the best of our knowledge, there is no existing work to explicitly define the functionality of CKKS-based approximate MPC protocols or provide a formal security analysis.

1.1 Our Contribution

In this paper, we investigate various approximate MPC protocols, formally demonstrate their functionalities, and analyze security guarantees and limitations. Specifically, we are focus on CKKS-based protocols that compute a public circuit C on private inputs x_i and return an approximate result \hat{z} of the exact computation $z = C(x_1, \dots, x_n)$ to all (or a designated subset of) parties.

At a high level, we consider two main scenarios: (1) a client-server asymmetric two-party computation (2PC) model and (2) symmetric MPC among multiple parties. From a security perspective, we explore both the standard MPC security definition [16] and a weaker notion called *liberal security* [12]. More specifically, under the standard definition, the simulator must reconstruct the view of an adversary A using only its input $(x_i)_{i \in A}$ and output \hat{z} . In contrast, under the liberal definition, the simulator is additionally provided with the exact output value $z = C(x_1, \dots, x_n)$. Roughly speaking, the liberal definition implies that the adversary learns no more than z beyond its intended output \hat{z} from the protocol, making it a weaker security notion than the standard definition. However, in certain use cases, this weaker definition enables more efficient protocols while still providing sufficient security guarantees. For further details, see Section 3.

Within the 2PC setting, we further distinguish between simple and general cases. In this scenario, only the client holds a private input, while the server acts solely as an evaluator of a public circuit. We demonstrate that a secure protocol satisfying standard security can be constructed using a simple noise smudging technique applied on the client side. While this approach provides a straightforward solution, its functionality is limited to evaluating a public circuit that the client could, in principle, compute alone. As a result, this case is theoretically less interesting and offers limited practical implications.

In the general case, both the client and the server hold private inputs and security considerations become significantly more challenging. For this case, we propose two protocols. In the first protocol, each party independently applies noise smudging. This approach is more efficient but only satisfies the liberal security definition. To achieve standard security, we introduce a second protocol that incorporates an additional distributed smudging sampling process.

Similarly, in the multi-party setting, we show that the approach where each party adds a smudging error during decryption and merges the results only satisfies the liberal security definition but does not achieve standard security. However, by utilizing distributed smudging error sampling, we demonstrate that it is possible to construct an MPC protocol that satisfies standard security.

Finally, for all the proposed protocols, we explicitly define their functionalities and provide formal security proofs to rigorously establish their correctness and security guarantees.

1.2 Technical Overview

Below we present our main ideas and results in more detailed level, and provide a road-map.

Section 4: Two-Party Computation. We mainly propose three protocols for different scenarios and security notions.

1. Section 4.1: Simple case - standard security

When only the client holds a private input, we demonstrate a secure protocol satisfying standard

security. On receiving the resulting ciphertext from the server after homomorphic evaluation, the client decrypts it and adds a smudging error which is much larger than the (unknown) error induced from approximate operations of CKKS. Then, the client may or may not share the output with the server. Notably, this scenario is closely related to recent studies on IND-CPA^D attacks [27] and their mitigation strategies [28].

2. Section 4.2: General case - liberal security

When both the client and the server have private inputs, using noise smudging on the client side or ciphertext randomization on the server side (which involves adding a random encryption of zero with a smudging error) alone is insufficient to achieve a secure protocol. However, we demonstrate that by integrating both techniques, it is possible to construct an efficient protocol satisfying the liberal security. However, this protocol does not achieve the standard security.

3. Section 4.3: General case - standard security

We discuss why the previous protocol fails to satisfy standard security and conclude that if one party gains too much (a non-negligible amount of) information about the smudging error, constructing a simulator for this party’s view becomes infeasible. To overcome this issue, we introduce an additional *distributed smudging error sampling* functionality. This functionality samples an error from the smudging distribution, generates a random additive share of the error, and distributes one share to each party. In the actual protocol, after performing homomorphic evaluation, the server adds its share, while the client adds its own share and decrypts the result, obtaining an approximation with the smudging error added. Since both parties gain no knowledge of the smudging error, the protocol satisfies stronger security requirements. Furthermore, we demonstrate that the functionality of distributed smudging error sampling can be implemented efficiently using oblivious transfer.

Section 5: Multi-Party Computation. In the multi-party setting where all parties act in a symmetric manner, we present two protocols for different security notions.

1. Section 5.1: Liberal security

Several studies have explored the construction of multi-party protocols using CKKS, following a similar distributed decryption approach in which each party independently samples a smudging error, adds it to its partial decryption, and then merges the results. However, to the best of our knowledge, no prior work has formally defined the functionality of these protocols or provided a rigorous security proof. In this work, we formally define their functionality and demonstrate that they satisfy liberal security.

2. Section 5.2: Standard security

Similar to the 2PC case, the above MPC protocol does not satisfy the standard security notion. However, we show that by incorporating multi-party distributed sampling of a smudging error, the protocol can be also modified to achieve standard security.

1.3 Related Works

In a standard HE-based 2PC protocol, there are two entities: the client who acts as the encryptor and decryptor, and the server who serves as the evaluator. To mitigate threats to client privacy, the noise smudging (flooding) technique [14, 35] has been employed to ensure that a semi-honest adversary cannot obtain any secret information from a result ciphertext. In the context of approximate HE schemes, a similar approach to noise smudging is used, where a calibrated amount of noise is added to accommodate the worst-case error growth during homomorphic computation [28]. Meanwhile, several studies have explored circuit privacy and sanitization techniques to ensure the privacy of evaluator (e.g., [2, 11, 14, 23]). A common approach is to randomize the resulting ciphertext after homomorphic evaluation, thereby eliminating any unnecessary information except for the underlying plaintext. However, a direct application of this approach to a CKKS-based protocol does not guarantee security, as it remains vulnerable to key-recovery attacks [27]. To the best of our knowledge, there is no existing work to address the privacy of both the client and the server, and it is not clear how to construct a secure 2PC protocol based on approximate HE schemes using these cryptographic techniques.

While the standard (single-key) CKKS scheme supports homomorphic operations only between ciphertexts encrypted under the same key, several variants have been proposed to extend CKKS for MPC, including threshold CKKS [1, 33, 34], multi-key CKKS [6, 22], and multi-group CKKS [24]. Furthermore, several studies [13, 33, 34] have explored privacy-preserving MPC over approximate values using the threshold CKKS scheme. In the threshold setting, decryption is carried out in a distributed manner, where each key owner publishes a partial decryption result of the evaluated ciphertexts. In all previous works, each party introduces independent auxiliary noise into its partial decryption to prevent noise leakage. However, none of these studies explicitly define the functionality of the MPC protocol or provide a formal security proof - a gap that will be discussed in Section 5.

2 Preliminaries

2.1 Notations

We use bold lower-case letters to denote column vectors. Given an integer $n \in \mathbb{N}$, we denote by $[n]$ the set $\{i \in \mathbb{N} \mid 1 \leq i \leq n\}$. Let n be a power of two and q be an integer. We denote by $R = \mathbb{Z}[X]/(X^n + 1)$ the ring of integers of the $2n$ -th cyclotomic field and $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ the residue ring of R modulo q . We denote sampling x from the distribution \mathcal{D} by $x \leftarrow \mathcal{D}$. For distributions \mathcal{D}_1 and \mathcal{D}_2 over a countable set S (e.g. \mathbb{Z}^n), the statistical distance between \mathcal{D}_1 and \mathcal{D}_2 is defined as $\frac{1}{2} \cdot \sum_{x \in S} |\mathcal{D}_1(x) - \mathcal{D}_2(x)| \in [0, 1]$. We denote the uniform distribution over S by $\mathcal{U}(S)$ when S is finite. For a vector x in \mathbb{R}^n or \mathbb{C}^n and for $p \in [1, \infty]$, we define the ℓ_p norm as $\|x\|_p = (\sum_{i \in [n]} |x_i|^p)^{1/p}$ when $p < \infty$ and $\|x\|_\infty = \max_{i \in [n]} \{x_i\}$. We also define a *smudging distribution* as follows.

Definition 1. Let $E > 0$ be a constant. A distribution \mathcal{D} over R is called a E -smudging distribution if \mathcal{D} and $e + \mathcal{D}$ are statistically indistinguishable for any $e \in R$ such that $\|e\| \leq E$.

2.2 The CKKS Scheme

HE is an encryption scheme that enables certain operations to be performed on encrypted data without decryption. Since Gentry's breakthrough [14], various HE schemes based on the (Ring) Learning with Errors (LWE/RLWE) assumption have been proposed, including BGV [5], BFV [3, 4], CKKS [8], and THE [9]. Among these, the CKKS scheme has received significant attention due to its support of fixed-point arithmetic in an approximate manner. In CKKS, the message space and the plaintext spaces are defined as $\mathbb{C}^{N/2}$ and $R = \mathbb{Z}[X]/(X^N + 1)$ for a power-of-two N , respectively. This structure allows multiple values to be encrypted into a single ciphertext, enabling efficient computations in a Single Instruction Multiple Data (SIMD) manner.

- **CKKS.Setup**(1^λ): Set the ring degree N , the special modulus P and the ciphertext modulus Q , the key distribution χ over R , and the error parameter σ . Choose a gadget decomposition $h : R_Q \rightarrow R^\ell$ with a gadget vector $\mathbf{g} \in R_Q^\ell$. Output the parameter set $\mathbf{pp} = (N, P, Q, \chi, \sigma, h, \mathbf{g})$.
- **CKKS.KeyGen**(\mathbf{pp}): Sample $s \leftarrow \chi$, $a \xleftarrow{\$} R_Q$ and $e \leftarrow D_\sigma$. Set the secret and public keys as $\mathbf{sk} = s$ and $\mathbf{pk} = (b, a) \in R_Q^2$ where $b = -s \cdot a + e \pmod{Q}$. Sample $\mathbf{k}_1 \xleftarrow{\$} R_Q^\ell$ and $\mathbf{e} \leftarrow D_\sigma^\ell$, and set the evaluation key as $\mathbf{evk} = (\mathbf{k}_0, \mathbf{k}_1) \in R_Q^{\ell \times 2}$ where $\mathbf{k}_0 = -s \cdot \mathbf{k}_1 + \mathbf{e} + s^2 \cdot \mathbf{g} \pmod{Q}$. Return \mathbf{sk}, \mathbf{pk} and \mathbf{evk} .
- **CKKS.Encode**($\Delta; \mathbf{m}$): Given a scaling factor Δ and a message vector $\mathbf{m} = (m_1, \dots, m_{N/2}) \in \mathbb{C}^{N/2}$, return a plaintext $\mu = \lfloor \Delta \cdot p \rfloor \in R$ where $p \in \mathbb{R}[X]/(X^N + 1)$ is a real polynomial such that $p(\zeta_i) = m_i$ and $p(\bar{\zeta}_i) = \bar{m}_i$ for $1 \leq i \leq N/2$.
- **CKKS.Decode**($\Delta; \mu$): Given $\mu \in R$, return $\mathbf{m} = \left(\frac{\mu(\zeta_1)}{\Delta}, \dots, \frac{\mu(\zeta_{N/2})}{\Delta} \right)$.
- **CKKS.Enc**($\mathbf{pk}; \mu$): Sample $w \leftarrow \chi$ and $e_0, e_1 \leftarrow D_\sigma$. Given an encoding $\mu \in R$, output the ciphertext $\mathbf{ct} = w \cdot \mathbf{pk} + (\mu + e_0, e_1) \pmod{Q}$.

- **CKKS.Add**(ct, ct'): Given two ciphertexts $\text{ct}, \text{ct}' \in R_Q^2$, output $\text{ct}_{add} = \text{ct} + \text{ct}' \pmod{Q}$.
- **CKKS.Mult**($\text{evk}; \text{ct}, \text{ct}'$): Given two ciphertexts $\text{ct} = (c_0, c_1), \text{ct}' = (c'_0, c'_1) \in R_Q^2$ and the relinearization key $\text{evk} \in R_{PQ}^{\ell}$, let (d_0, d_1, d_2) such that $d_0 = c_0 c'_0 \pmod{Q}$, $d_1 = c_0 c'_1 + c'_0 c_1 \pmod{Q}$, and $d_2 = c_1 c'_1 \pmod{Q}$. Output the ciphertext

$$\text{ct}_{mul} = (d_0, d_1) + (\lfloor \langle h(d_2), \mathbf{k}_0 \rangle / P \rfloor, \lfloor \langle h(d_2), \mathbf{k}_1 \rangle / P \rfloor) \pmod{Q}.$$

- **CKKS.Rescale**($\Delta; \text{ct}$): Given a ciphertext $\text{ct} = (c_0, c_1) \in R_Q^2$, return $\text{ct}' = (\lfloor \Delta^{-1} c_0 \rfloor, \lfloor \Delta^{-1} c_1 \rfloor) \in R_{Q'}^2$, where $Q' = Q/\Delta$.
- **CKKS.Dec**($\text{sk}; \text{ct}$): Given a ciphertext $\text{ct} = (c_0, c_1) \in R_Q^2$ and associated secret key $\text{sk} = s$, return $\mu = c_0 + c_1 \cdot s \pmod{Q}$.

For simplicity, we assume that the public key also includes the evaluation key throughout the paper. Throughout the paper, we denote q_{out} as an ciphertext modulus of the output ciphertext.

Correctness. The correctness of approximate HE schemes such as CKKS, can be defined as that the error in the evaluated ciphertext can be efficiently bounded depending on the evaluation circuit and the bounds of the input messages. We can formalize it as follows.

Definition 2 (Approximate Correctness). *Let Π be an HE scheme with a message space \mathcal{M} and \mathcal{L} be a space of circuits. For a subset of circuits with k input wires $\mathcal{L}_k \subseteq \mathcal{L}$, let $\text{Estimate} : \cup_{k \in \mathbb{N}} \mathcal{L}_k \times \mathbb{R}_{\geq 0}^k \rightarrow \mathbb{R}_{\geq 0}$ be an efficiently computable function, denoted as an error estimator of Π . We say that $(\Pi, \text{Estimate})$ satisfies approximate correctness if for all $k \in \mathbb{N}$, for all $\mathcal{C} \in \mathcal{L}_k$, for all $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$, if ct_i is an encryption of the message m_i such that $\|m_i\|_\infty \leq B_i$ (for $i \leq k$), then*

$$\|\text{Dec}_{\text{sk}}(\text{Eval}_{\text{pk}}(\mathcal{C}, \text{ct}_1, \dots, \text{ct}_k)) - \mathcal{C}(m_1, \dots, m_k)\|_\infty \leq \text{Estimate}(\mathcal{C}, \{B_i\}_{i \leq k}).$$

Security. We first recall the standard security notion of *indistinguishability under chosen plaintext attack* (IND-CPA) for HE schemes. The IND-CPA security implies that an adversary cannot distinguish between encryptions of different messages, even after seeing encryptions of chosen plaintexts.

Definition 3 (IND-CPA Security). *Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ be an HE scheme. The IND-CPA game is defined as an indistinguishability game parameterized by distribution ensembles $\{E_\theta^b\}_\theta$ for $b = 0$ or 1 , as described in Alg. 1. We say that the HE scheme Π is IND-CPA secure, if any probabilistic polynomial-time adversary \mathcal{A} has a negligible advantage $\text{Adv}^{\text{IND-CPA}}(\mathcal{A})$ against IND-CPA security defined as:*

$$|2 \cdot \Pr[b = b' | b \xleftarrow{\$} \{0, 1\}, b' \leftarrow \mathcal{A}(\text{pk}, \text{Enc}(b))] - 1.$$

In the context of HE, IND-CPA security is typically guaranteed by the hardness of the LWE [36] and Ring-LWE [32] problems. Li and Micciancio [27] demonstrated that the traditional formulation of IND-CPA security does not adequately capture the security of approximate encryption schemes, including the CKKS scheme, especially in scenarios where the decryption results are shared with external parties. To address this, they introduced a new security model called IND-CPA^D security, which provides the adversary \mathcal{A} with access to a decryption oracle that decrypts ciphertexts derived from encryption and evaluation of messages of \mathcal{A} ' choice. This is formally defined by granting the adversary access to three types of oracles: encryption (E_{pk}^b), evaluation (H_{evk}^b), and decryption (D_{pk}^b), as described in Alg. 1. Note that decryption queries are restricted on ciphertexts generated by the encryption and evaluation oracle, ensuring that only properly formed ciphertexts are decrypted. The definition of the IND-CPA^D security is described in Def. 4.

Definition 4 (IND-CPA^D Security [27]). *Let $\Pi = (\text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ be an HE scheme. The IND-CPA^D game is defined as an indistinguishability game parameterized by distribution ensembles $\{E_\theta^b, H_\theta^b, D_\theta^b\}_\theta$ for $b = 0$ or 1 of Alg. 1. We say that the HE scheme Π is IND-CPA^D secure, if any probabilistic polynomial-time adversary \mathcal{A} has a negligible advantage $\text{Adv}^{\text{IND-CPA}^{\text{D}}}(\mathcal{A})$ against IND-CPA^D security defined as:*

$$|2 \cdot \Pr[b = b' | b \xleftarrow{\$} \{0, 1\}, b' \leftarrow \mathcal{A}(\text{pk}, \text{Enc}(b))] - 1.$$

Algorithm 1 Oracles for the IND-CPA^D game.

<p>Initialize $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$</p> <p>global state $S \leftarrow \emptyset$ $i \leftarrow 0$</p> <p>Encryption oracle $E_{\text{pk}}^b(m_0, m_1) :=$ $\text{ct} \leftarrow \text{Enc}_{\text{pk}}(m_b)$ $S[i] \leftarrow (m_0, m_1, \text{ct})$ $i \leftarrow i + 1$ return ct, i</p>	<p>Evaluation oracle $H_{\text{evk}}^b(g, \mathbf{J} = (j_1, \dots, j_k), S) :=$ $\text{ct} \leftarrow \text{Eval}_{\text{evk}}(g, S[j_1].\text{ct}, \dots, S[j_k].\text{ct})$ $g_0 \leftarrow g(S[j_1].m_0, \dots, S[j_k].m_0)$ $g_1 \leftarrow g(S[j_1].m_1, \dots, S[j_k].m_1)$ $S[i] \leftarrow (g_0, g_1, \text{ct})$ $i \leftarrow i + 1$ return ct, i</p> <p>Decryption oracle $D_{\text{sk}}^b(j) :=$ if $S[j].m_0 = S[j].m_1$ return $\text{Dec}_{\text{sk}}(S[j].\text{ct})$ else return \perp</p>
--	--

Recent studies have explored the notion of IND-CPA^D security. Li et al. [28] present a method for transforming an IND-CPA secure approximate HE scheme into an HE scheme that achieves IND-CPA^D security. Similar to the noise smudging technique, they propose adding a specific amount of noise, tailored to the worst-case error growth during homomorphic computation. Guo et al. [17] introduce new key-recovery attacks on approximate HE schemes that employ noise-smudging countermeasures based on non-worst-case noise estimation. Recently, Cheon et al. [7] extend key-recovery attacks beyond approximate HE schemes to exact HE schemes. Their attack is based on the imperfect correctness of decryption in real-world implementations of exact HE schemes.

Threshold CKKS. While the standard (single-key) CKKS scheme supports homomorphic operations only between ciphertexts encrypted under the same key, several variants of CKKS for multiple parties have emerged, such as threshold CKKS [1, 33, 34], multi-key CKKS [6, 35], and multi-group CKKS [24].

As a typical example, we describe a threshold CKKS scheme, especially n -out-of- n threshold scheme, for the simplicity. In (n -out-of- n) threshold CKKS, multiple parties collaboratively generate a shared public key and receive additive share of the shared secret key as illustrated in Fig. 1. Homomorphic operations are performed in the same manner as in the single-key CKKS scheme using the shared public key while a decryption is carried out in a distributed manner. By incorporating techniques such as Shamir’s secret sharing or short secret sharing, the scheme can employ a t -out-of- n access structure, allowing any subset of at least t parties to reconstruct the secret key and decrypt the ciphertext.

Functionality $\mathcal{F}_{\text{ThKeyGen}}$
<p>Setup: Let pp be a public parameter for CKKS.</p> <hr/> <ol style="list-style-type: none"> 1. Generate $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$. 2. For $1 \leq i \leq n - 1$, sample $\text{sk}_i \xleftarrow{\\$} R_q$. 3. Let $\text{sk}_n := \text{sk} - \sum_{i=1}^{n-1} \text{sk}_i \pmod{q}$. 4. Output (sk_i, pk) to P_i for $1 \leq i \leq n$.

Fig. 1. Functionality of distributed (threshold) key generation

3 Secure Multi-party Computation of Approximations

Secure MPC protocol enables multiple parties to jointly evaluate a computational circuit over their respective inputs while ensuring that no information about individual inputs is revealed to any other party.

These protocols have been widely applied in various domains, including privacy-preserving federated learning [13, 18, 30]. In many such applications, approximate arithmetic is often employed instead of exact arithmetic over discrete data. As a result, the development of secure protocols for approximate computation in multi-party settings has emerged as a significant research challenge. To formally define the security of these MPC protocols, the simulation-based security model ensures that any information a corrupted party obtains during protocol execution can be efficiently simulated in an ideal model, wherein a trusted third party performs the computation. Consequently, this security guarantee ensures that an adversary gains no additional advantage beyond what would be possible in the ideal setting. The work of [12] not only formalized the standard definition of general MPC, as established in many studies [16, 29], but also introduced a more relaxed security definition tailored for approximate secure computation. In the following sections, we present these definitions in detail and highlight the distinctions between them.

3.1 Standard Definition

We designate some notations to define a secure MPC. Let n be a number of parties and let $f : (\{0, 1\}^*)^n \rightarrow (\{0, 1\}^*)^n$ be a (n -ary) probabilistic polynomial-time functionality. We denote $\mathbf{x} = (x_1, \dots, x_n)$ and $f(\mathbf{x}) = (f_1, \dots, f_n)$, where x_i and f_i correspond to the party P_i . Although a single party may have multiple inputs, throughout the paper, we assume that each party P_i has only one input x_i for simplicity. For a MPC protocol Π to compute the functionality f on input \mathbf{x} , the view of P_i is denoted by $\text{view}_i^\Pi(\mathbf{x})$ and defined as the tuple of its input, randomness, and all messages received from other parties. We denote the output of P_i from the execution of Π on input \mathbf{x} by $\text{output}_i^\Pi(\mathbf{x})$, and $\text{output}^\Pi(\mathbf{x}) = (\text{output}_1^\Pi(\mathbf{x}), \dots, \text{output}_n^\Pi(\mathbf{x}))$. We assume that the protocol proceeds in synchronous rounds, where in each round, each party sends a message based on its input, random input, and previously received messages. At the end of each round, parties may terminate and output some value based on their entire view, including their input, random input, and received messages. The definition of the secure MPC is as follows.

Definition 5 (Secure Multi-party Computation [16]). *Let Π be an n -party protocol for computing a functionality $f = (f_1, \dots, f_n)$. For a set of indices $I \subseteq \{1, \dots, n\}$, we denote $f_I(\mathbf{x}) = (f_i(\mathbf{x}))_{i \in I}$ and $\text{view}_I^\Pi(\mathbf{x}) = (\text{view}_i^\Pi(\mathbf{x}))_{i \in I}$. We say that Π securely computes f under the presence of semi-honest adversaries if there exists probabilistic polynomial-time algorithms Sim^Π such that for every $I \subseteq \{1, \dots, n\}$ and \mathbf{x} , the following holds.*

$$\{\text{Sim}^\Pi(I, (x_i)_{i \in I}, f_I(\mathbf{x})), f(\mathbf{x})\} \approx_c \{\text{view}_I^\Pi(\mathbf{x}), \text{output}^\Pi(\mathbf{x})\}.$$

The above privacy requirement asserts that the *joint distribution* of the simulator's output and the functionality's output should be indistinguishable from the view of the corrupted party and the outputs of parties. This requirement is particularly meaningful when the functionality is probabilistic, as it guarantees that the adversary does not obtain any additional information about the output.

3.2 Liberal Definition

Freigenbaum et al. [12] introduces a liberal security definition for secure approximations by extending the standard definition of secure computation. To formulate the definition of secure approximation, they first propose the notion of *functional privacy* for a target (deterministic) function. Informally, we say that \hat{f} is a (possibly randomized) functionally private approximation of f if the output of \hat{f} does not reveal more information about its input than f . To be precise, \hat{f} is functionally private with respect to f if there exists a probabilistic polynomial-time sampling algorithm \mathcal{S} such that the distribution $\mathcal{S}(f(\mathbf{x}))$ is indistinguishable from $\hat{f}(\mathbf{x})$ for every input \mathbf{x} . According to [12], a protocol for f is a secure approximation protocol if it securely computes \hat{f} , where \hat{f} is functionally private with respect to f .

Intuitively, their liberal definition of secure approximations ensures that no additional information is revealed during computation. This definition requires not only that computing the approximate output does not reveal more about other party's inputs and outputs than the approximate output but also that the approximate output does not reveal more information about other party's inputs and outputs than the

exact output does. More formally, there exist probabilistic polynomial-time algorithms that can simulate the joint distribution of the corrupted party’s view and the uncorrupted party’s output. Although this definition is formulated for the special case of 2PC, it can be simply extended to the multi-party setting, as described in Def. 6.

Definition 6 (Secure Multi-party Approximation: Liberal Definition [12]). *Let f be a deterministic functionality and $\hat{f} = (\hat{f}_1, \dots, \hat{f}_n)$ be a functionally private approximation of $f = (f_1, \dots, f_n)$. Let \mathcal{X} be the set of all valid inputs $\mathbf{x} = (x_1, \dots, x_n)$. For a set of indices $I \subseteq \{1, \dots, n\}$, we denote $\hat{f}_I(\mathbf{x}) = (\hat{f}_i(\mathbf{x}))_{i \in I}$ and $\text{view}_I^\Pi(\mathbf{x}) = (\text{view}_i^\Pi(\mathbf{x}))_{i \in I}$. We say that a protocol Π securely computes \hat{f} under the presence of semi-honest adversaries if there exist probabilistic polynomial-time algorithms Sim^Π such that for every $I \subseteq \{1, \dots, n\}$ and \mathbf{x} , the following holds.*

$$\{(\text{Sim}^\Pi(I, (x_i)_{i \in I}, f_I(\mathbf{x}), \hat{f}_I(\mathbf{x})), \hat{f}_J(\mathbf{x}))\} \approx_c \{(\text{view}_I^\Pi(\mathbf{x}), \text{output}_J^\Pi(\mathbf{x}))\},$$

where $J = \{1, \dots, n\} \setminus I$, $\hat{f}_J(\mathbf{x}) = (\hat{f}_j(\mathbf{x}))_{j \in J}$, and $\text{output}_J^\Pi(\mathbf{x}) = (\text{output}_j^\Pi(\mathbf{x}))_{j \in J}$.

Unlike the standard security definition, the liberal security definition allows the adversary in the ideal world to interact with a trusted third party that computes the exact value of the function f . In contrast, the standard definition restricts the adversary to obtaining only a functionally private approximation \hat{f} . This distinction implies that, under the liberal security definition, the adversary cannot gain any additional information beyond the exact deterministic output of f . For a more detailed discussion, we refer to [12].

4 Secure Approximate Two-party Computation from CKKS

In recent years, HE has emerged as one of the most promising technologies for enabling secure computation, thanks to its conceptual simplicity, versatility, and significant recent advancements in performance. In particular, extensive research has focused on designing secure MPC protocols using exact HE techniques (e.g., [10, 19, 31, 33, 35]).

On the other hand, the CKKS scheme is an HE scheme specifically designed for efficient approximate computations and has been widely adopted in various applications, including privacy-preserving machine learning. However, CKKS possesses a unique characteristic due to its inherent design, in which the noise introduced by the scheme is treated as part of the plaintext. Unfortunately, this property makes it significantly more challenging to accurately define functionalities and introduces security concerns that do not arise in other HE schemes. Consequently, only a few studies (e.g., [13, 25]) have explored the construction of MPC protocols for approximate computation. Moreover, to the best of our knowledge, all existing works either fail to explicitly define the functionality or lack formal security proofs.

From a more technical perspective, when constructing MPC protocols using traditional exact HE schemes [4, 5], various techniques, such as noise smudging, are employed to enhance security. However, it remains unclear whether these techniques can be directly applied to protocols based on CKKS. Furthermore, security-enhancing techniques may change plaintext values, ultimately affecting the functionality of the protocol. As a result, CKKS-based protocols face the unique challenge of ensuring both correctness and security concerns simultaneously.

The ultimate goal of this work is to classify CKKS-based multi-party protocols, formally describe their functionalities and study their security guarantees. In this section, we begin by exploring a primary application scenario of HE in constructing asymmetric 2PC protocols between a client and a server, where the client acts as the key owner and the server as the evaluator.

In Section 4.1, we focus on the simplest case in which only the client has a private input. Sections 4.2 and 4.3 will cover a more general scenario, each considering liberal and standard security notions, respectively.

4.1 Construction of Simple 2PC of Approximations

We begin with a simple scenario in which only the client has private input x , while the server behaves as an evaluator of a public circuit C . At the end of the protocol, the client obtains an approximate value of $C(x)$, while the server learns nothing. Or, the client may share the result with the server so that both parties have the same output. The ideal functionality is described in Fig. 2.

Functionality $\mathcal{F}_{\text{SimpApp2PC}}$ (or $\mathcal{F}'_{\text{SimpApp2PC}}$)
<p>Setup: Let C be an evaluation circuit and \mathcal{D} be an error distribution. The functionality receives x from P_1 (the client) but nothing from P_2 (the server).</p>
<ol style="list-style-type: none"> 1. Compute $z := C(x)$. 2. Sample $e \leftarrow \mathcal{D}$ and let $\hat{z} := z + e$. 3. Output \hat{z} to both P_1 and P_2 (or only to P_1).

Fig. 2. Functionality of simple 2PC of approximations

Our construction for the simple 2PC case has limited practical applicability to real-world scenarios, as the client can perform the computation on their own data in plaintext. Nevertheless, we note that it still has practical applications, such as Private information retrieval (PIR).

In the following, we demonstrate a folklore that the functionality $\mathcal{F}_{\text{SimpApp2PC}}$ (or $\mathcal{F}'_{\text{SimpApp2PC}}$) can be realized from an approximate HE such as CKKS for sufficiently large error distribution \mathcal{D} . To be precise, the standard HE-based 2PC protocol, where the client and server act as a key owner and an evaluator, respectively, can securely realize the desired functionality using the noise smudging technique. Fig. 3 illustrates the protocol $\Pi_{\text{SimpApp2PC}}$ (or $\Pi'_{\text{SimpApp2PC}}$) to achieve the desired functionality.

Protocol $\Pi_{\text{SimpApp2PC}}$ (or $\Pi'_{\text{SimpApp2PC}}$)
<ol style="list-style-type: none"> 1. P_1 generates a key pair $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)$. Then P_1 sends $(\text{pk}, \text{ct}_{\text{in}})$ to P_2. 2. P_2 computes $\text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}})$ and sends ct_{out} to P_1. 3. P_1 decrypts $z' := \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}})$, samples $e \leftarrow \mathcal{D}$, and computes $\hat{z} := z' + e$. 4. In $\Pi_{\text{SimpApp2PC}}$, P_1 sends \hat{z} to P_2, and both parties output \hat{z}. In $\Pi'_{\text{SimpApp2PC}}$, P_1 outputs \hat{z} while P_2 outputs nothing.

Fig. 3. Simple 2PC protocols of approximations

Theorem 1. *Let $\text{CKKS} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ be a CKKS scheme with respect to an error estimator Estimate . For a plaintext x and a circuit C , suppose that $\|x\|_\infty \leq B$ for some $B > 0$ and let $E := \text{Estimate}(C, B)$. If CKKS is IND-CPA secure and \mathcal{D} is an E -smudging distribution, then the 2PC protocol $\Pi_{\text{SimpApp2PC}}$ (or $\Pi'_{\text{SimpApp2PC}}$) securely realizes the functionality $\mathcal{F}_{\text{SimpApp2PC}}$ (or $\mathcal{F}'_{\text{SimpApp2PC}}$, respectively).*

Proof. Let us first show the correctness of $\Pi'_{\text{SimpApp2PC}}$. Let $z := C(x)$ and let $z' := \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}})$ be the decryption of the resulting ciphertext. By the definition of the function Estimate , we have $z' = z + e_{\text{out}}$ for some $e_{\text{out}} \in R$ such that $\|e_{\text{out}}\|_\infty \leq E$. Thus, the distribution of the output $\hat{z} = z' + e = z + (e_{\text{out}} + e)$ with $e \leftarrow \mathcal{D}$ is statistically indistinguishable from the distribution of the output $z + e$ in the functionality $\mathcal{F}'_{\text{SimpApp2PC}}$, since \mathcal{D} is an E -smudging distribution. The correctness of $\Pi_{\text{SimpApp2PC}}$ can be shown in a similar way.

To prove the security of $\Pi'_{\text{SimpApp2PC}}$, we first consider a corrupted client P_1 . The simulator $\text{Sim}^1_{\text{SimpApp2PC}}(x, \hat{z})$ for P_1 's view is illustrated in Fig. 4. It follows the real protocol $\Pi'_{\text{SimpApp2PC}}$ up to the point where it computes z' by decrypting ct_{out} , but the only difference is that the smudging error is computed as $e = \hat{z} - z'$, instead of sampling it from \mathcal{D} .

Simulator $\text{Sim}^1_{\text{SimpApp2PC}}(x, \hat{z})$
<ol style="list-style-type: none"> 1. Generate $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)$. 2. Compute $\text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}})$. 3. Compute $z' := \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}})$ and $e := \hat{z} - z'$. 4. Output $(x, \text{sk}, \text{pk}, \text{ct}_{\text{in}}, \text{ct}_{\text{out}}, e)$.

Fig. 4. Simulator for an adversarial client in protocol $\Pi'_{\text{SimpApp2PC}}$

As mentioned in the correctness proof, the decrypted result $z' = \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}})$ in the real protocol is $z' = z + e_{\text{out}}$ for some $\|e_{\text{out}}\|_\infty \leq E$, and the output can be written as $\hat{z} = z' + e = z + e_{\text{out}} + e$. Therefore, the simulator $\text{Sim}^1_{\text{SimpApp2PC}}(x, \hat{z})$ satisfies the desired condition under the assumption that \mathcal{D} is an E -smudging distribution:

$$\begin{aligned}
& \{(\text{view}_1^{\Pi'_{\text{SimpApp2PC}}}(x), \text{output}^{\Pi'_{\text{SimpApp2PC}}}(x))\} \\
& \equiv \{((x, \text{sk}, \text{pk}, \text{ct}_{\text{in}}, \text{ct}_{\text{out}}, e), (z' + e, \perp)) \mid e \leftarrow \mathcal{D}\} \\
& \approx_s \{((x, \text{sk}, \text{pk}, \text{ct}_{\text{in}}, \text{ct}_{\text{out}}, e - e_{\text{out}}), (z' + e - e_{\text{out}}, \perp)) \mid e_{\text{out}} := z' - z, e \leftarrow \mathcal{D}\} \\
& \equiv \{((x, \text{sk}, \text{pk}, \text{ct}_{\text{in}}, \text{ct}_{\text{out}}, (z + e) - z'), (z + e, \perp)) \mid e \leftarrow \mathcal{D}\} \\
& \equiv \{(\text{Sim}^1_{\text{SimpApp2PC}}(x, \hat{z}), (\hat{z}, \perp)) \mid (\hat{z}, \perp) \leftarrow \mathcal{F}'_{\text{SimpApp2PC}}(x)\},
\end{aligned}$$

where $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$, $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)$, $\text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}})$, $z = C(x)$, and $z' = \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}})$. The same simulator can be used to prove the security of $\Pi_{\text{SimpApp2PC}}$ since P_1 's view is identical in both protocols.

Next, we construct the simulators $\text{Sim}^2_{\text{SimpApp2PC}}(\hat{z})$ and $\text{Sim}^{2'}_{\text{SimpApp2PC}}(\perp)$ to simulate P_2 's view in $\Pi_{\text{SimpApp2PC}}$ and $\Pi'_{\text{SimpApp2PC}}$, respectively. $\text{Sim}^{2'}_{\text{SimpApp2PC}}(\perp)$ is described in Fig. 5. Then, we have:

$$\begin{aligned}
& \{(\text{view}_2^{\Pi'_{\text{SimpApp2PC}}}(x), \text{output}^{\Pi'_{\text{SimpApp2PC}}}(x))\} \\
& \equiv \{((\text{pk}, \text{ct}_{\text{in}}), (z' + e, \perp)) \mid \text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x), \text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}), z' := \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}}), e \leftarrow \mathcal{D}\} \\
& \approx_s \{((\text{pk}, \text{ct}_{\text{in}}), (z + e, \perp)) \mid \text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x), z := C(x), e \leftarrow \mathcal{D}\} \\
& \approx_c \{((\text{pk}, \text{ct}_{\text{in}}), (z + e, \perp)) \mid \text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(0), z := C(x), e \leftarrow \mathcal{D}\} \\
& \equiv \{(\text{Sim}^{2'}_{\text{SimpApp2PC}}(\perp), (\hat{z}, \perp)) \mid (\hat{z}, \perp) \leftarrow \mathcal{F}'_{\text{SimpApp2PC}}(x)\},
\end{aligned}$$

as desired. The first statistical indistinguishability follows from the assumption that \mathcal{D} is an E -smudging distribution, while the second computational indistinguishability follows from the IND-CPA security of the CKKS scheme. In the protocol $\Pi_{\text{SimpApp2PC}}$, we define $\text{Sim}^2_{\text{SimpApp2PC}}(\hat{z})$ similarly to $\text{Sim}^{2'}_{\text{SimpApp2PC}}(\perp)$, except that it appends \hat{z} to its simulated view. Then, it can be shown similarly that $\{(\text{view}_2^{\Pi_{\text{SimpApp2PC}}}(x), \text{output}^{\Pi_{\text{SimpApp2PC}}}(x))\}$ is computationally indistinguishable from $\{(\text{Sim}^2_{\text{SimpApp2PC}}(\hat{z}), \hat{z}) \mid \hat{z} \leftarrow \mathcal{F}_{\text{SimpApp2PC}}(x)\}$. \square

4.2 Construction of General 2PC of Approximations: Liberal Security

From this subsection, we further study the general functionality that approximately computes a circuit over private inputs of both parties, which is more theoretically significant and practically useful. Although the protocol $\Pi_{\text{SimpApp2PC}}$ securely computes the evaluation circuit C , its practical applicability is limited

Simulator $\text{Sim}_{\text{SimpApp2PC}}^{2'}(\perp)$
<ol style="list-style-type: none"> 1. Generate $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(0)$. 2. Output $(\text{pk}, \text{ct}_{\text{in}})$.

Fig. 5. Simulator for an adversarial server in protocol $\Pi_{\text{SimpApp2PC}}$

in real-world scenarios. In particular, if the client’s input is private but they are capable of local computation, they may prefer to evaluate the circuit on their own data in plaintext rather than outsourcing the computation to the server. To address this scenario, we consider a more general setting where both the client and the server provide their own inputs. In this setting, the server’s input is typically processed in plaintext for computational efficiency. Consequently, the resulting ciphertext depends on the client’s input ciphertext, the evaluation circuit, and the server’s plaintext input. The client can either keep its output value \hat{z} to itself or send it to the server.

Insecure example protocols. We first examine some naive approaches (Fig. 6) and explain why they fail to satisfy the security requirements. The two protocols, Π_1 and Π_2 , follow almost the same pipeline, and we note that they provide identical functionality of approximately computing $C(x, y)$ with an error sampled from a smudging distribution \mathcal{D} .

Protocol Π_1
<ol style="list-style-type: none"> 1. P_1 generates a key pair $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and encrypts $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)$. P_1 sends $(\text{pk}, \text{ct}_{\text{in}})$ to P_2. 2. P_2 computes $\text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}, y)$, and sends it to P_1. 3. P_1 decrypts $z' := \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}})$, samples $e \leftarrow \mathcal{D}$, and computes $\hat{z} := z' + e$. 4. P_1 sends \hat{z} to P_2, and both parties output \hat{z}.

Protocol Π_2
<ol style="list-style-type: none"> 1. P_1 generates a key pair $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and encrypts $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)$. P_1 sends $(\text{pk}, \text{ct}_{\text{in}})$ to P_2. 2. P_2 computes $\text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}, y)$, $\text{ct}_{\text{zero}} \leftarrow \text{Enc}_{\text{pk}}(0)$, and samples $e \leftarrow \mathcal{D}$. P_2 computes $\text{ct}'_{\text{out}} := \text{ct}_{\text{out}} + \text{ct}_{\text{zero}} + (e, 0)$, and sends it to P_1. 3. P_1 computes $\hat{z} := \text{Dec}_{\text{sk}}(\text{ct}'_{\text{out}})$ and sends it to P_2. Both parties output \hat{z}.

Fig. 6. Insecure approximate 2PC protocols

Unfortunately, these candidate protocols are insecure (even with respect to the liberal security notion) for different reasons. In the first example Π_1 , a smudging error e is sampled and added to the decrypted value by P_1 . This approach may enhance the security of the protocol against an adversarial server because the final output \hat{z} leaks no information about the approximate error $z' - z$ derived from CKKS due to the smudging error e . We remark that this technique has been introduced as a mitigation strategy against some attacks on the IND-CPA^D security of CKKS [28]. However, Π_1 is insecure as the privacy of P_2 ’s input y is not guaranteed. That is, P_1 might be able to learn much information about y (beyond the desired functionality) from the output ciphertext ct_{out} .

On the other hand, there have been several studies on circuit privacy or sanitization (e.g., [2, 11, 14, 23]) of HE schemes to achieve this privacy requirement for the evaluator. A simple solution is to randomize the resulting ciphertext after homomorphic evaluation (for example, by adding an encryption of zero with a large error as in Π_2), thereby washing out any unnecessary information except the underlying plaintext. However, Π_2 is insecure since it is vulnerable to a key-recovery attack of [27], that is, P_2 can easily compute the secret key sk from the ciphertext ct'_{out} and its decryption \hat{z} .

Our construction (liberal security). We observed that each security-enhancing technique provides only one-sided security rather than ensuring the security of the entire protocol. To address the security vulnerabilities in previous protocols, we propose a new approach that incorporates both noise smudging and ciphertext randomization techniques. Below, we present an ideal functionality $\mathcal{F}_{\text{LibApp2PC}}$ (Fig. 7) and a concrete protocol $\Pi_{\text{LibApp2PC}}$ to realize the functionality (Fig. 8). In the following, we show the correctness and liberal security of this protocol.

Functionality $\mathcal{F}_{\text{LibApp2PC}}$
<p>Setup: Let C be an evaluation circuit and \mathcal{D} be an error distribution. The functionality receives x from P_1 (the client) and y from P_2 (the server).</p> <hr/> <ol style="list-style-type: none"> 1. Compute $z := C(x, y)$. 2. Sample $e_1, e_2 \leftarrow \mathcal{D}$ and let $\hat{z} := z + e_1 + e_2$. 3. Output \hat{z} to both P_1 and P_2.

Fig. 7. Functionality of a general approximate 2PC protocol for liberal security

Protocol $\Pi_{\text{LibApp2PC}}$
<ol style="list-style-type: none"> 1. P_1 generates a key pair $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and encrypts $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)$. P_1 sends $(\text{pk}, \text{ct}_{\text{in}})$ to P_2. 2. P_2 computes $\text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}, y)$, $\text{ct}_{\text{zero}} \leftarrow \text{Enc}_{\text{pk}}(0)$, and samples $e_2 \leftarrow \mathcal{D}$. P_2 computes $\text{ct}'_{\text{out}} := \text{ct}_{\text{out}} + \text{ct}_{\text{zero}} + (e_2, 0)$, and sends it to P_1. 3. P_1 samples $e_1 \leftarrow \mathcal{D}$ and computes $\hat{z} := \text{Dec}_{\text{sk}}(\text{ct}'_{\text{out}}) + e_1$. P_1 sends \hat{z} to P_2, and both parties output \hat{z}.

Fig. 8. Our approximate 2PC protocol for liberal security

Theorem 2. Let $\text{CKKS} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ be a CKKS scheme with respect to an error estimator Estimate . For plaintexts x, y and a circuit C , suppose that $\|x\|_\infty \leq B_1$ and $\|y\|_\infty \leq B_2$ for some $B_1, B_2 > 0$. Let $E_0 > 0$ be an upper bound of public-key encryption error, i.e., $\Pr[\|\text{Dec}_{\text{sk}}(\text{Enc}_{\text{pk}}(0))\|_\infty \leq B_0]$ with an overwhelming probability. Let $E := \text{Estimate}(C, B_1, B_2) + E_0$. If CKKS is IND-CPA secure and \mathcal{D} is an E -smudging distribution, then the 2PC protocol $\Pi_{\text{LibApp2PC}}$ securely realizes the functionality $\mathcal{F}_{\text{LibApp2PC}}$ in the liberal sense.

Proof. We first show the correctness of $\Pi_{\text{LibApp2PC}}$. Let $z := C(x, y)$ and denote $z' := \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}} + \text{ct}_{\text{zero}})$ in the protocol. From the error estimator Estimate , we have $z' = z + e_{\text{out}}$ for some $e_{\text{out}} \in R$ such that $\|e_{\text{out}}\|_\infty \leq \text{Estimate}(C, B_1, B_2) + E_0 = E$. Then, the output \hat{z} can be written as:

$$\hat{z} = \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}} + \text{ct}_{\text{zero}}) + e_1 + e_2 = z + e_{\text{out}} + e_1 + e_2,$$

for some $e_1, e_2 \leftarrow \mathcal{D}$. Thus, the distribution of this output \hat{z} is statistically indistinguishable from the distribution of the output $z + e_1 + e_2$ of the functionality $\mathcal{F}_{\text{GenApp2PC}}$, since \mathcal{D} is an E -smudging distribution.

To prove the liberal security of $\Pi_{\text{LibApp2PC}}$, we first consider a corrupted client P_1 . The simulator $\text{Sim}_{\text{LibApp2PC}}^1(x, z, \hat{z})$ for P_1 's view is illustrated in Fig. 9. A key idea is to split the total error $e = \hat{z} - z$ into two separate error terms e_1 and e_2 by sampling them from the conditional distribution of $\mathcal{D} \times \mathcal{D}$

Simulator $\text{Sim}_{\text{LibApp2PC}}^1(x, z, \hat{z})$
<ol style="list-style-type: none"> 1. Generate $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)$. 2. Let $e := \hat{z} - z$ and sample $(e_1, e_2) \leftarrow \{(e_1, e_2) \leftarrow \mathcal{D} \times \mathcal{D} \mid e_1 + e_2 = e\}$. 3. Sample $a_{\text{out}} \xleftarrow{\\$} R_{q_{\text{out}}}$ and let $\text{ct}'_{\text{out}} := (b_{\text{out}} := a_{\text{out}}s + z + e_2, a_{\text{out}})$. 4. Output $(x, \text{sk}, \text{pk}, \text{ct}_{\text{in}}, \text{ct}'_{\text{out}}, e_1)$.

Fig. 9. Simulator for an adversarial client in protocol $\Pi_{\text{LibApp2PC}}$.

given $e_1 + e_2 = e$. Then, it holds that:

$$\begin{aligned}
 & \{(\text{view}_1^{\Pi_{\text{LibApp2PC}}}(x, y), \text{output}_2^{\Pi_{\text{LibApp2PC}}}(x, y))\} \\
 & \equiv \{(\dots, \text{ct}'_{\text{out}} = \text{ct}_{\text{out}} + \text{ct}_{\text{zero}} + (e_2, 0), e_1, z' + e_1 + e_2) \mid \\
 & \quad e_1, e_2 \leftarrow \mathcal{D}, \text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}, y), \text{ct}_{\text{zero}} \leftarrow \text{Enc}_{\text{pk}}(0), z' = \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}} + \text{ct}_{\text{zero}})\} \\
 & \approx_s \{(\dots, \text{ct}'_{\text{out}} = \text{ct}_{\text{out}} + \text{ct}_{\text{zero}} + (e_2 - e_{\text{out}}, 0), e_1, z' + e_1 + e_2 - e_{\text{out}}) \mid \\
 & \quad e_1, e_2 \leftarrow \mathcal{D}, \text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}, y), \text{ct}_{\text{zero}} \leftarrow \text{Enc}_{\text{pk}}(0), z' = \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}} + \text{ct}_{\text{zero}})\} \\
 & \equiv \{(\dots, \text{ct}'_{\text{out}} = ((a_{\text{out}} + a_{\text{zero}})s + z + e_1 + e_2, a_{\text{out}} + a_{\text{zero}}), e_1, z + e_1 + e_2) \mid \\
 & \quad e_1, e_2 \leftarrow \mathcal{D}, \text{ct}_{\text{out}} = (b_{\text{out}}, a_{\text{out}}) \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}, y), \text{ct}_{\text{zero}} = (b_{\text{zero}}, a_{\text{zero}}) \leftarrow \text{Enc}_{\text{pk}}(0)\} \\
 & \approx_c \{(\dots, \text{ct}'_{\text{out}} = (as + z + e_1 + e_2, a), e_1, z + e_1 + e_2) \mid e_1, e_2 \leftarrow \mathcal{D}, a \xleftarrow{\$} R_{q_{\text{out}}}\} \\
 & \equiv \{(\text{Sim}_{\text{LibApp2PC}}^1(x, z, \hat{z}), \hat{z}) \mid \hat{z} \leftarrow \mathcal{F}_{\text{LibApp2PC}}(x)\},
 \end{aligned}$$

where $\dots = (x, \text{sk}, \text{pk}, \text{ct}_{\text{in}})$ with $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)$ in all equations. The first statistical indistinguishability follows from the assumption that \mathcal{D} is an E -smudging distribution, while the second computational indistinguishability is derived from the fact that a_{zero} obtained from $\text{ct}_{\text{zero}} = (b_{\text{zero}}, a_{\text{zero}}) \leftarrow \text{Enc}_{\text{pk}}(0)$ is indistinguishable from a uniform distribution over $R_{q_{\text{out}}}$ under the RLWE assumption.

Simulator $\text{Sim}_{\text{LibApp2PC}}^2(y, z, \hat{z})$
<ol style="list-style-type: none"> 1. Generate $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(0)$. 2. Compute $\text{ct}_{\text{zero}} \leftarrow \text{Enc}_{\text{pk}}(0)$. 3. Compute $e := \hat{z} - z$ and samples $(e_1, e_2) \leftarrow \{(e_1, e_2) \leftarrow \mathcal{D} \times \mathcal{D} \mid e_1 + e_2 = e\}$. 4. Output $(y, \text{pk}, \text{ct}_{\text{in}}, \text{ct}_{\text{zero}}, e_2)$.

Fig. 10. Simulator for an adversarial server in protocol $\Pi_{\text{LibApp2PC}}$.

Next, we construct $\text{Sim}_{\text{LibApp2PC}}^2(y, z, \hat{z})$ that simulates P_2 's view in the protocol in Fig. 10. Similar to the previous case, it also splits the final error $e = \hat{z} - z$ into two factors e_1 and e_2 , and uses e_2 in the simulation. We have:

$$\begin{aligned}
 & \{\text{view}_2^{\Pi_{\text{LibApp2PC}}}(x, y), \text{output}_1^{\Pi_{\text{LibApp2PC}}}(x, y)\} \\
 & \equiv \{(y, \text{pk}, \text{ct}_{\text{in}}, \text{ct}_{\text{zero}}, e_2, \hat{z}) \mid e_1, e_2 \leftarrow \mathcal{D}, \text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x), \text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}, y), \\
 & \quad \text{ct}_{\text{zero}} \leftarrow \text{Enc}_{\text{pk}}(0), \text{ct}'_{\text{out}} = \text{ct}_{\text{out}} + \text{ct}_{\text{zero}} + (e_2, 0), z' := \text{Dec}_{\text{sk}}(\text{ct}'_{\text{out}}), \hat{z} = z' + e_1 + e_2\} \\
 & \approx_s \{(y, \text{pk}, \text{ct}_{\text{in}}, \text{ct}_{\text{zero}}, e_2, \hat{z}) \mid e_1, e_2 \leftarrow \mathcal{D}, \text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x), \text{ct}_{\text{zero}} \leftarrow \text{Enc}_{\text{pk}}(0), \hat{z} = z + e_1 + e_2\} \\
 & \approx_c \{(y, \text{pk}, \text{ct}_{\text{in}}, \text{ct}_{\text{zero}}, e_2, \hat{z}) \mid e_1, e_2 \leftarrow \mathcal{D}, \text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(0), \text{ct}_{\text{zero}} \leftarrow \text{Enc}_{\text{pk}}(0), \hat{z} = z + e_1 + e_2\} \\
 & \equiv \{(\text{Sim}_{\text{LibApp2PC}}^2(y, z, \hat{z}), \hat{z}) \mid \hat{z} \leftarrow \mathcal{F}_{\text{LibApp2PC}}(x)\}.
 \end{aligned}$$

The first statistical indistinguishability follows from the assumption that \mathcal{D} is E -smudging, while the subsequent computational indistinguishability is derived from the IND-CPA security of CKKS. \square

4.3 Construction of General 2PC of Approximations: Standard Security

Although the liberal definition provides a meaningful security guarantee in certain use cases, it remains necessary to construct a secure approximation protocol that satisfies the standard definition for several reasons. The standard definition of secure approximation is built on the standard notion of exact secure computation as a black-box, making it applicable to any security frameworks and applications. Furthermore, the liberal definition has a key limitation: an ideal-process adversary is allowed to interact with a trusted party that computes the exact value of the target function. In contrast, the standard definition provides the adversary only with an approximate result, so it can apply even when the exact computation of the target function is intractable - such as in approximate HE schemes. The standard definition also provides an additional advantage in the context of protocols based on approximate HE schemes, particularly in relation to ciphertext noise, which will be discussed later.

Insecurity of the previous protocol under the standard definition. In Section 4.2, we presented an approximate 2PC protocol $\Pi_{\text{LibApp2PC}}$ that is secure in the liberal manner. Unfortunately, this protocol does not achieve the standard security notion (Def. 5) since it is infeasible to construct valid simulators not receiving $z = C(x, y)$ as input.

For example, suppose that we aim to simulate the P_1 's view using only x and $\hat{z} \leftarrow \mathcal{F}_{\text{LibApp2PC}}$. Then, the simulator should be able to sample e_1 such that the distribution of (e_1, \hat{z}) is statistically indistinguishable from that of $(e_1, z + e_1 + e_2)$, where $e_1, e_2 \leftarrow \mathcal{D}$. If possible, the simulator may also compute $z + e_2$, which provides a better approximation of z than $\hat{z} = z + e_1 + e_2$. However, in general, it is impossible to derive a more precise approximation of z from a less accurate value. One might consider modifying the protocol so that P_1 and P_2 sample errors from different distributions. However, in such a case, the protocol remains secure for at most one party, specifically the one whose smudging error distribution is exponentially larger than that of the other.

Our construction (Standard Security). We introduce a novel approach to constructing a secure approximation computation protocol in the standard sense. A key insight from our analysis is that the protocol is unlikely to be secure if one party gains substantial (non-negligible) knowledge about the smudging error, as it is infeasible to design a simulator that extracts such information from the final output.

In a nutshell, we modify the ciphertext randomization procedure to prevent any information leakage about the smudging error. Instead of each party sampling its own error independently, they generate the smudging error in a distributed manner. As a result, neither party gains any partial knowledge of the error, ensuring security while allowing them to compute only the final output. To achieve this, we propose a *distributed smudging error sampling* that generates an additive share of a smudging error. The ideal functionality is illustrated in Fig. 11.

Functionality $\mathcal{F}_{\text{DistSamp}}$
<p>Setup: Let \mathcal{D} be a smudging error distribution over R and q_{out} denote the modulus of output ciphertext.</p> <hr/> <ol style="list-style-type: none"> 1. Sample $e \leftarrow \mathcal{D}$. 2. Sample $r_1 \xleftarrow{\\$} R_{q_{\text{out}}}$ and compute $r_2 := -r_1 + e \pmod{q_{\text{out}}}$. 3. Output r_i to P_i for $i = 1, 2$.

Fig. 11. Functionality of the distributed sampling for two parties

The distributed sampling scheme generates an additional share for two parties, ensuring that each share r_i is statistically indistinguishable from a uniformly sampled element from $R_{q_{\text{out}}}$. Below we describe ideal functionalities $\mathcal{F}_{\text{GenApp2PC}}$ and $\mathcal{F}'_{\text{GenApp2PC}}$ for approximate computation in Fig. 12, and our 2PC protocols $\Pi_{\text{GenApp2PC}}$ and $\Pi'_{\text{GenApp2PC}}$ in Fig. 13 which securely realize these functionalities.

Due to the functionality of distributed sampling, each r_i in P_i 's view is no longer correlated with the smudging error $e \leftarrow \mathcal{D}$. Recall that in the previously proposed liberally secure protocol $\Pi_{\text{LibApp2PC}}$, the

Functionality $\mathcal{F}_{\text{GenApp2PC}}$ (or $\mathcal{F}'_{\text{GenApp2PC}}$)
Setup: Let C be an evaluation circuit and \mathcal{D} be an error distribution. The functionality receives x from P_1 (the client) and y from P_2 (the server).
<ol style="list-style-type: none"> 1. Compute $z := C(x, y)$. 2. Sample $e \leftarrow \mathcal{D}$ and let $\hat{z} := z + e$. 3. Output \hat{z} to both P_1 and P_2 (or only to P_1).

Fig. 12. Functionality of a general approximate 2PC

Protocol $\Pi_{\text{GenApp2PC}}$ (or $\Pi'_{\text{GenApp2PC}}$)
<ol style="list-style-type: none"> 1. P_1 generates a key pair $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)$. P_1 sends $(\text{pk}, \text{ct}_{\text{in}})$ to P_2. 2. P_1 and P_2 call $\mathcal{F}_{2\text{-DistSamp}}$ and each party P_i obtains r_i for $i = 1, 2$. 3. P_2 computes $\text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}, y)$ and $\text{ct}_{\text{zero}} \leftarrow \text{Enc}_{\text{pk}}(0)$. P_2 computes $\text{ct}'_{\text{out}} := \text{ct}_{\text{out}} + \text{ct}_{\text{zero}} + (r_2, 0)$, and sends it to P_1. 4. P_1 computes $\hat{z} := \text{Dec}_{\text{sk}}(\text{ct}'_{\text{out}} + (r_1, 0))$. 5. In $\Pi_{\text{GenApp2PC}}$, P_1 sends \hat{z} to P_2. Both P_1 and P_2 output \hat{z}. In $\Pi'_{\text{GenApp2PC}}$, P_1 outputs \hat{z} while P_2 outputs nothing.

Fig. 13. General approximate 2PC protocols

output is of the form $\hat{z} = z' + e_1 + e_2$, where $e_i \leftarrow \mathcal{D}$ is sampled from P_i for $i = 1, 2$. Consequently, each party could compute a better approximation $z' + e_1$ (or $z' + e_2$), than what the ideal functionality provides. In contrast, in the $\Pi_{\text{GenApp2PC}}$ protocol, two parties collaboratively sample and add the smudging error e to the resulting ciphertext, ensuring that both parties obtain only the final output $\hat{z} = z' + e$ without knowing further knowledge of z . This key distinction guarantees that the protocol $\Pi_{\text{GenApp2PC}}$ (or $\Pi'_{\text{GenApp2PC}}$) achieves the security in the standard sense.

Theorem 3. *Let $\text{CKKS} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ be a CKKS scheme with respect to an error estimator Estimate . For plaintexts x, y and a circuit C , suppose that $\|x\|_\infty \leq B_1$ and $\|y\|_\infty \leq B_2$ for some $B_1, B_2 > 0$. Let $E_0 > 0$ be an upper bound of public-key encryption error, i.e., $\Pr[\|\text{Dec}_{\text{sk}}(\text{Enc}_{\text{pk}}(0))\|_\infty \leq B_0]$ with an overwhelming probability. Let $E := \text{Estimate}(C, B_1, B_2) + B_0$. If CKKS is IND-CPA secure and \mathcal{D} is an E -smudging distribution, then the 2PC protocol $\Pi_{\text{GenApp2PC}}$ (or $\Pi'_{\text{GenApp2PC}}$) securely realizes the functionality $\mathcal{F}_{\text{GenApp2PC}}$ (or $\mathcal{F}'_{\text{GenApp2PC}}$, respectively) in the $\mathcal{F}_{\text{DistSamp}}$ -hybrid model.*

Proof. We first show the correctness of our protocols. Let $z := C(x, y)$ and $z' := \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}} + \text{ct}_{\text{zero}})$. From the definition of error estimator Estimate , we have $z' = z + e_{\text{out}}$ for some $e_{\text{out}} \in R$ such that $\|e_{\text{out}}\|_\infty \leq E$. Therefore, the output can be written as

$$\hat{z} = \text{Dec}_{\text{sk}}(\text{ct}'_{\text{out}} + (r_1, 0)) = \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}} + \text{ct}_{\text{zero}} + (r_2, 0) + (r_1, 0)) = z + e_{\text{out}} + e$$

for some $e \leftarrow \mathcal{D}$. Thus, the output distribution is statistically indistinguishable from the distribution of the output $z + e$ of the functionality $\mathcal{F}_{\text{GenApp2PC}}$ since \mathcal{D} is an E -smudging distribution.

To prove the security of $\Pi'_{\text{GenApp2PC}}$, we first consider a corrupted client P_1 . The simulator $\text{Sim}^1_{\text{GenApp2PC}}(x, \hat{z})$ for P_1 's view is illustrated in Fig. 14. As mentioned in the correctness proof, the decrypted result $z' := \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}} + \text{ct}_{\text{zero}})$ in the real protocol is $z' = z + e_{\text{out}}$ with $\|e_{\text{out}}\|_\infty \leq E$, and the output can

Simulator $\text{Sim}_{\text{GenApp2PC}}^1(x, \hat{z})$
<ol style="list-style-type: none"> 1. Generate $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)$. 2. Sample $r_1 \xleftarrow{\\$} R_{q_{\text{out}}}$ and let $u' := \hat{z} - r_1$. 3. Sample $a_{\text{out}} \xleftarrow{\\$} R_{q_{\text{out}}}$ and let $\text{ct}'_{\text{out}} := (b_{\text{out}} := a_{\text{out}}s + u', a_{\text{out}})$. 4. Output $(x, r_1, \text{sk}, \text{pk}, \text{ct}_{\text{in}}, \text{ct}'_{\text{out}})$.

Fig. 14. Simulator for an adversarial client in protocol $\Pi_{\text{GenApp2PC}}$.

Simulator $\text{Sim}_{\text{GenApp2PC}}^{2'}(y, \hat{z})$
<ol style="list-style-type: none"> 1. Generate $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$ and $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(0)$. 2. Sample $r_2 \leftarrow R_q$. 3. Output $(y, r_2, \text{pk}, \text{ct}_{\text{in}})$.

Fig. 15. Simulator for an adversarial server in protocol $\Pi_{\text{GenApp2PC}}$.

be written as $\hat{z} = z' + e = z + e_{\text{out}} + e$. Therefore, we have:

$$\begin{aligned}
& \{(\text{view}_1^{\Pi_{\text{GenApp2PC}}}(x, y), \text{output}^{\Pi_{\text{GenApp2PC}}}(x, y))\} \\
& \equiv \{(x, r_1, \text{sk}, \text{pk}, \text{ct}_{\text{in}}, \text{ct}'_{\text{out}} := \text{ct}_{\text{out}} + \text{ct}_{\text{zero}} + (r_2, 0)), (z + e_{\text{out}} + e, \perp) \mid \\
& \quad (r_1, r_2) \leftarrow \mathcal{F}_{\text{DistSamp}}, \text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}, y), \text{ct}_{\text{zero}} \leftarrow \text{Enc}_{\text{pk}}(0), e \leftarrow r_1 + r_2\} \\
& \approx_s \{(x, r_1, \text{sk}, \text{pk}, \text{ct}_{\text{in}}, \text{ct}'_{\text{out}} := \text{ct}_{\text{out}} + \text{ct}_{\text{zero}} + (e - e_{\text{out}} - r_1, 0)), (z + e_{\text{out}} + e - e_{\text{out}}, \perp) \mid \\
& \quad (r_1, r_2) \leftarrow \mathcal{F}_{\text{DistSamp}}, \text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}, y), \text{ct}_{\text{zero}} \leftarrow \text{Enc}_{\text{pk}}(0)\} \\
& \equiv \{(x, r_1, \text{sk}, \text{pk}, \text{ct}_{\text{in}}, \text{ct}'_{\text{out}} = ((a_{\text{out}} + a_{\text{zero}})s + z + e - r_1, a_{\text{out}} + a_{\text{zero}}), (z + e, \perp) \mid \\
& \quad (r_1, r_2) \leftarrow \mathcal{F}_{\text{DistSamp}}, \text{ct}_{\text{out}} = (b_{\text{out}}, a_{\text{out}}) \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_{\text{in}}, y), \text{ct}_{\text{zero}} = (b_{\text{zero}}, a_{\text{zero}}) \leftarrow \text{Enc}_{\text{pk}}(0)\} \\
& \approx_c \{(x, r_1, \text{sk}, \text{pk}, \text{ct}_{\text{in}}, \text{ct}'_{\text{out}} = (as + z + e - r_1, a), (z + e, \perp) \mid (r_1, r_2) \leftarrow \mathcal{F}_{\text{DistSamp}}, a \xleftarrow{\$} R_{q_{\text{out}}}\} \\
& \equiv \{(\text{Sim}_{\text{GenApp2PC}}^1(x, \hat{z}), (\hat{z}, \perp)) \mid (\hat{z}, \perp) \leftarrow \mathcal{F}'_{\text{GenApp2PC}}(x)\},
\end{aligned}$$

where $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}(1^\lambda)$, $\text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)$, and $z = C(x, y)$. The first statistical indistinguishability follows from the assumption that \mathcal{D} is an E -smudging distribution, while the second computational indistinguishability is derived from the fact that the distribution of a_{zero} obtained from $\text{ct}_{\text{zero}} = (b_{\text{zero}}, a_{\text{zero}}) \leftarrow \text{Enc}_{\text{pk}}(0)$ is indistinguishable from a uniform distribution over $R_{q_{\text{out}}}$ under the RLWE assumption. The same simulator can be used to prove the security of $\Pi_{\text{GenApp2PC}}$ since P_1 's view is identical in both protocols.

Next, we construct the simulator $\text{Sim}_{\text{GenApp2PC}}^{2'}(y, \hat{z})$ to simulate P_2 's view in the protocol $\Pi'_{\text{GenApp2PC}}$, as shown in in Fig. 15. Then, we have:

$$\begin{aligned}
& \{\text{view}_2^{\Pi'_{\text{GenApp2PC}}}(x, y), \text{output}^{\Pi'_{\text{GenApp2PC}}}(x, y)\} \\
& \equiv \{(y, r_2, \text{pk}, \text{ct}_{\text{in}}), (z + e_{\text{out}} + e, \perp) \mid (r_1, r_2) \leftarrow \mathcal{F}_{\text{DistSamp}}, \text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)\} \\
& \approx_s \{(y, r_2, \text{pk}, \text{ct}_{\text{in}}), (z + e_{\text{out}} + e, \perp) \mid r_2 \xleftarrow{\$} R_{q_{\text{out}}}, \text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x)\} \\
& \approx_s \{(y, r_2, \text{pk}, \text{ct}_{\text{in}}), (z + e, \perp) \mid r_2 \xleftarrow{\$} R_{q_{\text{out}}}, \text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(x), e \leftarrow \mathcal{D}\} \\
& \approx_c \{(y, r_2, \text{pk}, \text{ct}_{\text{in}}), (z + e, \perp) \mid r_2 \xleftarrow{\$} R_{q_{\text{out}}}, \text{ct}_{\text{in}} \leftarrow \text{Enc}_{\text{pk}}(0)\} \\
& \equiv \{(\text{Sim}_{\text{GenApp2PC}}^{2'}(y, \hat{z}), (\hat{z}, \perp)) \mid (\hat{z}, \perp) \leftarrow \mathcal{F}'_{\text{GenApp2PC}}(x)\}.
\end{aligned}$$

The first statistical indistinguishability follows from the distributed sampling functionality, while the second follows from the assumption that \mathcal{D} is an E -smudging distribution. The final computational indistinguishability is ensured by the IND-CPA security of CKKS. In $\Pi_{\text{GenApp2PC}}$, we define $\text{Sim}_{\text{GenApp2PC}}^2(y, \hat{z})$

similarly to $\text{Sim}_{\text{GenApp2PC}}^{2'}(y)$, except that it appends \hat{z} to its simulated view. Then, it can be shown in a similar manner that the joint distribution of the server's view and output $\{(\text{view}_{\text{GenApp2PC}}^{\Pi}(x, y), \text{output}_{\text{GenApp2PC}}^{\Pi}(x, y))\}$ is computationally indistinguishable from the distribution $\{(\text{Sim}_{\text{GenApp2PC}}^2(y, \hat{z}), (\hat{z}, \hat{z})) \mid (\hat{z}, \hat{z}) \leftarrow \mathcal{F}_{\text{SimApp2PC}}(x)\}$. \square

Two-party Distributed Sampling Protocol. Our protocols achieve standard security by employing a distributed sampling functionality. In this discussion, we demonstrate that this functionality can be implemented efficiently without significant performance overhead. Specifically, we present an example protocol that distributively samples a smudging error from a uniform distribution over the interval $[-2^{k-1}, 2^k)$ for some integer k .

To begin with, two parties, P_1 and P_2 can generate an additive share of a random bit using a 1-out-of-2 *Oblivious Transfer* (OT) protocol as follows:

1. P_1 samples $r_1 \xleftarrow{\$} \mathbb{Z}_q$ and a bit $b \leftarrow \{0, 1\}$, and computes $(m_0, m_1) = (b - r_1, 1 - b - r_1) \pmod{q}$
2. P_2 chooses a selection bit $\sigma \xleftarrow{\$} \{0, 1\}$.
3. The parties run OT protocols on inputs (m_0, m_1) for P_1 and σ for P_2 , so that P_2 receives m_σ .

Then, it can be shown that the resulting values $(r_1, r_2 := m_\sigma)$ form a random additive share of a random bit, thus realizing the functionality of distributed sampling for $U(\{0, 1\})$.

To extend this to larger values, P_1 and P_2 execute the protocol k times to obtain shares $(r_{1,i}, r_{2,i})$ of random bits for $0 \leq i \leq k-1$. They then locally compute $r_1 = \sum_{i=0}^{k-1} 2^i r_{1,i}$ and $r_2 := \sum_{i=0}^{k-1} 2^i r_{2,i} - 2^{k-1}$, respectively. These values (r_1, r_2) form a random share of an error e uniformly sampled from $[-2^{k-1}, 2^{k-1})$.

Finally, this method can be naturally extended to sample a smudging error over R in a distributed manner, where each coefficient is uniformly distributed over $[-2^{k-1}, 2^{k-1})$. The round complexity of this protocol remains constant, as all instances can be generated in parallel. Furthermore, the distributed sampling protocol can be pre-processed in an offline phase or executed simultaneously with $\Pi_{\text{GenApp2PC}}$ (or $\Pi'_{\text{GenApp2PC}}$). Note that we can easily extend this construction to the n -party case.

5 Secure Approximate Multi-party Computation from CKKS

Variants of the standard (single-key) CKKS scheme have been proposed to support multi-party computation, including threshold CKKS [1, 33, 34], multi-key CKKS [6, 22], and multi-group CKKS [24]. In particular, various studies [13, 33, 34] have explored privacy-preserving MPC protocol over approximate values using the threshold CKKS scheme. Contrary to the two-party setting, threshold HE schemes involve multiple entities, each holding an individual secret keys. To decrypt the result ciphertext, each key owner publishes a partial decryption result of the evaluated ciphertext, which is then aggregated to reconstruct the final result. During this process, each party incorporates auxiliary noise into its partial decryption to mitigate potential information leakage.

The protocol ensures symmetry among all parties, allowing any participant to perform ciphertext evaluation. the functionality of approximate MPC outputs \hat{z} to all participating parties. However, similar to the 2PC protocol, where the server may or may not receive the computed result, certain parties in an MPC protocol may not obtain the output. This scenario can be formally represented using a set of indices $I \subseteq [n]$, where only the parties $\{P_i\}_{i \in I}$ take the decryption result. Nevertheless, in this section, we focus on the case in which all parties obtain the output. The proposed approach can be simply extended to scenarios where the computation result is shared with only a subset of participants.

In Section 5.1, we demonstrate that CKKS-based MPC protocols using this noise smudging technique are secure under the liberal MPC security notion. However, as discussed in Section 4.3, even if all parties introduce larger noise to their partial decryptions, this approach does not satisfy the standard security guarantees required for MPC. To overcome this limitation, in Section 5.2, we extend the secure approximate 2PC protocol to a multi-party setting by integrating a distributed sampling scheme for multiple parties, thereby ensuring stronger and more formal security guarantees.

5.1 Construction of MPC of Approximations: Liberal Security

Recent works [33,34] utilize the threshold CKKS scheme to construct a secure approximate MPC protocol. In the protocol, all participated parties incorporate smudging noise to their partial decryptions to prevent the attacks on the IND-CPA^D security. However, existing works lacks rigorous security proof or formal functionality explanation. Since the noise smudging technique enhances the security of the participated party and each party integrate an auxiliary smudging noise to the partial decryption in existing protocols, they achieve liberal MPC security notion, but not a standard MPC security, as similar to Sec. 4.2. In this section, we provide a formal security proof that the existing MPC protocols adopting the threshold CKKS scheme satisfies liberal definition of secure MPC protocol.

In CKKS-based MPC protocol, all participating parties collaboratively generate a shared public key and construct additive shares of the secret key during the key generation phase, as described in Fig. 1. Subsequently, each party encrypts its input and broadcasts the corresponding ciphertext. Upon receiving ciphertexts from other parties, all parties evaluate the circuit and obtain an encrypted result. When parties decrypt the result ciphertext, each party computes a partial decryption and adds an extra noise which is sampled from the smudging distribution. We present an ideal functionality $\mathcal{F}_{\text{LibAppMPC}}$ (Fig. 16) of this scenario and provide a concrete protocol $\Pi_{\text{LibAppMPC}}$ to realize the functionality (Fig. 17).

Functionality $\mathcal{F}_{\text{LibAppMPC}}$
<p>Setup: Let C be an evaluation circuit and \mathcal{D} be an error distribution. The functionality receives x_i from n parties P_i ($1 \leq i \leq n$).</p> <hr style="border: 0.5px solid black;"/> <ol style="list-style-type: none"> 1. Compute $z = C(x_1, \dots, x_n)$. 2. For $1 \leq i \leq n$, sample $e_i \leftarrow \mathcal{D}$ and let $\hat{z} := z + \sum_{i=1}^n e_i$. 3. Output \hat{z} to all parties.

Fig. 16. Functionality of the MPC protocol of approximations for liberal security

Protocol $\Pi_{\text{LibAppMPC}}$
<ol style="list-style-type: none"> 1. All parties call $\mathcal{F}_{\text{ThKeyGen}}$. Each party P_i obtains (sk_i, pk). 2. Each party P_i encrypts $\text{ct}_i \leftarrow \text{Enc}_{\text{pk}}(x_i)$ and broadcasts it. 3. All parties compute $\text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}) \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_1, \dots, \text{ct}_n)$. 4. Each party P_i samples $e_i \leftarrow \mathcal{D}$, computes $u'_i = a_{\text{out}} \cdot s_i + e_i \pmod{q_{\text{out}}}$ and broadcasts it. 5. All parties compute and output $\hat{z} = b_{\text{out}} + \sum_{i=1}^n u'_i \pmod{q_{\text{out}}}$.

Fig. 17. A secure approximate MPC protocol for evaluating a circuit C .

In the following, we prove the correctness and liberal security of this protocol.

Theorem 4. *Let $\text{CKKS} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ be a CKKS scheme with respect to an error estimator Estimate . For plaintexts x 's and a circuit C , suppose that $\|x_i\|_{\infty} \leq B_i$ for some $B_i > 0$ and let $E := \text{Estimate}(C, \{B_i\}_{i \leq n})$. If CKKS is IND-CPA secure and \mathcal{D} is an E -smudging distribution, then the MPC protocol $\Pi_{\text{LibAppMPC}}$ securely realizes the functionality $\mathcal{F}_{\text{LibAppMPC}}$ in the liberal sense in the $F_{\text{SecretSharing-hybrid}}$ model.*

Proof. We first show the correctness of $\Pi_{\text{LibAppMPC}}$. Let $z := C(x_1, \dots, x_n)$ and $z' := \text{Dec}_{\text{sk}}(\text{ct}_{\text{out}}) := b_{\text{out}} + \sum_{i=1}^n a_{\text{out}} \cdot s_i$. From the error estimator Estimate , we have $z' = z + e_{\text{out}}$ for some $e_{\text{out}} \in R$ such that

Simulator $\text{Sim}_{\text{LibAppMPC}}^A((x_i)_{i \in A}, z, \hat{z})$
<ol style="list-style-type: none"> 1. Emulating $\mathcal{F}_{\text{ThKeyGen}}$, generate key pairs (sk_i, pk) for each $i \in [n]$. 2. Encrypt $\text{ct}_i \leftarrow \text{Enc}_{\text{pk}}(x_i)$ for $i \in A$. 3. Generate $\text{ct}_h \leftarrow \text{Enc}_{\text{pk}}(0)$. 4. Compute $\text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}) \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_1, \dots, \text{ct}_n)$. 5. Compute $e := \hat{z} - z$ and sample $(e_i)_{i \in [n]} \leftarrow \{(e_i)_{i \in [n]} \leftarrow \mathcal{D}^n \mid \sum_{i=1}^n e_i = e\}$. 6. Compute $u'_i := a_{\text{out}} \cdot s_i + e_i$ for $i \in A$. 7. Set $u'_h := \hat{z} - b_{\text{out}} - \sum_{i \in A} u'_i$. 8. Output $((x_i, r_i, \text{sk}_i)_{i \in A}, \text{pk}, (\text{ct}_i)_{i \in [n]}, \text{ct}_{\text{out}}, u'_h)$.

Fig. 18. Simulator for an adversarial client in protocol Π_C^{MPC} .

$\|e_{\text{out}}\|_\infty \leq E$. By the linearity of the decryption process, the output can be represented as follows:

$$\begin{aligned} \hat{z} &= b_{\text{out}} + \sum_{i=1}^n u'_i = b_{\text{out}} + \sum_{i=1}^n (a_{\text{out}} \cdot s_i + e_i) \pmod{q_{\text{out}}} \\ &= z' + e = z + e_{\text{out}} + e, \end{aligned}$$

for some $e \leftarrow \mathcal{D}$. Since \mathcal{D} is an E -smudging distribution, we can conclude that the distribution of the protocol's output is statistically indistinguishable from the distribution of the output $z + e$ in the functionality $\mathcal{F}_{\text{AppMPC}}$.

To prove the security of $\Pi_{\text{LibAppMPC}}$, we assume the case where adversary corrupting exactly $n - 1$ parties. We denote a party P_h be the only honest party and $A := [n] \setminus \{h\}$ be the set of adversarial parties' indices. The simulator $\text{Sim}_{\text{LibAppMPC}}^A((x_i)_{i \in A}, z, \hat{z})$ for A 's view is illustrated in Fig. 18. A key idea is to construct the honest party's partial decryption u'_h using the adversarial parties' partial decryptions to maintain the consistency. i.e., computing $u'_h = \hat{z} - b_{\text{out}} - \sum_{i \in A} u'_i$. Then, we have:

$$\begin{aligned} & \{(\text{view}_A^{\Pi_{\text{LibAppMPC}}}(\mathbf{x}), \text{output}^{\Pi_{\text{LibAppMPC}}}(\mathbf{x}))\} \\ & \equiv \{(x_i, e_i, \text{sk}_i)_{i \in A}, \text{pk}, (\text{ct}_i)_{i \in [n]}, \text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}), u'_h, z + e_{\text{out}} + e \mid \\ & \quad ((\text{sk}_i)_{i \in [n]}, \text{pk}) \leftarrow \mathcal{F}_{\text{ThKeyGen}}, \text{ct}_i \leftarrow \text{Enc}_{\text{pk}}(x_i) (i \in [n]), u'_i := a_{\text{out}} s_i + e_i (i \in [n])\} \\ & \equiv \{(x_i, e_i, \text{sk}_i)_{i \in A}, \text{pk}, (\text{ct}_i)_{i \in [n]}, \text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}), u'_h, z + e_{\text{out}} + e \mid \\ & \quad ((\text{sk}_i)_{i \in [n]}, \text{pk}) \leftarrow \mathcal{F}_{\text{ThKeyGen}}, \text{ct}_i \leftarrow \text{Enc}_{\text{pk}}(x_i) (i \in [n]), \\ & \quad u'_i := a_{\text{out}} s_i + e_i (i \in A), u'_h := z + e_{\text{out}} + e - b_{\text{out}} - \sum_{i \in A} u'_i\} \\ & \approx_s \{(x_i, e_i, \text{sk}_i)_{i \in A}, \text{pk}, (\text{ct}_i)_{i \in [n]}, \text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}), u'_h, z + e \mid \\ & \quad ((\text{sk}_i)_{i \in [n]}, \text{pk}) \leftarrow \mathcal{F}_{\text{ThKeyGen}}, \text{ct}_i \leftarrow \text{Enc}_{\text{pk}}(x_i) (i \in [n]), \\ & \quad u'_i := a_{\text{out}} s_i + e_i (i \in A), u'_h := z + e - b_{\text{out}} - \sum_{i \in A} u'_i\} \\ & \approx_c \{(x_i, e_i, \text{sk}_i)_{i \in A}, \text{pk}, (\text{ct}_i)_{i \in [n]}, \text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}), u'_h, z + e \mid \\ & \quad ((\text{sk}_i)_{i \in [n]}, \text{pk}) \leftarrow \mathcal{F}_{\text{ThKeyGen}}, \text{ct}_i \leftarrow \text{Enc}_{\text{pk}}(x_i) (i \in A), \text{ct}_h \leftarrow \text{Enc}_{\text{pk}}(0), \\ & \quad u'_i := a_{\text{out}} s_i + e_i (i \in A), u'_h := z + e - b_{\text{out}} - \sum_{i \in A} u'_i\} \\ & \equiv \{(\text{Sim}_{\text{LibAppMPC}}^A((x_i)_{i \in A}, z, \hat{z}), \hat{z}) \mid \hat{z} \leftarrow \mathcal{F}_{\text{AppMPC}}\}, \end{aligned}$$

where $\text{sk}_i = s_i (i \in [n])$, $\text{ct}_{\text{out}} \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_1, \dots, \text{ct}_n)$, and $e_i \leftarrow \mathcal{D} (i \in [n])$. We omit $(\text{mod } q_{\text{out}})$ while computing u'_i 's for better readability. The second equivalence comes from $z + e_{\text{out}} + e = b_{\text{out}} + \sum_{i=1}^n u'_i$, as mentioned in the correctness proof. The next statistical indistinguishability follows from the assumption that \mathcal{D} is an E -smudging distribution, and the last computational indistinguishability follows from the IND-CPA security of CKKS. \square

5.2 Construction of MPC of Approximations: Standard Security

As discussed earlier, the protocol $\Pi_{\text{LibAppMPC}}$ is insecure under the standard MPC security definition, as it is infeasible to valid simulators without access to the exact computation result. Therefore, it has a limitation that the ideal-process adversary has to obtain the exact value of the evaluated circuit. To overcome this limitation, we employ a distributed sampling approach, similar to the technique described in Section 4.3, to generate smudging noise in a distributed manner. The functionality of the distributed sampling algorithm for n parties, which produces additive shares of a smudging error, is illustrated in Fig. 19. Through this distributed sampling protocol, each party obtains an additive share r_i of the smudging noise e , which is incorporated into the partial decryption process during the decryption phase. This ensures that no participating party can obtain any partial information about the ciphertext except the final output of the protocol. Consequently, an approximate MPC protocol incorporating a distributed smudging error sampling mechanism satisfies the standard MPC security definition. The formal definitions of the functionality and the approximate MPC protocol in this setting are provided in Fig. 20 and Fig. 21, respectively.

Functionality $\mathcal{F}_{n\text{-DistSamp}}$
<p>Setup: Let \mathcal{D} be a smudging error distribution over R.</p> <hr/> <ol style="list-style-type: none"> 1. Sample $e \leftarrow \mathcal{D}$. 2. Sample $r_i \stackrel{\\$}{\leftarrow} R_{q_{\text{out}}}$ for $1 \leq i \leq n-1$, and let $r_n := e - \sum_{i=1}^{n-1} r_i \pmod{q_{\text{out}}}$. 3. Output r_i to party P_i for $1 \leq i \leq n$.

Fig. 19. Functionality of the distributed sampling for multiple parties

Functionality $\mathcal{F}_{\text{AppMPC}}$
<p>Setup: Let C be an evaluation circuit and \mathcal{D} be an error distribution. The functionality receives x_i from n parties P_i ($1 \leq i \leq n$).</p> <hr/> <ol style="list-style-type: none"> 1. Compute $z \leftarrow C(x_1, \dots, x_n)$. 2. Sample $e \leftarrow \mathcal{D}$ and let $\hat{z} := z + e$. 3. Output \hat{z} to all parties.

Fig. 20. Functionality of the MPC protocol of approximations

Protocol Π_{AppMPC}
<ol style="list-style-type: none"> 1. All parties call $\mathcal{F}_{\text{ThKeyGen}}$. Each party P_i obtains (sk_i, pk). 2. Each party P_i encrypts $\text{ct}_i \leftarrow \text{Enc}_{\text{pk}}(x_i)$ and broadcasts it. 3. The parties call $\mathcal{F}_{n\text{-DistSamp}}$ and each party P_i receives $r_i \in R_{q_{\text{out}}}$. 4. All parties compute $\text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}) \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_1, \dots, \text{ct}_n)$. 5. Each party P_i computes $u'_i = a_{\text{out}} \cdot s_i + r_i \pmod{q_{\text{out}}}$ and broadcasts it. 6. All parties compute and output $\hat{z} = b_{\text{out}} + \sum_{i=1}^n u'_i \pmod{q_{\text{out}}}$.

Fig. 21. A secure approximate MPC protocol for evaluating a circuit C .

Simulator $\text{Sim}_{\text{AppMPC}}^A((x_i)_{i \in A}, \hat{z})$
<ol style="list-style-type: none"> 1. Emulating $\mathcal{F}_{\text{ThKeyGen}}$, generate key pairs (sk_i, pk) for each $i \in [n]$. 2. Generate $\text{ct}_i \leftarrow \text{Enc}_{\text{pk}}(0)$ for all $i \in H$. 3. Encrypt $\text{ct}_i \leftarrow \text{Enc}_{\text{pk}}(x_i)$ for all $i \in A$. 4. Compute $\text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}) \leftarrow \text{Eval}_{\text{pk}}(C, \text{ct}_1, \dots, \text{ct}_n)$. 5. Sample $r_i \xleftarrow{\\$} R_{q_{\text{out}}}$ and $u'_i := a_{\text{out}} \cdot s_i + r_i$ for all $i \in A$. 6. Sample $u'_i \xleftarrow{\\$} R_{q_{\text{out}}}$ for all $i \in H \setminus \{h\}$. 7. Set $u'_h := \hat{z} - b_{\text{out}} - \sum_{i \in [n] \setminus \{h\}} u'_i \pmod{q_{\text{out}}}$. 8. Output $((x_i, r_i, \text{sk}_i)_{i \in A}, \text{pk}, (\text{ct}_i)_{i \in [n]}, \text{ct}_{\text{out}}, (u'_i)_{i \in H})$.

Fig. 22. Simulator for an adversarial client in protocol Π_C^{MPC} .

Now, we show that the approximate MPC protocol Π_{AppMPC} is secure in the $F_{\text{SecretSharing}}$ -hybrid model.

Theorem 5. *Let $\text{CKKS} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$ be a CKKS scheme with respect to an error estimator Estimate . For plaintexts x_i 's and a circuit C , suppose that $\|x_i\|_\infty \leq B_i$ for some $B_i > 0$ and let $E := \text{Estimate}(C, \{B_i\}_{i \leq n})$. If CKKS is IND-CPA secure and \mathcal{D} is an E -smudging distribution, then the MPC protocol Π_{AppMPC} securely realizes the functionality $\mathcal{F}_{\text{AppMPC}}$ in the $F_{\text{SecretSharing}}, \mathcal{F}_{n\text{-DistSamp}}$ -hybrid model in a standard sense.*

Proof. We first guarantee the correctness of Π_{AppMPC} . Let $z := C(x_1, \dots, x_n)$ and $z' := b_{\text{out}} + \sum_{i=1}^n a_{\text{out}} \cdot s_i$. From the error estimator Estimate , we have $z' = z + e_{\text{out}}$ for some $e_{\text{out}} \in R$ such that $\|e_{\text{out}}\|_\infty \leq E$. By the linearity of the decryption process, the output can be represented as follows:

$$\begin{aligned} \hat{z} &= b_{\text{out}} + \sum_{i=1}^n u'_i = b_{\text{out}} + \sum_{i=1}^n a_{\text{out}} \cdot s_i + r_i \pmod{q_{\text{out}}} \\ &= z' + e = z + e_{\text{out}} + e \end{aligned}$$

for some $e \leftarrow \mathcal{D}$. Since \mathcal{D} is an E -smudging distribution, we can conclude that the distribution of the protocol's output is statistically indistinguishable from the distribution of the output $z + e$ in the functionality $\mathcal{F}_{\text{AppMPC}}$.

To prove the security of Π_{AppMPC} , we first denote $A \subsetneq [n]$ and $H = [n] \setminus A$ as a set of indices corresponding to the corrupted parties and honest parties, respectively. We construct a simulator $\text{Sim}_{\text{AppMPC}}^A((x_i)_{i \in A}, \hat{z})$ for the corrupted parties, as described in Fig. 22. A key idea is to construct the last honest party's partial decryption u'_h using the other u'_i 's to ensure consistency. Emulating $\mathcal{F}_{n\text{-DistSamp}}$, the simulator samples r_i uniformly at random from $R_{q_{\text{out}}}$ and computes the partial decryption $u'_i := a_{\text{out}} \cdot s_i + r_i$ for the adversarial parties. For the honest parties, the simulator samples from uniform $R_{q_{\text{out}}}$, except for the last honest party P_h , which it computes $u'_h = \hat{z} - \sum_{i \in [n] \setminus \{h\}} u'_i$. Then, we have:

$$\begin{aligned}
& \{(\text{view}_A^{\Pi_C^{MPC}}(\mathbf{x}), \text{output}^{\Pi_C^{MPC}}(\mathbf{x}))\} \\
& \equiv \{((x_i, r_i, \text{sk}_i)_{i \in A}, \mathbf{pk}, (\text{ct}_i)_{i \in [n]}, \text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}), (u'_i)_{i \in H}, z + e_{\text{out}} + e) \mid \\
& \quad ((\text{sk}_i)_{i \in [n]}, \mathbf{pk}) \leftarrow \mathcal{F}_{\text{ThKeyGen}}, \text{ct}_i \leftarrow \text{Enc}_{\mathbf{pk}}(x_i) (i \in [n]), \\
& \quad (r_i)_{i \in [n]} \leftarrow \mathcal{F}_{n\text{-DistSamp}}, u'_i := a_{\text{out}} s_i + r_i \pmod{q_{\text{out}}} (i \in [n])\} \\
& \equiv \{((x_i, r_i, \text{sk}_i)_{i \in A}, \mathbf{pk}, (\text{ct}_i)_{i \in [n]}, \text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}), (u'_i)_{i \in H}, z + e_{\text{out}} + e) \mid \\
& \quad ((\text{sk}_i)_{i \in [n]}, \mathbf{pk}) \leftarrow \mathcal{F}_{\text{ThKeyGen}}, \text{ct}_i \leftarrow \text{Enc}_{\mathbf{pk}}(x_i) (i \in [n]), \\
& \quad (r_i)_{i \in [n]} \leftarrow \mathcal{F}_{n\text{-DistSamp}}, u'_i \stackrel{\$}{\leftarrow} R_{q_{\text{out}}}(i \in H \setminus \{h\}), u'_h := (z + e_{\text{out}} + e) - b_{\text{out}} - \sum_{i \in [n] \setminus \{h\}} u'_i\} \\
& \approx_s \{((x_i, r_i, \text{sk}_i)_{i \in A}, \mathbf{pk}, (\text{ct}_i)_{i \in [n]}, \text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}), (u'_i)_{i \in H}, z + e) \mid \\
& \quad ((\text{sk}_i)_{i \in [n]}, \mathbf{pk}) \leftarrow \mathcal{F}_{\text{ThKeyGen}}, \text{ct}_i \leftarrow \text{Enc}_{\mathbf{pk}}(x_i) (i \in [n]), \\
& \quad (r_i)_{i \in [n]} \leftarrow \mathcal{F}_{n\text{-DistSamp}}, u'_i \stackrel{\$}{\leftarrow} R_{q_{\text{out}}}(i \in H \setminus \{h\}), u'_h := (z + e) - b_{\text{out}} - \sum_{i \in [n] \setminus \{h\}} u'_i\} \\
& \approx_c \{((x_i, r_i, \text{sk}_i)_{i \in A}, \mathbf{pk}, (\text{ct}_i)_{i \in [n]}, \text{ct}_{\text{out}} := (b_{\text{out}}, a_{\text{out}}), (u'_i)_{i \in H}, z + e) \mid \\
& \quad ((\text{sk}_i)_{i \in [n]}, \mathbf{pk}) \leftarrow \mathcal{F}_{\text{ThKeyGen}}, \text{ct}_i \leftarrow \text{Enc}_{\mathbf{pk}}(x_i) (i \in A), \text{ct}_i \leftarrow \text{Enc}_{\mathbf{pk}}(0) (i \in H), \\
& \quad (r_i)_{i \in [n]} \leftarrow \mathcal{F}_{n\text{-DistSamp}}, u'_i \stackrel{\$}{\leftarrow} R_{q_{\text{out}}}(i \in H \setminus \{h\}), u'_h := (z + e) - b_{\text{out}} - \sum_{i \in [n] \setminus \{h\}} u'_i\} \\
& \equiv \{(\text{Sim}_{\text{AppMPC}}^A((x_i)_{i \in A}, \hat{z}), \hat{z}) \mid \hat{z} \leftarrow \mathcal{F}_{\text{AppMPC}}\},
\end{aligned}$$

where $\text{ct}_{\text{out}} \leftarrow \text{Eval}_{\mathbf{pk}}(C, \text{ct}_1, \dots, \text{ct}_n)$ and $u'_i := a_{\text{out}} s_i + r_i \pmod{q_{\text{out}}}$ for $i \in A$. We omit $\pmod{q_{\text{out}}}$ for u'_h for better readability. The second equivalence comes from $z + e_{\text{out}} + e = b_{\text{out}} + \sum_{i=1}^n u'_i \pmod{q_{\text{out}}}$, as mentioned in the correctness proof. The next statistical indistinguishability follows from the assumption that \mathcal{D} is an E-smudging distribution, and the computational indistinguishability follows from the IND-CPA security of CKKS. \square

We remark that our proposed construction offers an advantage over previous MPC protocols based on the threshold HE schemes: the error in the final decryption result is lower compared to existing approaches. In conventional noise smudging techniques, the accumulated decryption error increases with the number of participating parties, as each party introduces its own independent error during the partial decryption process. In contrast, our proposed construction ensures that the final decryption error remains independent of the number of parties, thereby enhancing accuracy without compromising security.

It is worth noting that the proposed protocols operate under an n -out-of- n security model, where all participating entities must disclose their partial decryptions to reconstruct the correct evaluation result. However, this framework can be generalized to a t -out-of- n threshold setting by incorporating secret-sharing techniques, such as Shamir's secret sharing, to enable threshold decryption.

References

1. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38. pp. 565–596. Springer (2018)
2. Bourse, F., Del Pino, R., Minelli, M., Wee, H.: Fhe circuit privacy almost for free. In: Annual International Cryptology Conference. pp. 62–89. Springer (2016)
3. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual cryptology conference. pp. 868–886. Springer (2012)
4. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. pp. 868–886. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)* **6**(3), 1–36 (2014)
6. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. pp. 395–412 (2019)
7. Cheon, J.H., Choe, H., Passelègue, A., Stehlé, D., Suvanto, E.: Attacks against the ind-cpad security of exact fhe schemes. In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. pp. 2505–2519 (2024)
8. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic Encryption for Arithmetic of Approximate Numbers. In: *Proc. of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2017)* (2017)
9. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: *international conference on the theory and application of cryptology and information security*. pp. 3–33. Springer (2016)
10. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: *Annual Cryptology Conference*. pp. 643–662. Springer (2012)
11. Ducas, L., Stehlé, D.: Sanitization of the ciphertexts. In: *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8–12, 2016, Proceedings, Part I* 35. pp. 294–310. Springer (2016)
12. Feigenbaum, J., Ishai, Y., Malkin, T., Nissim, K., Strauss, M.J., Wright, R.N.: Secure multiparty computation of approximations. *ACM transactions on Algorithms (TALG)* **2**(3), 435–472 (2006)
13. Froelicher, D., Cho, H., Edupalli, M., Sousa, J.S., Bossuat, J.P., Pyrgelis, A., Troncoso-Pastoriza, J.R., Berger, B., Hubaux, J.P.: Scalable and privacy-preserving federated principal component analysis. In: *2023 IEEE Symposium on Security and Privacy (SP)*. pp. 1908–1925. IEEE (2023)
14. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: *Proc. of the Forty-First Annual ACM Symposium on Theory of Computing (STOC 2009)* (2009). <https://doi.org/10.1145/1536414.1536440>
15. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: *International conference on machine learning*. pp. 201–210. PMLR (2016)
16. Goldreich, O.: *Foundations of Cryptography, Volume 2*. Cambridge university press Cambridge (2004)
17. Guo, Q., Nabokov, D., Suvanto, E., Johansson, T.: Key recovery attacks on approximate homomorphic encryption with non-worst-case noise flooding countermeasures. In: *Usenix Security* (2024)
18. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al.: Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning* **14**(1–2), 1–210 (2021)
19. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making spdz great again. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. pp. 158–189. Springer (2018)
20. Kim, M., Harman, A.O., Bossuat, J.P., Carpov, S., Cheon, J.H., Chillotti, I., Cho, W., Froelicher, D., Gama, N., Georgieva, M., et al.: Ultrafast homomorphic encryption models enable secure outsourcing of genotype imputation. *Cell systems* **12**(11), 1108–1120 (2021)
21. Kim, M., Jiang, X., Lauter, K., Ismayilzada, E., Shams, S.: Secure human action recognition by encrypted neural network inference. *Nature communications* **13**(1), 4799 (2022)
22. Kim, T., Kwak, H., Lee, D., Seo, J., Song, Y.: Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition. In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. pp. 726–740 (2023)
23. Kluczniak, K., Santato, G.: On circuit private, multikey and threshold approximate homomorphic encryption. *Cryptology ePrint Archive, Paper 2023/301* (2023), <https://eprint.iacr.org/2023/301>
24. Kwak, H., Lee, D., Song, Y., Wagh, S.: A general framework of homomorphic encryption for multiple parties with non-interactive key-aggregation. In: *International Conference on Applied Cryptography and Network Security*. pp. 403–430. Springer (2024)
25. Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., et al.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access* **10**, 30039–30054 (2022)
26. Lee, S., Lee, G., Kim, J.W., Shin, J., Lee, M.K.: HETAL: efficient privacy-preserving transfer learning with homomorphic encryption. In: *International Conference on Machine Learning*. pp. 19010–19035. PMLR (2023)
27. Li, B., Micciancio, D.: On the security of homomorphic encryption on approximate numbers. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. pp. 648–677. Springer (2021)

28. Li, B., Micciancio, D., Schultz-Wu, M., Sorrell, J.: Securing approximate homomorphic encryption using differential privacy. In: Annual International Cryptology Conference. pp. 560–589. Springer (2022)
29. Lindell, Y.: How to simulate it—a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich* pp. 277–346 (2017)
30. Liu, F., Zheng, Z., Shi, Y., Tong, Y., Zhang, Y.: A survey on federated learning: a perspective from multi-party computation. *Frontiers of Computer Science* **18**(1), 181336 (2024)
31. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the forty-fourth annual ACM symposium on Theory of computing. pp. 1219–1234 (2012)
32. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Advances in Cryptology—EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 1–23. Springer (2010)
33. Mouchet, C., Bertrand, E., Hubaux, J.P.: An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption. *Journal of Cryptology* **36**(2), 10 (2023)
34. Mouchet, C., Troncoso-Pastoriza, J., Bossuat, J.P., Hubaux, J.P.: Multiparty homomorphic encryption from ring-learning-with-errors. *Proceedings on Privacy Enhancing Technologies* **2021**(4), 291–311 (2021)
35. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key fhe. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 735–763. Springer (2016)
36. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)* **56**(6), 1–40 (2009)