A Composable View of Verifiable Homomorphic Encryption in Multi-Party Settings

Ganyuan Cao

School of Computer and Communication Sciences

Master's Thesis

Responsible
Prof. Dr. Alessandro Chiesa
EPFL
Laboratory for Computation Security (CompSec)

Supervisor
Christian Knabenhans
EPFL
Laboratory for Computation Security (CompSec)

Supervisor
Dr. Sylvain Chatel
CISPA Helmholtz Center for Information Security

Co-examiner
Dr. Christian Mouchet
Hasso Plattner Institute

March 2025

Acknowledgement

I would first like to express my sincere gratitude to Alessandro Chiesa, Christian Knabenhans, and Sylvain Chatel for their dedicated supervision and for giving me the opportunity to conduct my Master's thesis within the CompSec group. Their guidance has been invaluable throughout this journey. I am also thankful to everyone in the CompSec and SPIRNG groups for the insightful discussions that have deepened my understanding and strengthened my work. Additionally, I would like to thank Giacomo Fenzi for his invaluable clarifications on key technical challenges that were crucial to this thesis. I am also grateful to Christian Mouchet for serving as the co-examiner on the committee and for providing valuable feedback on this work.

I would also like to extend my sincerest gratitude to Fabio Banfi and Ueli Maurer for supervising my semester project, which introduced me to composable security and sparked my deep interest in the field. My heartfelt thanks go to my internship supervisor, Najib Aouini, who guided me through my internship on homomorphic encryption, first exposing me to the field and inspiring me to further analyze its security. Their guidance led me to explore this direction further, ultimately shaping the focus of my Master's thesis. I am also deeply appreciative of the DuoKey team for their support during my internship and to Patricia Genet for her invaluable assistance in securing the internship and handling the administrative aspects of my Master's thesis.

Finally, I am forever grateful to my family for their unwavering and unconditional support. Without them, I would not have come this far. My mother, Daiying Yang, has been my constant source of strength, always by my side, listening to my frustrations, and calming me down when I felt overwhelmed. I am also deeply thankful to my father, Guoqiang Cao, whose unwavering support has given me the foundation to pursue my aspirations with confidence. Additionally, I am grateful to my grandmother, Peijun Zhang, my step grandmother, Xun Wang, my aunt, Ling Ling, my grandaunt, Lihua Wang, and my granduncle, Chunming Ling, for their generosity and care, which have played an essential role in my journey. My grandfather, Yaowu Yang, has been a guiding light, always offering wisdom and support whenever I faced major life decisions. Lastly, I hold in my heart my late grandfather, Jinlong Cao, and my late grandmother, Yinhua Wang, who now rest in peace. Their blessings and love continue to accompany me, giving me strength every step of the way.

An ending of a journey marks a beginning of another one. It is now time for the ~~imitation~~ simulation game...

Abstract

Homomorphic Encryption (HE) is a powerful cryptographic tool that enables computations on encrypted data, with the resulting ciphertexts reflecting the same operations as if they were executed on plaintexts. Notably, HE has been extended to a multi-party setting to facilitate collaborative computation on encrypted inputs from multiple sources. This capability is particularly valuable in fields such as cloud computing and privacy-preserving machine learning since it naturally serves as a secure multiparty computation (MPC), which guarantees the confidentiality of inputs from users.

However, while HE effectively preserves privacy, it does not inherently ensure the integrity of the inputs from users and the computed results. Guaranteeing this is vital for real-world applications that necessitate both confidentiality and integrity.

Establishing a security model for HE presents challenges due to its novel nature and the complexity of its use cases. Achieving integrity alongside confidentiality requires the integration of HE with other cryptographic tools, such as (Zero-Knowledge) Succinct Non-Interactive Arguments of Knowledge (zkSNARKs). Traditional game-based security models often struggle to encapsulate the intricate interactions that arise from such compositions, especially in the multi-party setting.

In this thesis, we revisit the security properties of HE in a multi-party setting by introducing new game-based security notions. From the perspective of composable security, we analyze the conditions for realizing an ideal functionality of HE in multi-party settings in the Universal Composability (UC) framework by establishing a reduction from the UC functionality to our proposed game-based notions.

We then extend this functionality by composing it with zkSNARKs to provide integrity through verifiable computation. We prove that this composition realizes an on-the-fly MPC system introduced by López-Alt et al. at STOC'13, which leverages an untrusted yet powerful server to assist computations over inputs from dynamically joining parties. We prove security against a malicious adversary capable of statically corrupting a subset of clients and the server. Our analysis addresses both confidentiality and integrity from the perspectives of clients and the server: client inputs remain private and non-malleable, while the server's circuits are kept confidential, and the correctness of evaluations is verifiable.

In this thesis, we provide a dual-perspective analysis of composable security for HE in a multi-party setting, focusing on both the secure realization of the primitive and its security when composed with other cryptographic building blocks in a larger system.

Keywords: Universal Composability, Homomorphic Encryption, Verifiable Computation, Multi-Party Computation, zkSNARK

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Homomorphic Encryption (HE), initially termed as "privacy homomorphism" by Rivest et al. [RAD$^+$78], enables computations to be performed directly on encrypted data. The resulting ciphertexts reflect the same operations as if they were executed on the underlying plaintexts. This capability has profound implications for real-world applications, including privacy-preserving machine learning [LKL$^+$22], cloud computing [ZLL14], and secure multiparty computation (MPC) [BDOZ11]. Over time, HE has evolved from earlier cryptographic schemes like RSA [RSA78] and Paillier [Pai99], which support only addition, and ElGamal [ElG85], which supports only multiplication, to *Fully Homomorphic Encryption* (FHE) [Gen09a, BGV12, CKKS17, CGGI20], capable of evaluating circuits of arbitrary depth.

To further enhance flexibility in multi-party setting, *Threshold Homomorphic Encryption* (ThHE) [AJL$^+$12, BGG$^+$18, MBH23] and *Multi-Key Homomorphic Encryption* (MKHE) [LTV12, CM15, BP16, PS16] have been proposed. ThHE requires collaboration among participants to jointly recover the secret but requires the participant set to be fixed before computation begins. On the other hand, MKHE allows inputs encrypted under different keys from multiple sources to be processed, allowing participants to dynamically join computation. However, MKHE struggles with scalability as the number of participants grows. More recently, *Multi-Group Homomorphic Encryption* (MGHE) [AH19, Par21, CMS$^+$23, KLSW24] has been introduced as a hybrid approach to perform encryption and evaluation at the group level, which combines the flexibility of MKHE with the scalability of ThHE.

However, it is important to note that HE, in its basic form, guarantees *confidentiality* alone. Thus, it is both natural and essential to consider *integrity* as another core objective of cryptography. Beyond the classical notion of integrity, which emphasizes the untampered transmission of messages, it is critical to ensure that any computation performed on the encrypted data is correct and honest. Increasing attention in the research community has been dedicated to incorporating integrity into homomorphic contexts. A prominent approach discussed in several works[Via23, KVMGH23, TW24] involves the composition of HE with *(Zero-Knowledge) Succinct Non-interactive Arguments of Knowledge* (zkSNARKs) [BCTV13], allowing for efficient verification of the correctness of computations.

Various notions based on *game-based proof* framework by Bellare and Rogaway [BR06] have also been proposed to address different adversarial scenarios that may arise during message evaluation. However, game-based proof may not be the most effective tool for formalizing security in this context. The complication of defining a game or adversarial behavior increases with the presence of message evaluation, especially in a multi-party setting. Moreover, these primitives are intended for use in large, complex systems, yet the composition property of these primitives in such systems remains unclear with game-based notions, as noted by Maurer [Mau11], particularly given that some primitives like zkSNARKs are generally formalized with *simulation-based proof.* To bridge this gap, our goal is to present an analysis of HE and augmentations on it for integrity from a composable perspective, ensuring that we capture the intended functionality of each primitive while also guaranteeing security in their compositions.

1.2 Related Work

In this section, we revisit some prior works on the homomorphic encryption (in multi-party settings) and techniques to add integrity to the primitive.

1.2.1 Established Notions for Confidentiality

In this section, we revisit some security notions defined to formalize confidentiality of homomorphic encryption. Specifically, we consider both client-side and server-side confidentiality, and also the security in multi-party settings where a subset of parties is corrupted.

Client-Side Confidentiality. The primary security requirement of homomorphic encryption is *semantic security*, formalized with the IND-CPA notion, as established by Gentry [Gen09a]. Additionally, Gentry introduced the notion of *circular security*, which addresses scenarios where an adversary has access to a "cycle of keys", meaning a key is encrypted under itself or another key. This notion models the security related to the *bootstrapping* operation.

In addition to security definitions for HE on exact numbers, Li and Micciancio [LM21] introduced the notion of IND-CPA$^{\mathrm{D}}$ for HE schemes on approximate numbers, such as [CKKS17]. This definition permits the adversary to query a restricted decryption oracle that only decrypts when the challenge ciphertexts are equivalent with respect to a given circuit. Subsequent works [AV21, CCP$^+$24, GNSJ24, CSBB24] identified that this issue arises from imperfect correctness and decryption failures rather than approximate evaluation. Recently, Bergamaschi et al. [BCD$^+$24] proposed a stronger security notion where the adversary is given an one-time access to the random coins used during encryption. Around the same time, Bernard et al. [BJSW24] introduced an even stronger variant allowing the adversary to specify the random coins used in encryption.

Recently, Chaturvedi et al. [CCM$^+$25] introduced the notion of IND-CPA$^{\mathrm{C}}$, which models the setting considered by Goldwasser [GKP$^+$13], where a server is allowed to decrypt the output of a predefined computation and nothing else. This notion captures scenarios in which HE serves as a building block of *functional decryption* [BSW11].

Although these notions are defined to account for different attacks, we note that these notions focus on guarantee the confidentiality of the inputs from the client.

Server-Side Confidentiality. Together, in a classical client-server computation delegation, the aforementioned notions focus on the confidentiality on the client side. Beyond this, confidentiality on the server side has also been discussed. The notion of *circuit privacy* [Gen09a, IP07, BdPMW16] guarantees that evaluated ciphertexts do not leak information about the server's circuit by requiring them to be (statistically) indistinguishable from ciphertexts generated by a simulator. Ducas and Stehlé [DS16] later termed the re-randomization techniques [Gen09b, AJL$^+$12] used to achieve circuit privacy as *ciphertext sanitization* and introduced a security notion such that any two ciphertexts encrypting the same message are (statistically) indistinguishable.

More recently, Hwang et al. [HMS25] argued that the *noise-flooding* [Gen09b, AJL$^+$12] technique for circuit privacy and the *soak-spin-repeat* [DS16] technique for ciphertext sanitizability incur significant practical overhead. To address this, they proposed the notion of *ciphertext simulatability*, a relaxation of circuit privacy. Specifically, this notion relaxes circuit privacy by allowing the evaluation algorithm to reveal circuit information and granting the ciphertext simulator access to the secret key. Additionally, they introduced the notion of *error simulatability*, where the simulator is only required to simulate the error in the output ciphertext of a circuit. Kluczniak and Santato [KS23] also proposed a relaxation of circuit privacy by introducing a game-based notion $\mathrm{IND\text{-}CIRC}$ by considering an indistinguishability game with a challenge evaluation oracle. There are also several other relaxations [BdPMW16, Gen09b] that allows the disclosure of the circuit depth. Additionally, other security notions [OPP14, DD22] address circuit leakage from a broader perspective by defining security in terms of a leakage function or even granting the simulator access to the circuit's view. We also note that [OPP14, DD22] consider *malicious* circuit privacy, where ciphertexts and evaluation keys may be adversarially generated.

Beyond focusing solely on circuits, Knabenhans et al. [KVMGH23] discussed about the case where certain server inputs do not require homomorphic evaluation but must still remain private. A notable example is the protection of machine learning model weights. To address this, they extended the security scope to encompass all private inputs provided by the server, beyond just the circuits.

Regardless of whether the definitions are simulation-based or game-based, or the scope of the circuit, we note these notions are defined to ensure the secrecy of the inputs from the server who does the evaluation.

Multi-Party Security. In the multi-party setting, López-Alt et al. [LTV12] stated that the semantic security of MKHE follows directly from the semantic security of its underlying encryption scheme. Subsequent works on MKHE-based MPC [MW16, AJJM20] explored partial decryption using individual secret keys from participants and analyzed the (statistical) indistinguishability of partial decryptions.

For ThHE, Asharov et al. [AJL$^+$12] were the first to combine the simulatability of partial decryptions with semantic security in the security proof of an MPC protocol. Boneh et al. [BGG$^+$18] later formalized the security of ThHE by explicitly considering the simulatability of each party's partial decryption. Boudgoust and Scholl [BS23] later proposed a variant of $\mathrm{IND\text{-}CPA}^{\mathrm{D}}$ in threshold setting providing a game-based formalization of ThHE. Passelègue and Stehlé further refined the notions introduced in [BS23] and [BGG$^+$18].

Kluczniak and Santato [KS23] also defined a variant of $\mathrm{IND\text{-}CPA}^{\mathrm{D}}$ tailored for both ThHE and MKHE. In the context of MGHE, Kwak et al. [KLSW24] analyzed $\mathrm{IND\text{-}CPA}$ security under partial corruption of groups and examined the simulatability of partial decryption for honest clients in their security proofs.

In addition to ensuring the confidentiality of clients' inputs, we note that these notions also aim to address scenarios where a subset of parties is corrupted, which is a common but necessary consideration in the multi-party settings.

1.2.2 Techniques and Notions for Integrity

In this section, we review some existing techniques for ensuring integrity when homomorphic evaluation is involved. Specifically, we consider homomorphic signatures (HS), homomorphic message authenticators (HA), and the use of zero-knowledge proofs for verifiable computation.

Homomorphic Signature. In [JMSW02], Johnson et al. introduced the concept of a *Homomorphic Signature* (HS). Since then, numerous works have advanced this primitive, particularly in the context of network coding, including [BFKW09, GKKR10, AL11]. However, these works are limited to linear functions. Boneh and Freeman later extended homomorphic signatures to support polynomial functions in [BF11].

We note that the primary goal with homomorphic signatures is to enable the evaluation of signed data, yet it does not ensure the integrity or the correctness of the evaluation itself.

Homomorphic Message Authenticator. Agrawal and Boneh [AB09] introduced the concept of a *Homomorphic Message Authenticator* (HA), a symmetric variant of the work by Johnson et al. [JMSW02], although they focused on its application in network coding. Later, Gennaro and Wichs [GW13] introduced the first fully homomorphic message authenticator and formalized the related security notion of unforgeability through the concept of a labeled program. Their construction can be viewed as a MAC-then-Encrypt (MtE) approach, where a MAC is applied to the plaintext before encrypting the authenticated plaintext using FHE. However, their method for verifying that a circuit has been honestly evaluated involves reconstructing a Merkle Tree representing the circuit, which is not efficiently verifiable.

Catalano and Fiore [CF13] introduced the term HomoUF-CMA for the unforgeability notion, which was later extended to SUF-CMA in [JY14] to account for the honest execution of the evaluation algorithm of an FHE scheme from a "ciphertext integrity" perspective. They proposed a construction that is efficiently verifiable, but only supports circuits of limited depth. Recently, Chatel et al. [CKP$^+$22] generalized these two constructions to achieve efficient verification and support for arbitrary circuits.

In addition to the MtE approach, other works explored Encrypt-then-MAC (EtM) [FGP14, BGV11] and Encrypt-and-MAC (E&M) paradigms [LWZ18], both of which require the encryption and MAC schemes to be homomorphically evaluatable. Notably, Li et al. [LWZ18] introduced the concept of a privacy-preserving homomorphic MAC, formalizing an additional security requirement through an indistinguishability game to ensure that the tag does not reveal the underlying message.

Although HA can guarantee the correctness of evaluation, we note that its verification may be inefficient, as it could require nearly the same computational effort as the evaluation itself. This undermines the purpose of outsourcing computation to a powerful server, which is one of the key use cases of homomorphic encryption.

Zero-Knowledge Proof. In this approach, the server first evaluates the FHE circuit and then generates a *(Zero-Knowledge) Succinct Non-interactive ARgument of Knowledge* ((zk)SNARK) as the proof of correct evaluation. A SNARK allows for the creation of a short proof for any NP statement, proving the prover's knowledge of the corresponding witness efficiently. Here, the witness is the circuit (zero-knowledge is needed if circuit is private), and the statement asserts that the circuit has been honestly evaluated to produce the output ciphertext from the given input.

The integration of (zk)SNARKs into FHE constructions has been explored multiple times in the literature. Notable examples include Boneh et al. [BSW12] and Canetti et al. [CRRV17], both employing the Naor-Yung paradigm [NY90] for encryption but replacing the Non-Interactive Zero-Knowledge proof system (NIZK) [BFM88] with a zkSNARK for efficiency. Later, during evaluation, zkSNARK is used again prove the honest execution of the evaluation algorithm. This technique is proven to construct a CCA1-secure FHE. More recent works utilizing this technique include Fiore et al. [FNP20], followed by Bois et al. [BCFK21]. Viand et al. [Via23, KVMGH23] introduced the concept of *verifiable FHE* by directly combining a CPA-secure FHE scheme with a SNARK, where the SNARK is only used to prove the honest execution of the evaluation algorithm. We remark that their construction, however, is not CCA1 secure given that it does not guarantee the integrity of ciphertexts that are obtained by simply encrypting a message. More recently, Manulis and Nguyen [MN24] similarly follows Naor-Yung paradigm as in [BSW12, CRRV17] to provide integrity of the ciphertexts but only run one round of the evaluation algorithm. They also showed that this technique actually achieves a stronger security (vCCA) they proposed that is stronger than CCA1 but slightly weaker than CCA2 security. Canard et al. [CFP+24] further extended the analysis in [MN24] to specifically account for schemes on approximate numbers.

On the security of (zk)SNARKs, Ganesh et al. [GKO+23] highlighted the necessity for zk-SNARKs to satisfy the stronger notion of *straightline simulation-extractability* [Sah99, DDO+01, PR05, FKMV12, JP14] to be secure in the UC framework, and UC-secure zkSNARK is possible only in *random oracle model* (ROM). They also introduced a transformation that lifts any simulation-extractable (but not necessarily straightline) zkSNARK into a UC-secure one in a model that provides both a global reference string and a global random oracle (that is only observable) using a compiler based on a polynomial commitment scheme. Subsequently, Chiesa and Fenzi [CF24] demonstrated that well-known zkSNARKs, including Micali construction [Mic00] and BCS construction [BCS16], are UC-secure without requiring modifications in the *pure* ROM with the random oracle being both *observable* and *programmable*. These works ensure that our discussion can proceed under the assumption that UC-secure zkSNARKs (in ROM) exist.

We note that zkSNARKs appear to be the most promising candidate for augmenting HE with integrity. Due to their succinctness, verification and communication between parties can be efficient. The heavy computation with HE can be outsourced to a powerful but untrusted server, while the client only needs to perform light computation to verify the proof to ensure the server has correctly computed the result.

1.3 Contribution

In this thesis, we provide an analysis of security of homomorphic encryption in a multi-party setting and its composition with zkSNARK to achieve integrity through verifiability. We utilize the *Universal Composability* (UC), a model for composable security introduced by Canetti [Can01] and refined in many later works [CDPW07, CKKR19, BCH+20], for security proofs in this thesis. The UC framework enables us to systematically examine the functionalities of individual cryptographic primitives and analyze their security when composed for larger, more complex systems. In this thesis, we specifically focus on *multi-group homomorphic encryption* (MGHE) and present an analysis from a dual perspective that emphasizes both secure realization of this primitive, and its external application in constructing a *on-the-fly MPC* protocol [LTV12], providing a complete view of the composable security of this primitive, following a roadmap illustrated in Figure 1.1. Specifically, our contributions are outlined as follows.

$$\begin{tikzcd}
\mathcal{G}_{\mathsf{KRK}} \arrow[r, "\Pi_{\mathsf{MGHE}}", "\text{Theorem 2 \& 3}"'] & \mathcal{F}_{\mathsf{MGHE}} \arrow[dr, "\Pi_{\mathsf{OtF\text{-}MPC}}", "\text{Theorem 5}"'] & \\
 & & \mathcal{F}_{\mathsf{OtF\text{-}MPC}} \\
\mathcal{G}_{\mathsf{RO}} \arrow[r, "\Pi_{\mathsf{NARG}}", "\text{Theorem 4}"'] & \mathcal{F}^{R_*}_{\mathsf{zkSNARK}} \arrow[ur] &
\end{tikzcd}$$

Figure 1.1: A roadmap of the constructions in this thesis.

– **Confidentiality of HE in a Multi-Party Setting**: We provide a concrete analysis of the confidentiality properties of homomorphic encryption in a multi-party setting. From a game-based security perspective, we revisit and introduce new notions for both client-side and server-side confidentiality. For client-side security, we extend the IND-CPA$^{\mathrm{D}}$ security game to address the specific requirements of threshold, multi-key, and multi-group homomorphic encryption. This family of notions, which we denote as IND-CPA$^{\mathrm{pD}}$, captures the cases involving the corruption of a subset of players and disclosure of partial decryptions using player's individual secret keys. On the server side, we similarly generalize the IND-CIRC framework to suit multi-party settings. These notions we introduce also account for the presence of a semi-malicious adversary capable of encrypting with malicious random coins, thereby reflecting the behavior of corrupted players in a multi-party system.

  From the UC security perspective, we examine a client-server model for computation delegation by specifically focusing on multi-group HE. In this setting, n clients can dynamically form groups of arbitrary size and collaboratively provide inputs for various computation tasks. We analyze the conditions for MGHE to provide both client-side and server-side confidentiality and bridge it with the game-based notions we introduce. Client-side security models that even in the presence of partial group corruption or complete server corruption, the inputs provided by clients remain confidential. Server-side security guarantees that the circuit evaluated by the server is not disclosed to the adversary even when a client group is completely corrupted.

– **Composition for On-the-Fly MPC**: We revisit the construction of *on-the-fly* MPC, initially introduced by López-Alt et al. [LTV12]. This system allows an untrusted server [1] to assist dynamic and arbitrary computations on data from multiple users without requiring user interaction or revealing the results to the server. It is particularly valuable when dealing with complex circuits or public computations that demand a powerful yet untrusted evaluator.

  Following the previous works on CCA1-secure homomorphic encryption [LMSV12, BSW12, CRRV17, MN24], we start the construction by augmenting MGHE with client-side message integrity, adopting the Naor-Yung double encryption paradigm [NY90]. To further provide integrity on server-side, we integrate zkSNARKs on the server side to validate the correctness of server-side evaluations. In the final phase, we further guarantee the correctness of partial decryption by using zkSNARKs to validate the decryption process. We show that this construction is secure against a malicious adversary capable of statically corrupting the server and up to the corruption threshold of clients in any group and possible the server.

To the best of our knowledge, this is the first work to establish UC-security for MGHE and its composition with zkSNARKs for verifiable homomorphic encryption in a multi-party setting. In contrast to prior works on CCA1-secure homomorphic encryption [LMSV12, BSW12, CRRV17, MN24], our analysis explicitly addresses the multi-party setting. Moreover, unlike previous works on on-the-fly MPC [LTV12, MW16, KLSW24], we also provide a concrete analysis of server-side security. In this thesis, we adopt a holistic perspective, focusing on the security of the system as a whole rather than the standalone security.

The structure of this thesisis outlined as follows. In Chapter 2, we provide the necessary preliminaries, including the notations used in this thesisand background knowledge on the UC framework with a global setup, which the security proofs in this thesisare based on. In Chapter 3, we delve into the confidential layer of the system, offering an analysis of MGHE from both game-based and UC perspectives. Chapter 4 introduces the construction of the integrity layer,

[1]By "untrusted server", we mean that the server is honest-but-curious when it is not corrupted by the adversary.

focusing on verifiability achieved through zkSNARK. In Chapter 5, we demonstrate how these two layers are composed to construct a UC-secure on-the-fly MPC protocol. Finally, Chapter 6 summarizes the results of this thesisand outlines potential directions for future research.

Chapter 2

Preliminaries

2.1 Notation

Throughout this paper, we adopt the following notations. The set of natural numbers is denoted by $\mathbb{N} = \{1, 2, \ldots\}$, and for any $n \in \mathbb{N}$, the notation $[n] := 1, \ldots, n$ represents the integers from 1 to n. A *list* is a sequence $\mathbf{x} = (x_1, \ldots, x_\ell)$ where each element in it can be a single item or a tuple. The length of a list $\mathbf{x}$ is then denoted by $|x| = \ell$. An empty list is initialized as $\mathbf{x} := (\,)$. To denote appending an element x' to a list $\mathbf{x}$, we use the notation $\mathbf{x}' := \mathbf{x} \bowtie x'$ where the new list $x' = (x_1, \ldots, x_\ell, x')$. A *bitstring* is defined as a list $x = (x_1, \ldots, x_\ell)$ where each $x_i \in \{0, 1\}$ for $i \in [\ell]$. For two bitstrings $x = (x_1, \ldots, x_\ell)$ and $y = (y_1, \ldots, y_{\ell'})$, their concatenation is then written as $x \parallel y := (x_1, \ldots, x_\ell, y_1, \ldots, y_{\ell'})$.

A *lookup table* $\mathcal{T}$ is a mapping from keys k to values v. An empty lookup table is initialized as $\mathcal{T} := [\,]$. To assign a value v to a key k, we write $\mathcal{T}[k] := v$, with any existing value at $\mathcal{T}[k]$ being overwritten. To retrieve the value associated with a key k, we use $v := \mathcal{T}[k]$. If no value is assigned to $\mathcal{T}[k]$, the symbol $\perp$ is returned to indicate this.

The symbol := is also used for defining terms or variables, such as $f(x) := x^2$, which defines $f(x)$ as x^2. We use $x \leftarrow \mathcal{A}$ to denote the assignment of a value produced by a probabilistic algorithm $\mathcal{A}$ to x. When sampling uniformly at random from a set S, we write $x \leftarrow\!\!\$\ S$. If sampling follows a specific distribution $\mathcal{D}$, the notation $x \leftarrow_{\mathcal{D}} S$ is used instead.

Additional notations will be introduced as necessary later throughout this thesis.

2.2 Game-based Proof

In this work, we discuss some security notions that follow the code-based game-playing framework of Bellare and Rogaway [BR06]. This framework employs a game G composed of an *Initialization* procedure (INIT), a *Finalization* procedure (FIN), and a set of oracle procedures, whose number varies depending on the specific game. An adversary $\mathcal{A}$ interacts with these oracles, receiving responses to its queries through return statements specified in the oracle codes.

A game G begins with the INIT procedure, followed by the adversary's interaction with the oracles. After making a series of oracle queries, the adversary halts and produces an *adversary output*. Subsequently, the FIN procedure is executed to generate a *game output*. If no explicit finalization procedure is defined, the *adversary output* is considered the *game output*.

We use the notation $y \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{O1}}, \mathcal{O}_{\mathrm{O2}}, \cdots}$ to denote running $\mathcal{A}$ given access to oracles $\mathcal{O}_{\mathrm{O1}}, \mathcal{O}_{\mathrm{O2}}, \ldots$, and then assignment of the output of $\mathcal{A}$ to y. We denote $\Pr[\mathcal{A}^{\mathrm{INIT}, \mathcal{O}_{\mathrm{O1}}, \mathcal{O}_{\mathrm{O2}}, \cdots} \Rightarrow b]$ as the probability that the adversary $\mathcal{A}$ outputs a value b after the INIT procedure and queries to oracles $\mathcal{O}_{\mathrm{O1}}, \mathcal{O}_{\mathrm{O2}}, \ldots$. We denote $\Pr[\mathrm{G}(\mathcal{A}) \Rightarrow b]$ as the probability that game G outputs b when adversary $\mathcal{A}$ plays game G. For simplicity, we define $\Pr[\mathrm{G}(\mathcal{A})] := \Pr[\mathrm{G}(\mathcal{A}) \Rightarrow 0]$. To simplify notation, we interchangeably use $\Delta_{\mathcal{A}}\left(\mathrm{O}_L; \mathrm{O}_R\right)$ and

$$\Delta_{\mathcal{A}}\begin{pmatrix}\mathrm{O}_L\\ \mathrm{O}_R\end{pmatrix} := \Pr[\mathcal{A}^{\mathrm{O}_L} \Rightarrow 0] - \Pr[\mathcal{A}^{\mathrm{O}_R} \Rightarrow 0]$$

to denote $\mathcal{A}$'s advantage in distinguishing between the oracles O_L and O_R.

We let $\mathbf{Adv}^{\mathrm{x}}_{\Pi}(\mathcal{A}_x)$ denote adversary $\mathcal{A}_x$'s advantage in breaking security notion X of a scheme Π. We say security notion X implies security notion Y, denoted $\mathsf{X} \mapsto \mathsf{Y}$, if $\mathbf{Adv}^{\mathrm{y}}_{\Pi}(\mathcal{A}_y) \leq c \cdot \mathbf{Adv}^{\mathrm{x}}_{\Pi}(\mathcal{A}_x)$ for some constant $c > 0$. We say security notion X is equivalent to security notion Y, denoted $\mathsf{X} \leftrightarrow \mathsf{Y}$, if $\mathsf{X} \mapsto \mathsf{Y}$ and $\mathsf{Y} \mapsto \mathsf{X}$.

2.3 Universal Composability (UC)

In this thesis, we utilize the *Universal Composability* (UC) framework introduced by Canetti [Can00]. The main advantage of the UC framework is that protocols are guaranteed to maintain their security in any context, even in the presence of arbitrarily many protocol sessions that run concurrently in an adversarially controlled manner.

2.3.1 System of ITMs

The UC framework is a *simulation-based* framework where security is defined in terms of a comparison between a "real world" versus an "ideal world". In the real world, a multi-party protocol Π is executed among *parties* $\mathcal{P}_1, \ldots, \mathcal{P}_n$ who are modeled as *Interactive Turing Machines* (ITM). An adversary $\mathcal{A}$, who is also modeled as an ITM, carries attacks on the protocol Π. The adversary can corrupt parties, gaining access to their current and past states as well as the contents of all their tapes. Once a party is corrupted, the adversary gains full control over its future actions.

In the ideal world, a trusted *functionality* $\mathcal{F}$, modeled as an ITM and representing the desired security property, interacts with (dummy) parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$. These dummy parties can only forward inputs to them to $\mathcal{F}$ and outputs responses from $\mathcal{F}$ to them as their outputs. The protocol in which $\mathcal{F}$ interacts with the dummy parties is referred as an *ideal protocol* $\mathrm{I}_{\mathcal{F}}$. Additionally, there is an *ideal-process adversary* $\mathcal{S}$, also commonly called the *simulator*. The simulator's capabilities are restricted to corrupting parties, delaying messages, or suppressing messages sent from $\mathcal{F}$ to a party.

An *environment* $\mathcal{Z}$, who is sometimes referred as a *distinguisher* and also modeled as an ITM, chooses protocol inputs to all parties and may read all outputs locally made by the parties and communicate with the adversary. Informally, we say a protocol is UC-secure if there is no environment that can distinguishes between whether it is interacting with the real adversary $\mathcal{A}$ or the ideal-process adversary $\mathcal{S}$.

Inspired by previous works on composable security [BM18, BBM18, BBD24], we adopt a code-based approach to describe the behaviors of functionalities, simulators, and protocols in this thesis. This approach offers greater clarity compared to the typical verbal descriptions often used in works about Universal Composability (UC) framework.

We use $\mathcal{P}_{\mathsf{pid}}$ to represent a party in the system, where pid denotes the party's ID. An *instance* (also referred as a *session* sometimes) of the ITM (also denoted as ITI) is machine in the system that is specified as $(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid})$, where $\mathcal{P}_{\mathsf{pid}}$ is the party running the machine and sid identifies the session in which it operates. When describing a protocol, we use the notation $\mathcal{P}_{\mathsf{pid}} \leftarrow \langle \mathtt{Act} : [\mathsf{sid}, \mathsf{pid}, v] \rangle$ to indicate its action upon receiving a specific action "identifier" $\mathtt{Act}$

and a value v needed for the action in a specific session sid. In the code for an ideal functionality $\mathcal{F}$, we use the notation **In** $\langle \texttt{Act} : [\mathsf{sid}, \mathsf{pid}, v] \rangle \leftarrow \mathcal{P}_{\mathsf{pid}}$ to indicate that the dummy party $\mathcal{P}_{\mathsf{pid}}$ sends the input to $\mathcal{F}$ and the machine $(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid})$ interacts with the functionality $\mathcal{F}$ in a session sid. If an input can be sent from more than one party, we use the notation **In** $\langle \cdot \rangle \leftarrow \{\mathcal{P}_{\mathsf{pid}_1}, \mathcal{P}_{\mathsf{pid}_2}\}$. The functionality $\mathcal{F}$ then processes according to the action identifier $\texttt{Act}$ and generates a response. The response is denoted by **Out** $\{\texttt{"res"} : [\mathsf{sid}, \mathsf{pid}, v]\} \rightarrow \mathcal{P}_{\mathsf{pid}}$, indicating that $\mathcal{F}$ sends the response of the type $\texttt{"res"}$ back to the party $\mathcal{P}_{\mathsf{pid}}$.

To simplify syntax, we assume that a dummy party $\mathcal{P}_{\mathsf{pid}}$, when interacting with a functionality $\mathcal{F}$, does not use a different party ID for identification purposes, i.e., it does not send an input with $\mathsf{pid}' \neq \mathsf{pid}$. In other works on UC-security [Can01, Can03, CDG+18], this is typically enforced by having the functionality $\mathcal{F}$ abort if $\mathcal{P}_{\mathsf{pid}}$ submits requests with a mismatched party ID pid'.

2.3.2 UC Emulation, Realization and Composition

UC Emulation. Informally, a protocol Π is said to *UC-emulate* another protocol Ψ in the UC framework if, for any adversary $\mathcal{A}$ attacking it, there exists a simulator $\mathcal{S}$ "attacking" Ψ, such that no *environment* $\mathcal{Z}$ can distinguish between these two executions. Note that we can consider two types of protocols. In the first case, the communication between the parties in this protocol is over an insecure channel. We also refer to this as a *bare* model. In the second case, the protocol runs (multiple sessions of) an ideal functionality $\mathcal{F}$ as a subroutine. We refer to such a protocol as a *$\mathcal{F}$-hybrid protocol*. For the completeness, in Definitions 1 and 2, we consider a $\mathcal{F}$-hybrid model. Essentially, a bare model can be considered as a $\mathcal{F}$-hybrid model with $\mathcal{F}$ that simply forwards the message.

Definition 1 (UC-Emulation [Can00])**.** Let Π and Ψ be a n-party protocol such that Π is a $\mathcal{F}$-hybrid protocol for some ideal functionality $\mathcal{F}$. We say that Π *UC-emulates* Ψ in the *$\mathcal{F}$-hybrid* model if, for any PPT adversary $\mathcal{A}$, there exists a PPT *simulator* $\mathcal{S}$ such that for any PPT *environment* $\mathcal{Z}$, it holds that

$$\Delta_{\mathcal{Z}}\left(\{\mathsf{EXEC}_{\mathcal{F},\Pi,\mathcal{A},\mathcal{Z}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x},\lambda,z}; \{\mathsf{EXEC}_{\Psi,\mathcal{S},\mathcal{Z}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x},\lambda,z}\right) \leq \varepsilon(\lambda)$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ are the inputs from parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, z is the auxiliary input for $\mathcal{A}$, and λ is the security parameter. We use the notation $\Delta_{\mathcal{Z}}(\cdot\,;\,\cdot)$ to denote the probability that the environment $\mathcal{Z}$ distinguishes between these two worlds.

UC Realization. However, the emulation of a protocol alone may not adequately capture its security. For instance, by trivially setting $\Pi := \Psi$ and $\mathcal{S} := \mathcal{A}$, the environment $\mathcal{Z}$ is unable to distinguish between the two executions but it does not imply any security. To address this, it is essential to consider the *ideal protocol* $\mathrm{I}_{\mathcal{F}}$. We say that a protocol Π *UC-realizes* a functionality $\mathcal{F}$ if the protocol Π UC-emulates the ideal protocol $\mathrm{I}_{\mathcal{F}}$. In Definition 2, we formally define UC-realization for an ideal functionality $\mathcal{F}$.

Definition 2 (UC-Realization [Can00])**.** Let $\mathcal{F}$ and $\mathcal{F}$' be ideal functionalities. Let Π be a $\mathcal{F}$-hybrid protocol. We say that Π *UC-realizes $\mathcal{F}'$* in $\mathcal{F}$-hybrid model if, for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for any PPT environment $\mathcal{Z}$, it holds that

$$\Delta_{\mathcal{Z}}\left(\{\mathsf{REAL}_{\mathcal{F},\Pi,\mathcal{A},\mathcal{Z}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x},\lambda,z}; \{\mathsf{IDEAL}_{\mathcal{F}',\mathcal{S},\mathcal{Z}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x},\lambda,z}\right) \leq \varepsilon(\lambda)$$

where $\mathsf{REAL}_{\mathcal{F},\Pi,\mathcal{A},\mathcal{Z}} := \mathsf{EXEC}_{\mathcal{F},\Pi,\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}_{\mathcal{F}',\mathcal{S},\mathcal{Z}} := \mathsf{EXEC}_{\mathrm{I}_{\mathcal{F}'},\Pi,\mathcal{A},\mathcal{Z}}$ with $\mathrm{I}_{\mathcal{F}'}$ being the ideal protocol of $\mathcal{F}'$, $\mathbf{x} = (x_1, \ldots, x_n)$ are the inputs from parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, z is the auxiliary input for $\mathcal{A}$, and λ is the security parameter.

Composition Theorem. The UC framework allows us to show the security when multiple protocols are composed for another protocol. We let the protocol Φ be a $\mathcal{F}$-hybrid protocol and let Π be a protocol that UC-realizes an ideal functionality $\mathcal{F}$. The intuition is that we can then construct a *composed protocol* $\Phi^{\mathcal{F}\to\Pi}$ by changing Φ's calls to a session of $\mathcal{F}$ to a session of Π. At a high level, the composition theorem states that running the protocol $\Phi^{\mathcal{F}\to\Pi}$ is *at least as secure* as running the original protocol Φ who invokes the ideal functionality $\mathcal{F}$.

We describe some properties of the protocols described by Canetti [Can00] for the composition theorem to hold. Informally, we say that a protocol Φ is (Π, Ψ)-*compliant*[1] [Can00, Section 6.1] if it does not invokes both protocol Π and Ψ with the same session identifier. We say a protocol is *subroutine-respecting* [Can20, Definition 19] if the subprotocols in it communicate only with parties outside their session through the main protocol. We say a protocol is *subroutine-exposing* [Can20, Definition 21] if the adversary can obtain a list of subroutines invoked by the protocol.

Theorem 1 (UC Composition Theorem [Can00, Theorem 22]). *Let Φ, Π, Ψ be PPT protocols. Let Φ is (Π, Ψ)-compliant. Let Π and Ψ be subroutine-respecting and subroutine-exposing. If Π UC-emulates Ψ, then $\Phi^{\Psi\to\Pi}$ UC-emulates Φ.*

By Theorem 1, if we replace Ψ with the ideal protocol $\mathrm{I}_{\mathcal{F}}$ for some ideal functionality $\mathcal{F}$, then it is the theorem for UC-realization. We then use the notation $\Phi^{\mathcal{F}\to\Pi}$ as earlier to denote that.

2.3.3 Global Setup & Subroutine

Generalized UC. In the basic UC framework, a functionality can only be accessed by a single protocol instance. This means that each protocol instance relies on its own independent setup, separate from the setups of other instances. However, this model is not particularly realistic sometimes. For example, in this thesis, we consider a random oracle (RO) who serves as an idealization of a hash function. Therefore, it should not be restricted to a single protocol instance since everyone can run a hash function.

To address these cases, Canetti et al. [CDPW07] introduced the *Generalized* UC (GUC) framework, which allows all protocol instances to share access to a *global setup* functionality $\mathcal{G}$. In this framework, the environment $\mathcal{Z}$ is no longer constrained and can also interact with the global functionality $\mathcal{G}$. Depending on the context, the environment $\mathcal{Z}$ may *directly* send inputs to $\mathcal{G}$ or instruct the adversary (or corrupted parties via the adversary) to send inputs to the global functionality.

Note in basic UC, the environment $\mathcal{Z}$ is restricted to invoking only a *single session* of the "challenge protocol" Π. This means all ITIs running the parties involved in Π must share the same sid. In contrast, GUC lifts this restriction, allowing the environment $\mathcal{Z}$ to invoke multiple sessions of the challenge protocol Π alongside other protocols, which makes the environment *unconstrained*.

UC with Global Subroutine. However, this GUC framework has limitations in that the environment becomes too "unconstrained" which makes some protocols impossible to be secure, and the global setup cannot have multiple sessions. This may be a problem in some cases. For example, in a PKI, the user may choose to use different public keys for different purposes but the

[1]In the original definition, an addition identity-bound predicate ξ is also introduced. To be a compliant protocol, the extended identities in an extended session of protocol Φ must also satisfy the predicate ξ. This is less related to this thesisso we omit the discussion on this part.

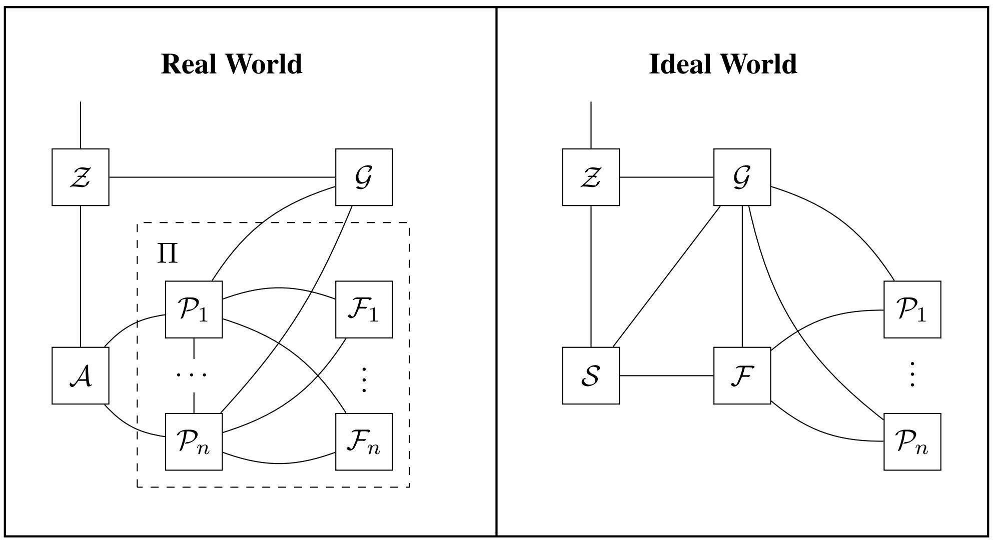

Figure 2.1: An illustration of UC model with global subroutine $\mathcal{G}$. In the real world, a "real" adversary $\mathcal{A}$ attacks a session of a $[\mathcal{F}_1, \ldots, \mathcal{F}_n]$-hybrid protocol Π between $\mathcal{P}_1, \ldots, \mathcal{P}_n$ that uses functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_n$ as subroutines. In the real world, an ideal-process adversary (simulator) $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}$ which we would like to realize and the (dummy) parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ sends input for the execution of $\mathcal{F}$. The global functionality $\mathcal{G}$ is outside the protocol instance which is accessible also from the environment $\mathcal{Z}$.

global key registry introduced by Canetti et al. [CDPW07] restricts the use of the same public key during the lifetime of a party $\mathcal{P}_{\mathsf{pid}}$.

Badertscher et al. [BCH+20] proposed UC with *global subroutine* (UCGS) by using a transformation named *management protocol* [BCH+20, Section 3.1] to allow plain UC framework capture the global setup. A management protocol $\mathsf{Mgmt}[\Pi, \mathcal{G}]$ for a protocol Π and a global subroutine $\mathcal{G}$ is a UC-protocol, such that one instance of $\mathsf{Mgmt}[\Pi, \mathcal{G}]$ behaves like a single instance of Π along with one or more instances of $\mathcal{G}$. Specifically, the incoming communication to $\mathsf{Mgmt}[\Pi, \mathcal{G}]$ specifies a session sid, plus a session identifier for either Π or a specific instance of $\mathcal{G}$. The input is then passed to Π or the respective instance of $\mathcal{G}$.

UCGS can actually be viewed as a hybrid approach between basic UC and GUC. Namely, the environment $\mathcal{Z}$ is allowed to invoke a single instance of the challenge protocol Π while it is allowed to invoke arbitrarily many instances of the global subroutine $\mathcal{G}$. This also allows us better flexibility when defining the global subroutine since we no longer need to fix the global functionality for the lifetime of a party.

Definition 3 (UC-Emulation with Global Subroutine [BCH+20, Definition 3.1])**.** Let Π, Ψ be protocols and let $\mathcal{G}$ be a global subroutine. Let Mgmt be a management protocol. If $\mathsf{Mgmt}[\Pi, \mathcal{G}]$ UC-emulates $\mathsf{Mgmt}[\Psi, \mathcal{G}]$, then we say that Π UC-emulates Ψ in presence of a global subroutine $\mathcal{G}$. We use the notation

$$\Delta_{\mathcal{Z}}\left(\{\mathsf{EXEC}^{\mathcal{G}}_{\mathcal{F},\Pi,\mathcal{A},\mathcal{Z}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x},\lambda,z}; \{\mathsf{EXEC}^{\mathcal{G}}_{\Psi,\mathcal{S},\mathcal{Z}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x},\lambda,z}\right) \leq \varepsilon(\lambda)$$

to denote the advantage of the environment in distinguishing between these two execution in presence of a global subroutine $\mathcal{G}$.

Definition 4 (UC-realization with Global Subroutine)**.** Let Π be protocols. Let $\mathcal{F}$ be an ideal functionality. Let $\mathcal{G}$ be a global subroutine. Let Mgmt be a management protocol. If $\mathsf{Mgmt}[\Pi, \mathcal{G}]$

UC-emulates $\mathsf{Mgmt}[\mathrm{I}_{\mathcal{F}}, \mathcal{G}]$ where $\mathrm{I}_{\mathcal{F}}$ is the ideal protocol of $\mathcal{F}$, then we say that Π UC-realizes $\mathcal{F}$ in presence of a global subroutine $\mathcal{G}$. We use the notation

$$\Delta_{\mathcal{Z}}\left(\{\mathsf{REAL}^{\mathcal{G}}_{\mathcal{F},\Pi,\mathcal{A},\mathcal{Z}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x},\lambda,z}; \{\mathsf{IDEAL}^{\mathcal{G}}_{\mathcal{F}',\mathcal{S},\mathcal{Z}}(\mathbf{x}, \lambda, z)\}_{\mathbf{x},\lambda,z}\right) \leq \varepsilon(\lambda)$$

to denote the advantage of the environment $\mathcal{Z}$ in distinguishing between these two executions in presence of a global subroutine $\mathcal{G}$.

For composition theorem in this framework, we need two additional properties of the protocols. We say that a protocol Π is *$\mathcal{G}$-subroutine-respecting* if its subprotocols are allowed to pass and receive output from $\mathcal{G}$. We say that $\mathcal{G}$ is *Π-regular* if $\mathcal{G}$ is not allowed to spawn new ITMs and from using Π as a subroutine. Note in Corollary 1, we directly state the theorem for UC-realization since it is more needed than emulation.

Corollary 1 (UC Composition with Global Subroutine [BCH$^+$20, Theorem 3.5]). *Let $\mathcal{G}$ be a global subroutine. Let* Mgmt *be a management protocol. Let Φ, Ψ and Π be protocols. Assume that*

(1) Φ is subroutine-exposing, (Π, Ψ)-compliant, $(\Pi, \mathsf{Mgmt}[\Pi, \mathcal{G}])$-compliant, and $(\Pi, \mathsf{Mgmt}[\Psi, \mathcal{G}])$-compliant.

(2) $\mathcal{G}$ is subroutine-respecting and Π-regular.

(3) Π and Ψ are $\mathcal{G}$-subroutine-respecting.

If $\mathsf{Mgmt}[\Pi, \mathcal{G}]$ UC-emulates $\mathsf{Mgmt}[\Psi, \mathcal{G}]$, then the composed protocol $\Phi^{\Psi \to \Pi}$ UC-emulates protocol Φ.

2.3.4 Reduction

To determine the conditions for securely realizing a functionality, we relate it to the corresponding (game-based) security notion via a *reduction*. As described in Section 2.2, game-based security involves interactions between an adversary $\mathcal{A}^*$, a game (or *challenger*) G, and several oracles $\mathcal{O}_1, \ldots, \mathcal{O}_{\mathsf{n}}$. In the reduction illustrated in Figure 2.2, we let $\mathcal{A}^*$ "simulate" the interaction between an environment $\mathcal{Z}$ and two worlds. Whenever $\mathcal{Z}$ provides input to a party $\mathcal{P}_{\mathsf{pid}}$ or the global subroutine $\mathcal{G}$, the adversary $\mathcal{A}^*$ queries the corresponding oracle to simulate the behavior of the respective ITI. The oracle's response is then written as the ITI's output.

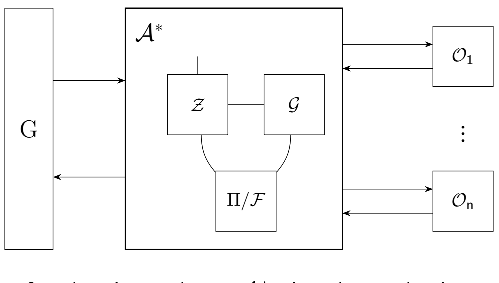

Figure 2.2: An illustration of reduction where $\mathcal{A}^*$ simulates the interaction between an environment $\mathcal{Z}$ with the two worlds of execution.

Ultimately, $\mathcal{Z}$ produces an output bit b, indicating whether it distinguishes between interacting with the protocol Π and the functionality $\mathcal{F}$ (or another protocol Ψ). The adversary $\mathcal{A}^*$ then uses the input-output tapes from this experiment (and possibly the bit b directly) to win the game G. If

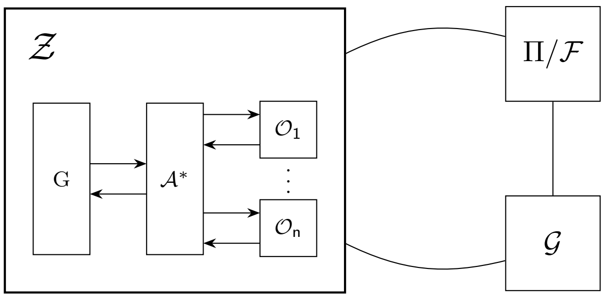

Figure 2.3: An illustration of reduction where the environment $\mathcal{Z}$ internally runs the game G and use the response from $\mathcal{A}^*$ to distinguish between Π and $\mathcal{F}$.

$\mathcal{Z}$ can distinguish between the two worlds, we construct $\mathcal{A}^*$ to win the game. By contrapositive, this shows that if protocol Π is secure in the game G, then Π UC-realizes the functionality $\mathcal{F}$.

With the above, we show the "if" direction of a security proof. For the "only if" direction, we need to show that we can construct an environment $\mathcal{Z}$ that uses the adversary $\mathcal{A}^*$ (that plays the security game) as a subroutine. For this, we let $\mathcal{Z}$ simulate the interaction between the adversary $\mathcal{A}^*$, the game G, and its oracles, as illustrated in Figure 2.3. Specifically, for each oracle query made by $\mathcal{A}^*$, the environment $\mathcal{Z}$ activates the corresponding parties with the inputs provided by the adversary $\mathcal{A}^*$ and answers $\mathcal{A}^*$'s query with the outputs from respective parties. Thus, we can then say if the protocol Π UC-realizes the functionality $\mathcal{F}$, then the protocol Π is secure with respect to the game G.

Chapter 3

Confidential Layer

In this chapter, we focus on the confidential layer of the system, with an emphasis on the discussion of Multi-Group Homomorphic Encryption (MGHE). In Section 3.1, we examine variants of homomorphic encryption in the multiparty setting, including Multi-Key Homomorphic Encryption and Threshold Homomorphic Encryption, highlighting their similarities and differences with MGHE. In Section 3.2, we revisit and introduce game-based security notions in the multiparty setting, addressing both client-side and server-side confidentiality. Finally, in Section 3.3, we present a composable perspective by defining the functionality for MGHE, which serves as the foundation for the system's confidential layer.

3.1 Homomorphic Encryption in Multi-Party Setting

In this section, we revisit the syntax of standard homomorphic encryption (HE) and its variant in multi-party settings including threshold HE, multi-key HE, and multi-group HE which is a hybrid construction of the above two schemes.

3.1.1 Syntax of Standard HE

Before transitioning to the multi-party setting, we first review the syntax of *Homomorphic Encryption* (HE) in the single-party setting. We define the syntax of an HE scheme in Definition 5. In our formalism for HE and its variants in multi-party settings, we use pk to represent both encryption and evaluation keys for simplicity of notation. In some literature, a separate evaluation key ek is explicitly defined in the syntax. Conceptually, we can interpret pk as $\mathsf{pk} = (\mathsf{pk}', \mathsf{ek})$, where pk' is as the public encryption key, since both pk' and ek are public.

Definition 5 (Homomorphic Encryption)**.** A *Homomorphic Encryption* scheme is a 4-tuple of probabilistic polynomial time (PPT) algorithms, denoted as $\mathsf{HE} = (\mathtt{KGen}, \mathtt{Enc}, \mathtt{Eval}, \mathtt{Dec})$, where

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{KGen}(1^\lambda)$: Given a public parameter λ as input, the algorithm outputs a public key pk and a secret key sk.
- $c \leftarrow \mathtt{Enc}(\mathsf{pk}, m)$: Given a public key pk and a plaintext message m as inputs, the algorithm outputs a ciphertext c.
- $\hat{c} \leftarrow \mathtt{Eval}(f, \mathsf{pk}, \{\mathsf{c}_j\}_{j\in[\ell]})$: Given a circuit f, a public key pk, and ℓ ciphertexts $c_1, \ldots, c_\ell$ as inputs, the algorithm outputs an evaluated ciphertext $\hat{c}$.
- $m \leftarrow \mathtt{Dec}(\mathsf{sk}, c)$: Given a ciphertext c and a secret key sk as inputs, the algorithm outputs a plaintext m.

Remark 1 (Random coins)**.** In Definition 5 (and subsequent definitions for HE variants in the multi-party setting), the value of the random coins used in these algorithms is not explicitly specified, and it is assumed to be sampled from a distribution defined by the scheme (e.g. Gaussian Distribution). In the following discussion, we also consider the case where the random coins is chosen by the adversary. To capture this, for example, we use the notations $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE}.\mathtt{KGen}(1^\lambda; r)$ and $c \leftarrow \mathsf{HE}.\mathtt{Enc}(\mathsf{pk}, m; \omega)$ to denote key generation or encryption with a specific random coin r.

3.1.2 Variants of HE in Multi-Party Setting

The extension of HE for multi-party functionality includes three variants: threshold HE, multi-key HE, and multi-group HE. In Figure 3.1, we illustrate how public and secret keys are used for encryption, evaluation, and decryption in each of these primitives, providing a high-level overview of how these schemes operate.

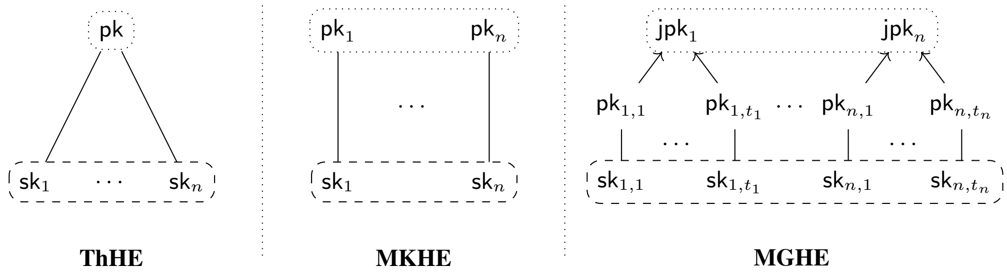

Figure 3.1: Key relationships in multi-key HE, threshold HE, and multi-group HE. The dotted boxes represent keys used for encryption and evaluation and the dashed boxes represent keys used for decryption. Observe that in MGHE, individual public keys are not used for encryption or evaluation compared to MKHE.

3.1.2.1 Threshold HE

In a *Threshold Homomorphic Encryption* (ThHE) scheme [AJL+12, BGG+18, MBH23, BS23], a group of n parties jointly executes a key generation protocol to establish a shared public key pk. Each party also receives an individual secret key share sk_i for $i \in [n]$. We assume that the scheme support *partial decryption*, where each party decrypts a ciphertext with its individual key. These partial decryptions are then combined together to reconstruct the message. We assume a (t, n)-threshold access structure [1], requiring the involvement of at least $t + 1$ parties to provide partial decryptions to reconstruct the message. In Definition 6, we formally define the syntax of a ThHE scheme.

Definition 6 (Threshold Homomorphic Encryption [BS23])**.** A (t, n)-*Threshold Homomorphic Encryption* scheme is a 5-tuple of probabilistic polynomial time (PPT) algorithms, denoted as $\mathsf{ThHE} = (\mathtt{Setup}, \mathtt{Enc}, \mathtt{Eval}, \mathtt{PDec}, \mathtt{Combine})$, where

– $(\mathsf{pp}, \mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathtt{Setup}(1^\lambda, 1^\kappa, n, t)$: Given a security parameter λ, a maximum circuit depth κ, the number of parties n, and a threshold value $t \in [n]$ as inputs, the algorithm outputs a public key pk and a set of secret key shares $\{\mathsf{sk}_i\}_{i\in[n]}$.

[1] We note that one can transform a (n, n)-threshold to a (t, n)-threshold one through the techniques like Shamir Secret Sharing [Sha79].

- $c \leftarrow \texttt{Enc}(\mathsf{pk}, m)$: Given a public key pk and a plaintext message m as inputs, the algorithm outputs a ciphertext c.
- $\hat{c} \leftarrow \texttt{Eval}(f, \mathsf{pk}, \{\mathsf{c}_j\}_{j\in[\ell]})$: Given a circuit f of depth at most κ, a public key pk, and ℓ ciphertexts $c_1, \ldots, c_\ell$ as inputs, the algorithm outputs an evaluated ciphertext $\hat{c}$.
- $m \leftarrow \texttt{DistDec}(\{\mathsf{sk}_i\}_{i\in S}, c)$: Given a ciphertext and a set of secret keys $\{\mathsf{sk}_i\}_{i\in S}$ for $S \subseteq [n]$ with $|S| \geq t$, the algorithm outputs a message m.
    - $d \leftarrow \texttt{PDec}(\mathsf{sk}_i, c)$: Given a ciphertext c and a secret key share sk_i where $i \in [n]$ as inputs, the algorithm outputs a partial-decryption d for the party.
    - $m \leftarrow \texttt{Combine}(c, \{d_i\}_{i\in S})$: Given a set of partial-decryption $\{d_i\}_{i\in S}$ for $S \subseteq [n]$ with $|S| \geq t$ as inputs, the algorithm outputs a message m.

3.1.2.2 Multi-Key HE

In a *Multi-key Homomorphic Encryption* (MKHE) scheme [LTV12, BP16, MW16, AJJM20], each party j independently generates its own key pair $(\mathsf{pk}_j, \mathsf{sk}_j)$ and encryption of data can be performed without interaction with other users. MKHE allows the evaluation of circuits on ciphertexts encrypted under different keys, producing a ciphertext that requires the secret keys of all participating parties for decryption. This design offers better flexibility by allowing new parties to join computations dynamically without needing to predefine the set of participants. However, the scalability of MKHE is limited, as its space and time complexity increases with the number of participants.

There are two families of MKHE schemes, distinguished by the decryption algorithm's structure. The first family [LTV12, BP16] employs *one-round decryption*, where each party performs a partial-decryption, and the final plaintext is reconstructed from these partial-decryptions. The second family [MW16, AJJM20] uses *unstructured decryption*, which requires all secret keys as inputs to decrypt the plaintext directly.

In Definition 7, we present the unstructured decryption variant to align with the syntax of ThHE. This definition also better reflects real-world applications, where it is not necessary for all parties to be simultaneously online during decryption. We also assume that the MKHE scheme is *multi-hop*, or more concisely, *re-usable*. This means that a ciphertext produced by the evaluation algorithm can be re-used in subsequent circuit evaluations without requiring decryption. For this feature, we consider an additional algorithm $\mathsf{MKHE}.\texttt{KJoin}(\cdot)$ which takes as input the public (evaluation) keys of all parties whose messages are involved in the evaluation, and generates a new public (evaluation) key for that ciphertext for the future rounds of evaluation. Several works [PS16, BP16, CZW17] have explicitly support this feature in their construction for MKHE.

Definition 7 (Multi-Key Homomorphic Encryption [AJJM20]). A *Multi-Key Homomorphic Encryption* scheme is a 6-tuple of probabilistic polynomial time (PPT) algorithms, denoted as $\mathsf{MKHE} = (\texttt{KGen}, \texttt{KJoin}, \texttt{Enc}, \texttt{Eval}, \texttt{PDec}, \texttt{Combine})$, where

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \texttt{KGen}(1^\lambda, 1^\kappa)$: Given a security parameter λ, a maximum circuit depth κ as inputs, the algorithm outputs a public key pk and a of secret key sk.
- $\hat{\mathsf{pk}} \leftarrow \texttt{KJoin}(\{\mathsf{pk}_i\}_{i\in[n]})$: Given a set of public keys $\{\mathsf{pk}_i\}_{i\in[n]}$ for n parties as inputs, the algorithm outputs a joint public key pk'.
- $c \leftarrow \texttt{Enc}(\mathsf{pk}, m)$: Given a public key pk and a plaintext message m as inputs, the algorithm outputs a ciphertext c.

- $\hat{c} \leftarrow \texttt{Eval}(f, (\mathsf{pk}_j, \mathsf{c}_j)_{j\in[\ell]})$: Given a circuit f of depth at most κ, ℓ ciphertexts $c_1, \ldots, c_\ell$, and their associated public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell$ as inputs, the algorithm outputs an evaluated ciphertext $\hat{c}$.
- $m \leftarrow \texttt{DistDec}(\{\mathsf{sk}_i\}_{i\in[\ell]}, c)$: Given a ciphertext and ℓ secret keys $\{\mathsf{sk}_i\}_{i\in[\ell]}$, the algorithm outputs a message m.
    - $d \leftarrow \texttt{PDec}(\mathsf{sk}, c)$: Given a ciphertext c and a secret key sk as inputs, the algorithm outputs a partial-decryption d of c under sk for the party.
    - $m \leftarrow \texttt{Combine}(c, \{d_i\}_{i\in[\ell]})$: Given a set of partial-decryptions $\{d_i\}_{i\in[\ell]}$ as inputs, the algorithm outputs a message m.

Remark 2. Several works on MKHE [MW16, BP16, CZW17] adopt a syntax where a ciphertext under a public key pk_i is first "expanded" using ℓ public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell$, and decryption subsequently requires the corresponding secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell$. For simplicity, we incorporate this operation into $\mathsf{MKHE}.\texttt{Eval}(\cdot)$ as a single step. We refer readers to these works for further details on their constructions.

Remark 3 (Keyswitching for ciphertext reusing)**.** We assume that ciphertexts generated by an MKHE scheme should be reusable in subsequent evaluations. This may require a *keyswitching* operation to transform an evaluated ciphertext into one encrypted under the aggregated key $\hat{\mathsf{pk}}$, which is generated using $\mathsf{MKHE}.\texttt{KJoin}(\cdot)$, as described by Cheon et al. [CCKY25]. For simplicity, we omit the explicit description of such a keyswitching algorithm and assume it is implicitly performed, allowing the use of pk' directly in the next round of evaluation.

3.1.2.3 Multi-Group HE

A *Multi-Group Homomorphic Encryption* (MGHE) scheme [AH19, MTBH21, Par21, CMS$^+$23, KLSW24, CCKY25] is a hybrid construction that combines the properties of multi-key and threshold HE. At a high level, a *group* of parties, through a ThHE scheme, acts as a single entity within an MKHE scheme.

In more detail, each party in the system independently generates its own key pair, $(\mathsf{pk}_i, \mathsf{sk}_i)$. Groups of arbitrary size can be formed at any time by generating a joint public key, jpk, for the group via a multi-party key generation protocol between the parties in the group. Encryption for a group j is then performed using this joint public key jpk_j. Evaluations on ciphertexts across different groups follow a similar approach to MKHE but use the joint public keys $\mathsf{jpk}_1, \ldots, \mathsf{jpk}_k$ instead. Additionally, a public (evaluation) key $\hat{\mathsf{jpk}}$ for the multi-hop evaluation can be derived similarly with $\mathsf{MGHE}.\texttt{KJoin}(\cdot)$ by running the key generation protocol for the new group in a similar manner. To decrypt, all secret keys of the parties involved in the computation are required.

MGHE improves MKHE on scalability by reducing the number of public keys involved and offers better flexibility than ThHE by allowing new parties to join the system dynamically.

Definition 8 (Multi-Group Homomorphic Encryption)**.** A *Multi-Group Homomorphic Encryption* scheme is a 7-tuple of probabilistic polynomial time (PPT) algorithms, denoted as $\mathsf{MGHE} = (\texttt{PGen}, \texttt{KGen}, \texttt{KJoin}, \texttt{Enc}, \texttt{Eval}, \texttt{PDec}, \texttt{Combine})$, where

- $\mathsf{pp} \leftarrow \texttt{PGen}(1^\lambda, 1^\kappa)$: Given a security parameter λ and a maximal circuit depth κ as inputs, the algorithm outputs a public parameter pp for the system.
- $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \texttt{KGen}(\mathsf{pp})$: Given a public parameter pp as input, the algorithm outputs a public key pk and a secret key sk for a party i.

– $\mathsf{jpk} \leftarrow \mathtt{KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{i\in I})$: Given key pairs $(\mathsf{pk}_i, \mathsf{sk}_i)_{i\in I}$ of parties whose indice are in I, the algorithm outputs a joint public key jpk for the group.

– $c \leftarrow \mathtt{Enc}(\mathsf{jpk}, m)$: Given a joint public key jpk and a plaintext message m as inputs, the algorithm outputs a ciphertext c.

– $\hat{c} \leftarrow \mathtt{Eval}(f, (\mathsf{jpk}_j, c_j)_{j\in[\ell]})$: Given a circuit f of depth at most κ, ℓ ciphertexts $c_1, \dots, c_\ell$, and their associated joint public keys $\mathsf{jpk}_1, \dots, \mathsf{jpk}_\ell$ as inputs, the algorithm outputs an evaluated ciphertext $\hat{c}$.

– $m \leftarrow \mathtt{DistDec}(\{\mathsf{sk}_i\}_{i\in I}, c)$: Given a ciphertext and the secret keys $\{\mathsf{sk}_i\}$ of parties in a group, the algorithm outputs a message m.

  - $d \leftarrow \mathtt{PDec}(\mathsf{sk}, c)$: Given a ciphertext c and a secret key of a party sk as inputs, the algorithm outputs a partial-decryption d of c under sk for the party.
  - $m \leftarrow \mathtt{Combine}(c, \{d_i\}_{i\in I})$: Given a set of partial-decryptions $\{d_i\}_{i\in I}$ for an index set I as inputs, the algorithm outputs a message m.

Remark 4. We define $\mathsf{MGHE}.\mathtt{KJoin}(\cdot)$ to take both public keys and secret keys as inputs. That is because, in most of the existing works [AJL+12, Par21, MTPBH21], the generation of the public *evaluation* key for a group would require the secret keys from each client in it (especially for the relinearization key due to its quadratic structure). Thus in practice, this key aggregation process should be done via a MPC protocol due to the involvement of secret keys. Recently, Kwak et al. [KLSW24] proposed a scheme where the public key is *nearly linear* with the corresponding secret key. In this case, the evaluation key can be generated in a fully *non-interactive* manner. To maintain generality, however, we adopt the definition of a multi-party protocol for key aggregation.

In Definition 9, we extend the correctness definition of MGHE by introducing a threshold factor t. Since MGHE allows dynamically formed groups of arbitrary size, we define correctness using a fractional threshold rather than a fixed number of clients for reconstruction. Specifically, if a ciphertext c is intended for a group of clients indexed by the set I, then partial decryptions from at least $\frac{|I|}{t}$ clients within I suffice to reconstruct the plaintext. In this thesis, the threshold factor t is assumed to be fixed before any computation and must be adhered to by all groups involved. In common cases, such as the scheme introduced by Kwak et al. [KLSW24], a threshold of $t = 1$ is typically assumed, meaning that decryption requires partial decryptions from all members of the group I_j when a message is encrypted with the public key of the group I_j. Similarly, for an evaluated ciphertext associated with a group $\hat{I} = I_1 \cup \dots \cup I_k$, partial decryptions from all members of $\hat{I}$ are necessary.

We note that this requirement can be achieved by carefully designing the *expanding* operation on each input ciphertext for evaluation, as discussed in Remark 2. We remark that this expanding operation may need secret sharing schemes that support dynamic access structures [Cac95, KNY16, TDB16, KP17].

In this thesis, we assume that the threshold factor remains fixed across different groups. However, in the case where each group has a distinct threshold, additional consensus before evaluation may be required to determine the threshold factor for the aggregated group. A detailed analysis of dynamically adjusting access structures is left as an open direction for future work.

Definition 9 (Correctness). Let $\mathsf{MGHE} = (\mathtt{PGen}, \mathtt{KGen}, \mathtt{Enc}, \mathtt{KJoin}, \mathtt{Eval}, \mathtt{PDec}, \mathtt{Combine})$ be a multi-group homomorphic encryption scheme and $\mathsf{pp} \leftarrow \mathsf{MGHE}.\mathtt{PGen}(1^\lambda, 1^\kappa)$ be the parameters. For $1 \le j \le \ell$, let $\mathsf{jpk}_j \leftarrow \mathsf{MGHE}.\mathtt{KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{i\in I_j})$ for an index set I_j and $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MGHE}.\mathtt{KGen}(\mathsf{pp})$ for each i. Let $c_j \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m_j)$. Let f be a circuit whose depth is bounded by κ, and let $\hat{c} = \mathsf{MGHE}.\mathtt{Eval}(f, (\mathsf{jpk}_j, c_j)_{j\in[\ell]})$. Let $\hat{I} = \cup_{j\in[\ell]} I_j$ and $d_i = \mathsf{MGHE}.\mathtt{PDec}(\mathsf{sk}_i, \hat{c})$ for $i \in \hat{I}$. Then MGHE has *ε-correctness* if for any set I such that $|I \cap \hat{I}| \geq \frac{|\hat{I}|}{t}$, it has

$$\Pr\left[\mathsf{MGHE}.\mathtt{Combine}(\{d_i\}_{i\in\hat{I}}, \hat{c}) = f(m_1, \ldots, m_\ell)\right] = 1 - \varepsilon.$$

We say MGHE has *perfect correctness* if $\varepsilon = 0$.

Note that schemes on approximate numbers, such as [CKKS17], cannot achieve correctness by design. However, they are considered "acceptable" if the distance between the decrypted value and the original message remains within an acceptable range. In Definition 10, we formalize this notion as *approximate correctness*.

Definition 10 (Approximate Correctness)**.** Let $\mathsf{MGHE} = (\mathtt{PGen}, \mathtt{KGen}, \mathtt{Enc}, \mathtt{KJoin}, \mathtt{Eval}, \mathtt{PDec}, \mathtt{Combine})$ be a multi-group homomorphic encryption scheme and $\mathsf{pp} \leftarrow \mathsf{MGHE}.\mathtt{PGen}(1^\lambda, 1^\kappa)$ be the parameters. For $1 \leq j \leq \ell$, let $\mathsf{jpk}_j \leftarrow \mathsf{MGHE}.\mathtt{KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{i\in I_j})$ for an index set I_j and $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MGHE}.\mathtt{KGen}(\mathsf{pp})$ for each i. Let $c_j \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m_j)$. Let f be a circuit whose depth is bounded by κ, and let $\hat{c} = \mathsf{MGHE}.\mathtt{Eval}(f, (\mathsf{jpk}_j, c_j)_{j\in[\ell]})$. Let $\hat{I} = \cup_{j\in[\ell]} I_j$ and $d_i = \mathsf{MGHE}.\mathtt{PDec}(\mathsf{sk}_i, \hat{c})$ for $i \in \hat{I}$. Then MGHE has *approximate correctness* if for any set I such that $|I \cap \hat{I}| \geq \frac{|\hat{I}|}{t}$, it has

$$||\mathsf{MGHE}.\mathtt{Combine}(\hat{c}, \{d_i\}_{i\in\hat{I}}) - f(m_1, \ldots, m_\ell)|| \leq \mathsf{Estimate}(f, t_1, \ldots, t_\ell)$$

where $\forall j \in [\ell]$, $t_j \leq ||\mathsf{MGHE}.\mathtt{Combine}(c_j, \{d_i\}_{i\in I_j}) - m_j||$ with $d_i = \mathsf{MGHE}.\mathtt{PDec}(\mathsf{sk}_i, c_j)$ for $i \in I_j$.

Beyond correctness, a homomorphic encryption scheme must also be compact. This means that the ciphertext length does not grow trivially with the number of homomorphic operations. Compactness ensures that the ciphertext remains manageable after evaluating a circuit, preventing it from becoming too large to transmit efficiently.

Definition 11 (Compactness)**.** A multi-group fully homomorphic encryption scheme is compact if the size of any evaluated ciphertext c, output by $\mathsf{MGHE}.\mathtt{Eval}(\cdot)$, depends only on the security parameter λ and the number of keys n. More formally, there exists a polynomial P such that $|c| \leq P(\lambda, n)$.

3.2 Security Notions for Confidential Layer

In this section, we revisit and propose some security notions defined for homomorphic encryption in the multi-party setting that are needed for realizing the ideal functionality of this primitive. Specifically, for client-side security, we consider the notions in the family of *semantic security*. For the server-side security, we consider variants of *circuit privacy*.

3.2.1 Semantic Security

For client-side confidentiality, we consider *semantic security*. At a high level, this means that the ciphertext from the client does reveal negligible information about its underlying plaintext. In this section, we first revisit the security notions defined for single-party HE and then propose variants for them for HE in the multi-party setting.

3.2.1.1 CPA-Security

We recall that the basic confidentiality guarantee for homomorphic encryption is formalized with *indistinguishability under chosen plaintext attack* (IND-CPA). We illustrate the IND-CPA game in Figure 3.2 and define the adversary's advantage in Definition 12.

```
Game: $\mathrm{G}^{\text{IND-CPA-}b}_{\mathsf{HE}}$

Procedure INIT
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE}.\mathtt{KGen}(1^\lambda)$

Procedure FIN
  $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}}(\mathsf{pk})$
  return $b'$

Oracle $\mathcal{O}_{\mathsf{Enc}}(m_0, m_1)$
  $c \leftarrow \mathsf{HE}.\mathtt{Enc}(\mathsf{pk}, m_b)$
  return $c$
```

Figure 3.2: IND-CPA game for a homomorphic encryption (HE) scheme.

Definition 12 (IND-CPA for HE)**.** A homomorphic encryption scheme HE is said to satisfy (q, t, ε)-*indistinguishability under chosen plaintext attack* (IND-CPA), if for any adversaries $\mathcal{A}$ that runs in time t and makes q queries to $\mathcal{O}_{\mathsf{Enc}}$, the advantage $\mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{HE}}(\mathcal{A}) \leq \varepsilon$ where

$$\mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{HE}}(\mathcal{A}) := \left|\Pr\left[\mathrm{G}^{\text{IND-CPA-0}}_{\mathsf{HE}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}^{\text{IND-CPA-1}}_{\mathsf{HE}}(\mathcal{A}) \Rightarrow 1\right]\right|.$$

Kwak et al. [KLSW24] later extended this notion to multi-group setting by allowing the adversary to corrupt a subset of parties. For completeness, we illustrate the security game in Figure 3.3 and define the adversary's advantage in Definition 13.

```
Game: $\mathrm{G}^{\text{IND-CPA-}b}_{\mathsf{MGHE}}$

Procedure INIT
  $\mathsf{pp} \leftarrow \mathsf{MGHE}.\mathtt{PGen}(1^\lambda, 1^\kappa)$
  $(\{\gamma_i\}_{i \in I_\mathcal{A}}), I_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{pp})$ // $I_\mathcal{A} \subset [n]$
  for $i \in [n] \setminus I_\mathcal{A}$ do
    $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MGHE}.\mathtt{KGen}(\mathsf{pp})$
  for $i \in I_\mathcal{A}$ do
    $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MGHE}.\mathtt{KGen}(\mathsf{pp}; \gamma_i)$
  $\{I_1, \ldots, I_k\} \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{i \in [n]}, \{\mathsf{sk}_i\}_{i \in I_\mathcal{A}})$
  for $j \in [k]$ do
    $\mathsf{jpk}_j \leftarrow \mathsf{MGHE}.\mathtt{KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{i \in I_j})$

Oracle $\mathcal{O}_{\mathsf{Enc}}(\mathsf{jpk}_j, m_0, m_1)$
  if $|I_j \cap I_\mathcal{A}| \geq |I_j|/t$ then
    return $\perp$
  $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m_b)$
  $T := T \bowtie (I_j, \mathsf{jpk}_j, m_0, m_1, c)$
  return $c$

Procedure FIN
  $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}}(\{\mathsf{pk}_i\}_{i \in [n]}, \{\mathsf{jpk}_j\}_{j \in [k]}, \{\mathsf{sk}_i\}_{i \in I_\mathcal{A}})$
  return $b'$
```

Figure 3.3: IND-CPA game for a multi-group homomorphic encryption (MGHE) scheme by Kwak et al. [KLSW24].

Definition 13 (IND-CPA for MGHE)**.** A multi-group homomorphic encryption scheme MGHE is said to satisfy (q, t, ε)-*indistinguishability under chosen plaintext attack* (IND-CPA), if for any adversaries $\mathcal{A}$ that runs in time t and makes q queries to $\mathcal{O}_{\mathsf{Enc}}$, the advantage $\mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{MGHE}}(\mathcal{A}) \leq \varepsilon$ where

$$\mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{MGHE}}(\mathcal{A}) := \left|\Pr\left[\mathrm{G}^{\text{IND-CPA-0}}_{\mathsf{MGHE}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}^{\text{IND-CPA-1}}_{\mathsf{MGHE}}(\mathcal{A}) \Rightarrow 1\right]\right|.$$

3.2.1.2 Security under Decryption Disclosure

Li and Micciancio [LM21] later showed that IND-CPA might be inadequate for *approximate* homomorphic encryption schemes, where revealing decrypted data could lead to fully recovery of secret key. This is especially important in our case since an adversary may learn the decrypted message through a corrupted player in a MPC system. To strengthen security in these cases, they proposed a stronger notion of IND-CPA$^{\mathrm{D}}$.

In IND-CPA$^{\mathrm{D}}$, the adversary is given access to a *restricted* evaluation oracle $\mathcal{O}_{\mathsf{Eval}}$, which evaluates only those ciphertexts produced by either the encryption oracle $\mathcal{O}_{\mathsf{Enc}}$ or itself, $\mathcal{O}_{\mathsf{Eval}}$. Additionally, the adversary is provided with a *restricted* decryption oracle $\mathcal{O}_{\mathsf{Dec}}$, which decrypts only ciphertexts whose underlying plaintexts are independent of the challenge bit. These restrictions are enforced using a lookup table T, where each entry has the structure (m_0, m_1, c), with m_0 and m_1 being the challenge plaintexts and c the corresponding ciphertext. Queries to $\mathcal{O}_{\mathsf{Eval}}$ and $\mathcal{O}_{\mathsf{Dec}}$ are processed by referencing an index in T,[2] and decryption is permitted only if $m_0 = m_1$ for the selected index.

In Figure 3.4, we illustrate the IND-CPA$^{\mathrm{D}}$ game for a homomorphic encryption scheme HE. Notably, the adversary is allowed to make *adaptive* oracle queries for stronger security. IND-CPA$^{\mathrm{D}}$ also includes a *non-adaptive* variant, and a variant when the number of queries to $\mathcal{O}_{\mathsf{Dec}}$ is limited.

Game: $\mathrm{G}_{\mathsf{HE}}^{\text{IND-CPA}^{\mathrm{D}}\text{-}b}$

```
Procedure INIT
1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE.KGen}(1^\lambda)$
2 : $T := [\,]$
3 : // Elements in $T$ have structure $(m_0, m_1, c)$.

Oracle $\mathcal{O}_{\mathsf{Eval}}(f, L)$
1 : $\hat{c} \leftarrow \mathsf{HE.Eval}(f, \mathsf{pk}, (T[l].c)_{l \in L})$
2 : $\hat{m}_0 := f((T[l].m_0)_{l \in L})$
3 : $\hat{m}_1 := f((T[l].m_1)_{l \in L})$
4 : $T := T \bowtie (\hat{m}_0, \hat{m}_1, \hat{c})$
5 : return $\hat{c}$

Oracle $\mathcal{O}_{\mathsf{Enc}}(m_0, m_1)$
1 : $c \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, m_b)$
2 : $T := T \bowtie (m_0, m_1, c)$
3 : return $c$

Oracle $\mathcal{O}_{\mathsf{Dec}}(l)$
1 : if $T[l].m_0 = T[l].m_1$ then
2 :     $m \leftarrow \mathsf{HE.Dec}(\mathsf{sk}, T[l].c)$
3 :     return $m$

Procedure FIN
1 : $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\{\mathsf{Enc},\mathsf{Eval},\mathsf{Dec}\}}}(\mathsf{pk})$
2 : return $b'$
```

Figure 3.4: IND-CPA$^{\mathrm{D}}$ games for a homomorphic encryption (HE) scheme introduced by Li and Micciancio [LM21].

Definition 14 (IND-CPA$^{\mathrm{D}}$)**.** A homomorphic encryption scheme HE is said to satisfy (q_e, q_v, q_d, t,ε)*-indistinguishability under chosen plaintext attack with decryption query* (IND-CPA$^{\mathrm{D}}$), if for any adversaries $\mathcal{A}$ running in time t, making q_e queries to $\mathcal{O}_{\mathsf{Enc}}$, q_v queries to $\mathcal{O}_{\mathsf{Eval}}$, and q_d queries to $\mathcal{O}_{\mathsf{Dec}}$, the advantage $\mathbf{Adv}_{\mathsf{HE}}^{\text{IND-CPA}^{\mathrm{D}}}(\mathcal{A}) \leq \varepsilon$ where

$$\mathbf{Adv}_{\mathsf{HE}}^{\text{IND-CPA}^{\mathrm{D}}}(\mathcal{A}) := \left| \Pr\left[\mathrm{G}_{\mathsf{HE}}^{\text{IND-CPA}^{\mathrm{D}}\text{-}0}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}_{\mathsf{HE}}^{\text{IND-CPA}^{\mathrm{D}}\text{-}1}(\mathcal{A}) \Rightarrow 1\right] \right|.$$

[2]We remark that it is implicitly assumed that if $T[l] = \perp$ for some index l, then the oracle returns $\perp$ to indicate failure.

Li and Micciancio [LM21] identified the cause of this vulnerability as the use of *approximate* homomorphic evaluation in the scheme. To address this, *noise-flooding* [3] techniques have been proposed as a practical solution [BBB$^+$22], where noise [4] (drawn from a Gaussian distribution) is added to the decrypted message before it is revealed to the adversary to "flush out" the error raised from approximate evaluation. Li et al. [LMSS22] further demonstrated that the level of noise-flooding must be sufficiently high to account for the *worst-case* noise.

Subsequently, Cheon et al. [CCP$^+$24] clarified that IND-CPA$^{\mathrm{D}}$ attacks stem not from the use of approximate and exact evaluation but rather from issues related to decryption failures and incorrectness. Intuitively, all FHE ciphertexts retain some level of noise, which propagates through evaluation. However, many schemes and implementations rely on heuristic assumptions about noise growth. If the noise is incorrectly estimated, a ciphertext may decrypt to a different message that enable such attacks. Additionally, many schemes employ bootstrapping to reduce noise, yet bootstrapping has a heuristic failure probability. Cheon et al. demonstrated attacks on B/FV [FV12, Bra12], an *exact* homomorphic encryption scheme, by exploiting both incorrect decryptions and bootstrapping failures. Their attacks leverage correlations among ciphertexts and exploit the fact that noise is estimated using *average-case* noise analysis, leading to incorrect decryption, and bootstrapping failures that provide "approximate inequality hints" on the secret key.

Given that multiple attacks [AV21, CCP$^+$24, GNSJ24, CSBB24] have been proposed against CPA-secure exact homomorphic encryption, in Lemma 1, we state only the direction where IND-CPA$^{\mathrm{D}}$ implies IND-CPA, as this direction holds for both approximate and exact HE schemes.

Lemma 1. (IND-CPA$^{\mathrm{D}}$ $\mapsto$ IND-CPA) *For any homomorphic encryption scheme* HE*, if* HE *is* IND-CPA$^{\mathrm{D}}$ *secure, then it is* IND-CPA *secure.*

To model security against the attacks described above, Bergamaschi et al. [BCD$^+$24] introduced a *semi-honest* variant, IND-CPA$^{\mathrm{DSH}}$, which allows the adversary access to the internal random coins used by the system. The structure of each entry in the lookup table T is extended to $(m_0, m_1, c, \rho, d, u)$, where ρ represents the random coins used during encryption, d indicates the decryption of a ciphertext, and u specifies whether the ciphertext has been used in an evaluation. In the finalization phase, the adversary is granted *one-time* access to the full contents of T, with no additional oracle queries allowed afterward. Achieving IND-CPA$^{\mathrm{DSH}}$ security enables scaling the noise-flooding level based on average-case noise rather than worst-case noise, effectively mitigating the described attacks. Figure 3.5 provides an illustration of the IND-CPA$^{\mathrm{DSH}}$ game.

Definition 15 (IND-CPA$^{\mathrm{DSH}}$)**.** A homomorphic encryption scheme HE is said to satisfy (q_e, q_v, q_d, t, ε)*-semi-honest indistinguishability under chosen plaintext attack with decryption query* (IND-CPA$^{\mathrm{DSH}}$), if for any adversaries $\mathcal{A}$ that runs in time t, makes q_e queries to $\mathcal{O}_{\mathsf{Enc}}$, q_v queries to $\mathcal{O}_{\mathsf{Eval}}$, and q_d queries to $\mathcal{O}_{\mathsf{Dec}}$, the advantage $\mathbf{Adv}_{\mathsf{HE}}^{\text{IND-CPA}^{\text{DSH}}}(\mathcal{A}) \leq \varepsilon$ where

$$\mathbf{Adv}_{\mathsf{HE}}^{\text{IND-CPA}^{\text{DSH}}}(\mathcal{A}) := \left|\Pr\left[\mathrm{G}_{\mathsf{HE}}^{\text{IND-CPA}^{\text{DSH}}\text{-0}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}_{\mathsf{HE}}^{\text{IND-CPA}^{\text{DSH}}\text{-1}}(\mathcal{A}) \Rightarrow 1\right]\right|.$$

Lemma 2. (IND-CPA$^{\mathrm{DSH}}$ $\mapsto$ IND-CPA) *For any homomorphic encryption scheme* HE*, if* HE *is* IND-CPA$^{\mathrm{DSH}}$ *secure, then it is* IND-CPA$^{\mathrm{D}}$ *secure.*

[3]We remark that this technique is also called *noise-smudging* when it is applied in the threshold setting as in previous works [AJL$^+$12, MBH23, MW16].

[4]We may interchangeably use the word "noise" and "error"

Game: $\mathrm{G}_{\mathsf{HE}}^{\text{IND-CPA}^{\text{DSH}}\text{-}b}$

Procedure INIT
```
1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE.KGen}(1^\lambda)$
2 : $\mathcal{T} := [\,]$
3 : //Elements in $\mathcal{T}$ have structure $(m_0, m_1, c, \omega, d, u)$.
```

Oracle $\mathcal{O}_{\mathsf{Eval}}(f, L)$
```
1 : $\hat{c} \leftarrow \mathsf{HE.Eval}(f, \mathsf{pk}, (\mathcal{T}[l].c)_{l \in L})$
2 : $\hat{m}_0 := f((\mathcal{T}[l].m_0)_{l \in L})$
3 : $\hat{m}_1 := f((\mathcal{T}[l].m_1)_{l \in L})$
4 : $\mathcal{T} := \mathcal{T} \bowtie (\hat{m}_0, \hat{m}_1, \hat{c}, \perp, \perp, 1)$
5 : for $l \in L$ then
6 :     $\mathcal{T}[l].u := 1$
7 : return $\hat{c}$
```

Procedure FIN
```
1 : $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\{\mathsf{Enc},\mathsf{Eval},\mathsf{Dec}\}}}(\mathsf{pk}, \mathcal{T})$
2 : return $b'$
```

Oracle $\mathcal{O}_{\mathsf{Enc}}(m_0, m_1)$
```
1 : $\omega \leftarrow_{\mathcal{D}} \Omega$
2 : $c \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, m_b; \omega)$
3 : if $m_0 = m_1$ then
4 :     $\mathcal{T} := \mathcal{T} \bowtie (m_0, m_1, c, \omega, \perp, 0)$
5 : else
6 :     $\mathcal{T} := \mathcal{T} \bowtie (m_0, m_1, c, \perp, \perp, 0)$
7 : return $c$
```

Oracle $\mathcal{O}_{\mathsf{Dec}}(l)$
```
1 : if $\mathcal{T}[l].m_0 = \mathcal{T}[l].m_1$ then
2 :     $d \leftarrow \mathsf{HE.Dec}(\mathsf{sk}, \mathcal{T}[l].c)$
3 :     $\mathcal{T}[l].d := d$
4 :     return $d$
```

Figure 3.5: IND-CPA$^{\text{DSH}}$ game for a homomorphic encryption (HE) scheme introduced by Bergamaschi et al. [BCD$^+$24].

Proof. It is trivial to see that given an IND-CPA$^{\text{D}}$ adversary $\mathcal{A}$, we can construct an IND-CPA$^{\text{DSH}}$ adversary $\mathcal{B}$ by forwarding each query from $\mathcal{A}$ to $\mathcal{B}$'s oracles. Finally, let $\mathcal{B}$ return the same bit b' returned by $\mathcal{A}$. Thus we have that $\mathbf{Adv}_{\mathsf{HE}}^{\text{IND-CPA}^{\text{D}}}(\mathcal{A}) \leq \mathbf{Adv}_{\mathsf{HE}}^{\text{IND-CPA}^{\text{DSH}}}(\mathcal{B})$. □

Also from the perspective of random coins, Bernard et al. [BJSW24] argued that IND-CPA$^{\text{D}}$ is weak as the definition of $\mathcal{O}_{\mathsf{Enc}}$ restricts the adversary to submitting ciphertexts generated with "honest" random coins. However, in practice, an adversary can select arbitrary random coins. Indeed, we can use zero-knowledge proof to prove the well-formedness of ciphertext with respect to some random coins, yet it is expensive to show that the value is sampled honestly following a specific distribution. To address this limitation, Bernard et al. [BJSW24] proposed a stronger variant, sIND-CPA$^{\text{D}}$, which provides the adversary with an additional (non-challenging) encryption oracle, $\mathcal{O}_{\mathsf{Enc}^*}$, allowing encryption with adversarially chosen random coins.

Definition 16 (sIND-CPA$^{\text{D}}$)**.** A homomorphic encryption scheme HE is said to satisfy (q_e, q_{e^*}, q_v, q_d, t, ε)-*indistinguishibility under chosen plaintext attack with decryption query* (IND-CPA$^{\text{D}}$), if for any adversaries $\mathcal{A}$ that runs in time t, makes q_e queries to $\mathcal{O}_{\mathsf{Enc}}$, q_{e^*} queries to $\mathcal{O}_{\mathsf{Enc}^*}$, q_v queries to $\mathcal{O}_{\mathsf{Eval}}$, and q_d queries to $\mathcal{O}_{\mathsf{Dec}}$, the advantage $\mathbf{Adv}_{\mathsf{HE}}^{\text{sIND-CPA}^{\text{D}}}(\mathcal{A}) \leq \varepsilon$ where

$$\mathbf{Adv}_{\mathsf{HE}}^{\text{sIND-CPA}^{\text{D}}}(\mathcal{A}) := \left|\Pr\left[\mathrm{G}_{\mathsf{HE}}^{s\text{IND-CPA}^{\text{D}}\text{-}0}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}_{\mathsf{HE}}^{s\text{IND-CPA}^{\text{D}}\text{-}1}(\mathcal{A}) \Rightarrow 1\right]\right|.$$

Note that sIND-CPA$^{\text{D}}$ considers the security under *semi-malicious* adversary. Thus it is stronger than IND-CPA$^{\text{DSH}}$ which considers a semi-honest adversary. In Lemma 3, we show the implication between these two notions.

Game: $\mathrm{G}_{\mathsf{HE}}^{s\text{IND-CPA}^{\mathrm{D}}\text{-}b}$

Procedure INIT
```
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE.KGen}(1^\lambda)$
$T := [\,]$
// Elements in $T$ have structure $(m_0, m_1, c)$.
```

Oracle $\mathcal{O}_{\mathsf{Eval}}(f, L)$
```
$\hat{c} \leftarrow \mathsf{HE.Eval}(f, \mathsf{pk}, (T[l].c)_{l \in L})$
$\hat{m}_0 := f((T[l].m_0)_{l \in L})$
$\hat{m}_1 := f((T[l].m_1)_{l \in L})$
$T := T \bowtie (\hat{m}_0, \hat{m}_1, \hat{c})$
return $\hat{c}$
```

Procedure FIN
```
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\{\mathsf{Enc}, \mathsf{Enc}^*, \mathsf{Eval}, \mathsf{Dec}\}}}(\mathsf{pk})$
return $b'$
```

Oracle $\mathcal{O}_{\mathsf{Enc}}(m_0, m_1)$
```
$c \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, m_b)$
$T := T \bowtie (m_0, m_1, c)$
return $c$
```

Oracle $\mathcal{O}_{\mathsf{Enc}^*}(m; \omega)$
```
$c \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, m; \omega)$
$T := T \bowtie (m, m, c)$
return $c$
```

Oracle $\mathcal{O}_{\mathsf{Dec}}(l)$
```
if $T[l].m_0 = T[l].m_1$ then
    $m \leftarrow \mathsf{HE.Dec}(\mathsf{sk}, T[l].c)$
    return $m$
```

Figure 3.6: sIND-CPA$^{\mathrm{D}}$ games for a homomorphic encryption (HE) scheme introduced by Bernard et al. [BJSW24].

Lemma 3. (sIND-CPA$^{\mathrm{D}}$ $\mapsto$ IND-CPA$^{\mathrm{DSH}}$) *For any homomorphic encryption scheme* HE*, if* HE *is* sIND-CPA$^{\mathrm{D}}$ *secure, then it is* IND-CPA$^{\mathrm{DSH}}$ *secure.*

Proof. We show that for each IND-CPA$^{\mathrm{DSH}}$ adversary, we can construct an sIND-CPA$^{\mathrm{D}}$ adversary $\mathcal{B}$ as follows. We let $\mathcal{B}$ maintain a transcript T from empty. Then $\mathcal{B}$ handles the queries from $\mathcal{A}$ as follows.

– **On the Query** (m_0, m_1) **to** $\mathcal{O}_{\mathsf{Enc}}$ **from** $\mathcal{A}$: If $m_0 \neq m_1$, then $\mathcal{B}$ forwards the query to its oracle $\mathcal{O}_{\mathsf{Enc}}$ to get c, and append T with $(m_0, m_1, c, \perp, \perp, 0)$. Otherwise, $\mathcal{B}$ first samples $r \leftarrow\!\!\$\ \mathcal{R}$ and queries it oracle $\mathcal{O}_{\mathsf{Enc}^*}$ with $(m; \omega)$. Then $\mathcal{B}$ appends T with $(m_0, m_1, c, r, \perp, 0)$ and returns c to $\mathcal{A}$.

– **On the Query** (f, L) **to** $\mathcal{O}_{\mathsf{Eval}}$ **from** $\mathcal{A}$: We let $\mathcal{B}$ first computes $\hat{m}_0 = f(\{T[l].m_0\}_{l \in L})$ and $\hat{m}_1 = f(\{T[l].m_1\}_{l \in L})$. Then $\mathcal{B}$ queries it oracle $\mathcal{O}_{\mathsf{Eval}}$ to get $\hat{c}$. Then $\mathcal{B}$ appends T with a new tuple $(\hat{m}_0, \hat{m}_1, \hat{c}, \perp, \perp, 1)$ and modifies $T[l].u$ to 1 for each $l \in L$. Then $\mathcal{B}$ returns $\hat{c}$ to $\mathcal{A}$.

– **On the Query** l **to** $\mathcal{O}_{\mathsf{Dec}}$ **from** $\mathcal{A}$: We let $\mathcal{B}$ forward this query to its oracle $\mathcal{O}_{\mathsf{Dec}}$. If $\mathcal{B}$ receives decryption m from $\mathcal{O}_{\mathsf{Dec}}$, then it sets $T[l].d$ as m and returns m to $\mathcal{A}$.

At a state, $\mathcal{A}$ inputs the transcript T, we then let $\mathcal{B}$ returns T to $\mathcal{A}$ and no longer accept any further query from $\mathcal{A}$. Eventually, $\mathcal{A}$ returns a bit b' and we let $\mathcal{B}$ return the same bit. Thus we have that $\mathbf{Adv}_{\mathsf{HE}}^{\text{IND-CPA}^{\mathrm{DSH}}}(\mathcal{A}) \leq \mathbf{Adv}_{\mathsf{HE}}^{\text{sIND-CPA}^{\mathrm{D}}}(\mathcal{B})$. □

3.2.1.3 Threshold Security

Several works [BS23, CCP+24] suggest that IND-CPA$^{\mathrm{D}}$ can be viewed as the simplest form of threshold security, given that partial-decryption is possible in threshold homomorphic encryption.

We consider a (t, n)-threshold scheme, where inputs from at least t out of n parties are needed to reconstruct the secret.

Game: $\mathrm{G}^{\text{IND-CPA}^{\text{pD}}\text{-}b}_{\mathsf{ThHE}}$

Procedure Init
```
$(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{ThHE.Setup}(1^\lambda, 1^\kappa, n, t)$
$I_\mathcal{A} \leftarrow \mathcal{A}(\mathsf{pk})$
$/\!/ I_\mathcal{A} \subset [n] \wedge |I_\mathcal{A}| < t$
$\mathcal{T} := [\,]$
$/\!/$Elements in $\mathcal{T}$ have structure $(m_0, m_1, c)$.
```

Oracle $\mathcal{O}_{\mathsf{Eval}}(f, L; \sigma)$
```
$\hat{c} \leftarrow \mathsf{MGHE.Eval}(f, \mathsf{pk}, \{\mathcal{T}[l].c\}_{l\in L}; \sigma)$
$\hat{m}_0 := f((\mathcal{T}[l].m_0)_{l\in L})$
$\hat{m}_1 := f((\mathcal{T}[l].m_1)_{l\in L})$
$\mathcal{T} := \mathcal{T} \bowtie (\hat{m}_0, \hat{m}_1, \hat{c})$
return $\hat{c}$
```

Oracle $\mathcal{O}_{\mathsf{Enc}}(m_0, m_1)$
```
$c \leftarrow \mathsf{ThHE.Enc}(\mathsf{pk}, m_b)$
$\mathcal{T} := \mathcal{T} \bowtie (m_0, m_1, c)$
return $c$
```

Oracle $\mathcal{O}_{\mathsf{Enc}^*}(m; \omega)$
```
$c \leftarrow \mathsf{ThHE.Enc}(\mathsf{pk}, m; \omega)$
$\mathcal{T} := \mathcal{T} \bowtie (m, m, c)$
return $c$
```

Oracle $\mathcal{O}_{\mathsf{Dec}}(l)$
```
if $\mathcal{T}[l].m_0 = \mathcal{T}[l].m_1$ then
    for $i \in \mathcal{T}[l].I$ do
        $d_i \leftarrow \mathsf{ThHE.PDec}(\mathsf{sk}_i\ \mathcal{T}[l].c)$
    return $\{d_i\}_{i\in\mathcal{T}[l].I}$
```

Procedure Fin
```
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\{\mathsf{Enc},\mathsf{Enc}^*,\mathsf{Eval},\mathsf{Dec}\}}}(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in I_\mathcal{A}})$
return $b'$
```

Figure 3.7: IND-CPA$^{\text{pD}}$ game for a (t, n)-threshold homomorphic encryption (ThHE) scheme.

During the initialization phase of the security game, the adversary $\mathcal{A}$ selects an index set $I_\mathcal{A} \subset [n]$ *non-adaptively* such that $|I_\mathcal{A}| < t$.[5] The parties indexed by $I_\mathcal{A}$ are then corrupted by $\mathcal{A}$, who gains access to their secret keys $\{\mathsf{sk}_i\}_{i\in I_\mathcal{A}}$.

Boudgoust and Scholl [BS23] first extended the IND-CPA$^{\text{D}}$ notion to a threshold setting by revealing the partial decryptions instead. However, as pointed out by Passelègue and Damien Stehlé [PS24], the security notion introduced in [BS23] does not account for the disclosure of the evaluated ciphertext, whereas in practice, each party must know the ciphertext first in order to (partially) decrypt it. Furthermore, they highlighted that, in a multi-party setting, the corruption of the party who evaluates the circuit could lead to the exposure of internal random coins used during the randomized evaluation, potentially causing a security breach.

We combine the ideas from these two works and introduce the security notion of IND-CPA$^{\text{pD}}$ i.e., CPA-security with partial-decryption. We also include the oracle $\mathcal{O}_{\mathsf{Enc}^*}$ for a stronger security against a semi-malicious adversary. To account for the case where the evaluator is also corrupted, we also let the adversary select the random coin at its query to the oracle $\mathcal{O}_{\mathsf{Eval}}$. We then illustrate the IND-CPA$^{\text{pD}}$ game for threshold HE in Figure 3.7 and define the adversary's advantage in Definition 17.

Definition 17 (IND-CPA$^{\text{pD}}$ for ThHE)**.** A threshold homomorphic encryption scheme ThHE is

[5]Some other works [BGG$^+$18, KS23] introduction a monotone access structure in the definition. For simplicity and clarity, we use client indices instead.

said to satisfy $(q_e, q_v, q_d, t, n, \varepsilon)$-*indistinguishibility under chosen plaintext attack with partial-decryption* (IND-CPA$^{\text{pD}}$), if for any adversaries $\mathcal{A}$ that runs in time t, makes q_e queries to $\mathcal{O}_{\mathsf{Enc}}$, q_v queries to $\mathcal{O}_{\mathsf{Eval}}$, q_d queries to $\mathcal{O}_{\mathsf{Dec}}$, and corrupts n parties, the advantage $\mathbf{Adv}^{\text{IND-CPA}^{\text{pD}}}_{\mathsf{ThHE}}(\mathcal{A}) \leq \varepsilon$ where

$$\mathbf{Adv}^{\text{IND-CPA}^{\text{pD}}}_{\mathsf{ThHE}}(\mathcal{A}) := \left|\Pr\left[\mathrm{G}^{\text{IND-CPA}^{\text{pD}}\text{-}0}_{\mathsf{ThHE}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}^{\text{IND-CPA}^{\text{pD}}\text{-}1}_{\mathsf{ThHE}}(\mathcal{A}) \Rightarrow 1\right]\right|.$$

3.2.1.4 Multi-Key Security

We next examine the security when multiple encryption keys are used. To our knowledge, most existing works on MKHE [LTV12, BP16, MW16] primarily address achieving semantic security but do not consider security in presence of partial-decryptions. Recently, Kluczniak and Santato [KS23] introduced a notion that allows partial-decryption disclosures; however, this approach is limited by its non-adaptive nature.

Game: $\mathsf{G}^{\text{IND-CPA}^{\text{pD}}\text{-}b}_{\mathsf{MKHE}}$

```
Procedure INIT
  $(\{\gamma_i\}_{i \in I_\mathcal{A}}, I_\mathcal{A}) \leftarrow \mathcal{A}(\cdot)$ // $I_\mathcal{A} \subset [n]$
  for $i \in [n] \setminus I_\mathcal{A}$ do
    $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MKHE.KGen}(1^\lambda, 1^\kappa)$
  for $i \in I_\mathcal{A}$ do
    $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MKHE.KGen}(1^\lambda, 1^\kappa; \gamma_i)$
  $T := []$
  // Elements in $T$ have structure $(I, \mathsf{pk}, m_0, m_1, c)$.

Oracle $\mathcal{O}_{\mathsf{Eval}}(f, L; \sigma)$
  $\hat{I} := \cup_{l \in L} T[l].I$
  if $\hat{I} \subseteq I_\mathcal{A}$ then
    return $\perp$
  $\hat{c} \leftarrow \mathsf{MKHE.Eval}(f, (T[l].\mathsf{pk}, T[l].c)_{l \in L}; \sigma)$
  $\hat{m}_0 := f((T[l].m_0)_{l \in L})$
  $\hat{m}_1 := f((T[l].m_1)_{l \in L})$
  $\hat{\mathsf{pk}} \leftarrow \mathsf{MKHE.KJoin}(\{\mathsf{pk}_i\}_{i \in \hat{I}})$
  $T := T \bowtie (\hat{I}, \hat{\mathsf{pk}}, \hat{m}_0, \hat{m}_1, \hat{c})$
  return $\hat{c}$

Oracle $\mathcal{O}_{\mathsf{Enc}}(\mathsf{pk}_i, m_0, m_1)$
  if $i \in I_\mathcal{A}$ then
    return $\perp$
  $c \leftarrow \mathsf{MKHE.Enc}(\mathsf{pk}_i, m_b)$
  $T := T \bowtie (\{i\}, \mathsf{pk}_i, m_0, m_1, c)$
  return $c$

Oracle $\mathcal{O}_{\mathsf{Enc}^*}(\mathsf{pk}_i, m; \omega)$
  $c \leftarrow \mathsf{MKHE.Enc}(\mathsf{pk}_i, m; \omega)$
  $T := T \bowtie (\{i\}, \mathsf{pk}_i, m, m, c)$
  return $c$

Oracle $\mathcal{O}_{\mathsf{Dec}}(l)$
  if $T[l].m_0 = T[l].m_1$ then
    for $i \in T[l].I$ do
      $d_i \leftarrow \mathsf{MKHE.PDec}(\mathsf{sk}_i\ T[l].c)$
    return $\{d_i\}_{i \in T[l].I}$

Procedure FIN
  $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\{\mathsf{Enc},\mathsf{Enc}^*,\mathsf{Eval},\mathsf{Dec}\}}}(\{\mathsf{pk}\}_{i \in [n]}, \{\mathsf{sk}_i\}_{i \in I_\mathcal{A}})$
  return $b'$
```

Figure 3.8: IND-CPA$^{\text{pD}}$ game for a multi-key homomorphic encryption (MKHE) scheme.

In this thesis, we propose an adaptive security notion for MKHE, modeled on IND-CPA$^{\text{D}}$ security. In our setting, we assume a system with n parties, where the adversary $\mathcal{A}$ can designate a subset $I_{\mathcal{A}} \subset [n]$ of parties as corrupted. We require that the system remains secure as long as at least one party involved in the computation remains uncorrupted.

In Figure 3.8, we use a transcript $\mathcal{T}$ in which each element is structured as $(I, \mathsf{pk}, m_0, m_1, c)$, where I represents the set of indices of parties whose public keys are used for encryption or whose messages are involved in an evaluation, and pk denotes the corresponding public key used in encryption or evaluation. This helps us to track which public keys are to be used for evaluation, and when to apply which secret keys for partial-decryption.

Definition 18 (IND-CPA$^{\mathrm{pD}}$ for MKHE)**.** A multi-group homomorphic encryption scheme MGHE is said to satisfy $(q_e, q_v, q_d, t, n, \varepsilon)$*-indistinguishability under chosen plaintext attack with decryption query* (IND-CPA$^{\mathrm{pD}}$), if for any adversaries $\mathcal{A}$ that runs in time t, makes q_e queries to $\mathcal{O}_{\mathsf{Enc}}$, q_v queries to $\mathcal{O}_{\mathsf{Eval}}$, q_d queries to $\mathcal{O}_{\mathsf{Dec}}$, and corrupts n parties, the advantage $\mathbf{Adv}^{\text{IND-CPA}^{\mathrm{pD}}}_{\mathsf{MGHE}}(\mathcal{A}) \leq \varepsilon$ where

$$\mathbf{Adv}^{\text{IND-CPA}^{\mathrm{pD}}}_{\mathsf{MKHE}}(\mathcal{A}) := \left|\Pr\left[\mathrm{G}^{\text{IND-CPA}^{\mathrm{pD}}\text{-0}}_{\mathsf{MKHE}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}^{\text{IND-CPA}^{\mathrm{pD}}\text{-1}}_{\mathsf{MKHE}}(\mathcal{A}) \Rightarrow 1\right]\right|.$$

3.2.1.5 Multi-Group Security

We view multi-group HE as a hybrid approach that combines the features of threshold and multi-key HE. Thus we define IND-CPA$^{\mathrm{pD}}$ security for MGHE by combining the IND-CPA$^{\mathrm{pD}}$ games for threshold and multi-key HE illustrated in Figure 3.7 and Figure 3.8.

We assume there are n parties in the system. We first allow the adversary to *non-adaptively* corrupt parties by selecting an index set $I_{\mathcal{A}} \subset [n]$ and choose random coins $\{r_i\}_{i \in I_{\mathcal{A}}}$ to generate keys for corrupted parties. Given the public keys $\{\mathsf{pk}_i\}_{i\in[n]}$ of each party and the secret keys $\{\mathsf{sk}_i\}_{i\in I_{\mathcal{A}}}$ of corrupted parties, we let $\mathcal{A}$ define k group by specifying index sets $I_j \subseteq [n]$ for each $j \in [k]$.

For encryption queries, we verify if a group public key jpk_j corresponds to a group j that has been completely corrupted by $\mathcal{A}$ by checking whether $I_j \subseteq I_{\mathcal{A}}$. If this condition holds, we return $\perp$, indicating an invalid query, since $\mathcal{A}$ possesses all the secret keys needed for decryption, which would result in a trivial win.

For evaluation queries, we similarly check if $\hat{I} \subseteq I_{\mathcal{A}}$ where $\hat{I}$ is the set of indices of parties whose messages are involved in the evaluation. If so, $\mathcal{A}$ can trivially decrypt the evaluated ciphertext when $\mathcal{T}[l].m_0 \neq \mathcal{T}[l].m_1$ without querying oracle $\mathcal{O}_{\mathsf{Dec}}$ to win the game. [6]

In Figure 3.9, we illustrate the IND-CPA$^{\mathrm{pD}}$ game for an MGHE scheme, with the adversary's advantage formally defined in Definition 19.

Definition 19 (IND-CPA$^{\mathrm{pD}}$ for MGHE)**.** A multi-group homomorphic encryption scheme MGHE is said to satisfy $(q_e, q_v, q_d, t, n, \varepsilon)$*-indistinguishability under chosen plaintext attack with decryption query* (IND-CPA$^{\mathrm{pD}}$), if for any adversaries $\mathcal{A}$ that runs in time t, makes q_e queries to $\mathcal{O}_{\mathsf{Enc}}$, q_v queries to $\mathcal{O}_{\mathsf{Eval}}$, q_d queries to $\mathcal{O}_{\mathsf{Dec}}$, and corrupts n parties, the advantage $\mathbf{Adv}^{\text{IND-CPA}^{\mathrm{pD}}}_{\mathsf{MGHE}}(\mathcal{A}) \leq \varepsilon$ where

$$\mathbf{Adv}^{\text{IND-CPA}^{\mathrm{pD}}}_{\mathsf{MGHE}}(\mathcal{A}) := \left|\Pr\left[\mathrm{G}^{\text{IND-CPA}^{\mathrm{pD}}\text{-0}}_{\mathsf{MGHE}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}^{\text{IND-CPA}^{\mathrm{pD}}\text{-1}}_{\mathsf{MGHE}}(\mathcal{A}) \Rightarrow 1\right]\right|.$$

It can be observed that for a *single-party* homomorphic encryption, IND-CPA$^{\mathrm{pD}}$ is equivalent to IND-CPA$^{\mathrm{D}}$ since it is assumed that an adversary cannot corrupt the single party to get the single secret key.

[6] We assume that $\mathcal{A}$ does not do any "external" evaluation except for querying the oracle $\mathcal{O}_{\mathsf{Eval}}$ so that it can exploit a corrupted group to make the distinguishing.

Game: $\mathsf{G}^{\text{IND-CPA}^{\text{pD}}\text{-}b}_{\mathsf{MGHE}}$

Procedure INIT

```
$\mathsf{pp} \leftarrow \mathsf{MGHE}.\mathtt{PGen}(1^\lambda, 1^\kappa)$
$(\{\gamma_i\}_{i \in I_\mathcal{A}}), I_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{pp})$ // $I_\mathcal{A} \subset [n]$
for $i \in [n] \setminus I_\mathcal{A}$ do
    $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MGHE}.\mathtt{KGen}(\mathsf{pp})$
for $i \in I_\mathcal{A}$ do
    $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MGHE}.\mathtt{KGen}(\mathsf{pp}; \gamma_i)$
$\{I_1, \ldots, I_k\} \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{i \in [n]}, \{\mathsf{sk}_i\}_{i \in I_\mathcal{A}})$
for $j \in [k]$ do
    $\mathsf{jpk}_j \leftarrow \mathsf{MGHE}.\mathtt{KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{i \in I_j})$
$\mathcal{T} := [\,]$
// Elements in $\mathcal{T}$ have structure $(I, \mathsf{pk}, m_0, m_1, c)$.
```

Oracle $\mathcal{O}_{\mathsf{Eval}}(f, L; \sigma)$

```
$\hat{I} := \cup_{l \in L} \mathcal{T}[l].I$
if $|\hat{I} \cap I_\mathcal{A}| \geq |\hat{I}|/t$ then
    return $\perp$
$\hat{c} \leftarrow \mathsf{MGHE}.\mathtt{Eval}(f, (\mathcal{T}[l].\mathsf{pk}, \mathcal{T}[l].c)_{l \in L}; \sigma)$
$\hat{m}_0 := f((\mathcal{T}[l].m_0)_{l \in L})$
$\hat{m}_1 := f((\mathcal{T}[l].m_1)_{l \in L})$
$\hat{\mathsf{jpk}} \leftarrow \mathsf{MGHE}.\mathtt{KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{i \in \hat{I}})$
$\mathcal{T} := \mathcal{T} \bowtie (\hat{I}, \hat{\mathsf{jpk}}, \hat{m}_0, \hat{m}_1, \hat{c})$
return $\hat{c}$
```

Oracle $\mathcal{O}_{\mathsf{Enc}}(\mathsf{jpk}_j, m_0, m_1)$

```
if $|I_j \cap I_\mathcal{A}| \geq |I_j|/t$ then
    return $\perp$
$c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m_b)$
$\mathcal{T} := \mathcal{T} \bowtie (I_j, \mathsf{jpk}_j, m_0, m_1, c)$
return $c$
```

Oracle $\mathcal{O}_{\mathsf{Enc}^*}(\mathsf{jpk}_j, m; \omega)$

```
$c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m; \omega)$
$\mathcal{T} := \mathcal{T} \bowtie (I_j, \mathsf{jpk}_j, m, m, c)$
return $c$
```

Oracle $\mathcal{O}_{\mathsf{Dec}}(l)$

```
if $\mathcal{T}[l].m_0 = \mathcal{T}[l].m_1$ then
    for $i \in \mathcal{T}[l].I$ do
        $d_i \leftarrow \mathsf{MGHE}.\mathtt{PDec}(\mathsf{sk}_i\ \mathcal{T}[l].c)$
    return $\{d_i\}_{i \in \mathcal{T}[l].I}$
```

Procedure FIN

```
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\{\mathsf{Enc}, \mathsf{Enc}^*, \mathsf{Eval}, \mathsf{Dec}\}}}(\{\mathsf{pk}_i\}_{i \in [n]}, \{\mathsf{jpk}_j\}_{j \in [k]}, \{\mathsf{sk}_i\}_{i \in I_\mathcal{A}})$
return $b'$
```

Figure 3.9: IND-CPA$^{\text{pD}}$ game for a multi-group homomorphic encryption (MGHE) scheme.

Remark 5. In Definitions 7 and 8, we define the oracle $\mathcal{O}_{\mathsf{Enc}^*}$ to accept the public key of a corrupted party (or group). This reflects that, in the multi-party setting, the adversary has the freedom to choose public keys, including those of corrupted parties. The oracle $\mathcal{O}_{\mathsf{Enc}^*}$ models the behavior of a corrupted party in the system, while the challenge oracle $\mathcal{O}_{\mathsf{Enc}}$ represents the behavior of an honest party. Together, these oracles enable the security game to model the security where the inputs of honest parties remain confidential.

3.2.2 Circuit Privacy

For server-side confidentiality, we consider *circuit privacy*. We remark that a notion proposed by Knabenhans [KVMGH23] considers all the secret inputs from the server. In our definition, we just use the term "circuit" to include those secret inputs (and public inputs) from server in addition to the operation gates. Indeed, that would simply depend on how a circuit is defined as we can essentially take those secret inputs as a part of the circuit.

3.2.2.1 Simulation-based versus Game-based Circuit Privacy

Prior to Gentry's thesis [Gen09a], *circuit privacy* has been discussed by Ishai and Paskin [IP07] from a simulation-based perspective, in which a simulator is expected to generate a ciphertext, given an evaluation on plaintext, that is *statistically* indistinguishable from an actual ciphertext evaluation. This definition was also adopted by Bourse et al. [BdPMW16], where the simulator additionally receives the ciphertexts to be evaluated.

In Gentry's thesis [Gen09a], however, a simulator is not explicitly used. Instead, circuit privacy is defined as the statistical indistinguishability between the encryption of an evaluated message and the evaluation of the corresponding ciphertexts, i.e., $\texttt{Enc}(\mathsf{pk}, f(m_1, \ldots, m_\ell)) \approx_s \texttt{Eval}(\mathsf{pk}, f, c_1, \ldots, c_\ell)$, where c_i is the encryption of m_i for $i \in [\ell]$.

Definition 20 (Circuit Privacy (SIM-CIRC) [IP07])**.** A homomorphic encryption scheme HE satisfies *circuit privacy* if for there exists a PPT simulator $\texttt{Sim}_{circ}$ such that for any $c_1, \ldots, c_\ell$ ciphertexts, it has

$$\texttt{Sim}_{circ}(\mathsf{pk}, f(m_1, \ldots, m_n)) \overset{s}{\approx} \mathsf{HE}.\texttt{Eval}(f, \mathsf{pk}, c_1, \ldots, c_\ell)$$

where each $c_i \leftarrow \mathsf{HE}.\texttt{Enc}(\mathsf{pk}, m_i)$ for $i \in [\ell]$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE}.\texttt{KGen}(1^\lambda)$.

Kluczniak and Santato [KS23] argued that this definition is specifically made for schemes with *exact* evaluation and may not be well-suited for schemes employing *approximate* evaluation, such as CKKS [CKKS17]. The key limitation stems from the fact that the error magnitude can leak information about the circuit's size or topology. Specifically, if $\texttt{Sim}_{circ}$ simulates using $\hat{m} + \delta$ for an evaluated ciphertext $\hat{m}$ where δ is the error incurred during approximate evaluation, observe that in Definition 10, the correctness of an evaluated ciphertext depends on the accumulated error from its input ciphertexts. Consequently, the value of δ inherently reveals the potential number of inputs in the circuit if we disclose it for simulation.

Furthermore, the use of $\hat{m}$ for simulation overlooks the fact that the evaluation might be approximate. Specifically, let $\hat{m} = f(m_1, \ldots, m_\ell)$ and $\hat{c} = \mathsf{HE}.\texttt{Eval}(\mathsf{pk}, f, c_1, \ldots, c_\ell)$. Here, $\hat{c}$ decrypts to a message $\hat{m}' \neq \hat{m}$. Thus the view from $\texttt{Sim}_{circ}(\mathsf{pk}, \hat{m})$ is different from $\texttt{Sim}_{circ}(\mathsf{pk}, \hat{m}')$, which makes it a not good definition for circuit privacy.

An alternative approach [LM21, KS23] adopts the classical indistinguishability game through *left-or-right* (LoR) evaluation. Li and Micciancio [LM21] briefly suggested an extension of their IND-CPA$^{\mathrm{D}}$ notion by modifying the evaluation oracle to accept two circuits, f_0 and f_1, along with two sets of indices, L_0 and L_1, specifying ciphertexts output by $\mathcal{O}_{\mathsf{Enc}}$ or $\mathcal{O}_{\mathsf{Eval}}$ to be evaluated. Additionally, they note that a weaker notion can be achieved by using a single index set, L, instead of two sets L_0 and L_1. This reveals the circuit's topology while hiding the values at each gate. Kluczniak and Santato [KS23] also proposed a similar but relaxed security game where the adversary $\mathcal{A}$ *non-adaptively* provides ℓ messages, $m_1, \ldots, m_\ell$, and two circuits, f_0 and f_1, for evaluation.

In Figure 3.10, we formalize the circuit indistinguishability by combining the ideas from both works [LM21, KS23] by allowing the adversary to *adaptively* query oracles for a stronger security. Specifically, we give the adversary access to an oracle $\mathcal{O}_{\mathsf{Enc}}$ that simply does encryption and an oracle $\mathcal{O}_{\mathsf{Eval}}$ that does the indistinguishability challenge.

Definition 21 (Circuit Indistinguishability (IND-CIRC))**.** A homomorphic encryption scheme HE is said to satisfy $(q_e, q_v, q_d, t, \varepsilon)$*-circuit indistinguishability* (IND-CIRC), if for any adversaries $\mathcal{A}$ running in time t, making q_e queries to $\mathcal{O}_{\mathsf{Enc}}$, q_v queries to $\mathcal{O}_{\mathsf{Eval}}$, and q_d queries to

Game: $\mathrm{G}_{\mathsf{HE}}^{\text{IND-CIRC-}b}$

```
Procedure INIT
1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{HE}.\mathtt{KGen}(1^\lambda)$
2 : $\mathcal{T} := []$
3 : // Elements in $\mathcal{T}$ have structure $(m, c)$.

Oracle $\mathcal{O}_{\mathsf{Eval}}(f_0, f_1, L_0, L_1)$
1 : $\hat{m}_0 := f_0((\mathcal{T}[l].m)_{l \in L_0})$
2 : $\hat{m}_1 := f_1((\mathcal{T}[l].m)_{l \in L_1})$
3 : if $\hat{m}_0 \neq \hat{m}_1$ then
4 :     return $\perp$
5 : $\hat{c} \leftarrow \mathsf{HE}.\mathtt{Eval}(f_b, \mathsf{pk}, \{\mathcal{T}[l].c\}_{l \in L_b})$
6 : $\mathcal{T} := \mathcal{T} \bowtie (\hat{m}_0, \hat{c})$
7 : return $\hat{c}$

Oracle $\mathcal{O}_{\mathsf{Enc}}(m; \omega)$
1 : $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{pk}, m; \omega)$
2 : $\mathcal{T} := \mathcal{T} \bowtie (m, c)$
3 : return $c$

Oracle $\mathcal{O}_{\mathsf{Dec}}(l)$
1 : $m \leftarrow \mathsf{HE}.\mathtt{Dec}(\mathsf{sk}, \mathcal{T}[l].c)$
2 : return $m$

Procedure FIN
1 : $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\{\mathsf{Enc},\mathsf{Eval},\mathsf{Dec}\}}}(\mathsf{pk})$
2 : return $b'$
```

Figure 3.10: IND-CIRC game for a homomorphic encryption scheme.

$\mathcal{O}_{\mathsf{Dec}}$, the advantage $\mathbf{Adv}_{\mathsf{HE}}^{\text{IND-CIRC}}(\mathcal{A}) \leq \varepsilon$ where

$$\mathbf{Adv}_{\mathsf{HE}}^{\text{IND-CIRC}}(\mathcal{A}) := \left|\Pr\left[\mathrm{G}_{\mathsf{HE}}^{\text{IND-CIRC-0}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}_{\mathsf{HE}}^{\text{IND-CIRC-1}}(\mathcal{A}) \Rightarrow 1\right]\right|.$$

For the rest of this thesis, we focus on the IND-CIRC notion. In Lemma 4, we show that SIM-CIRC implies IND-CIRC for any homomorphic encryption with perfect correctness.

Lemma 4. (SIM-CIRC $\mapsto$ IND-CIRC) *For any homomorphic encryption scheme* HE *with exact evaluation, if* HE *satisfies* SIM-CIRC*, then it satisfies* IND-CIRC.

Proof. For any IND-CIRC adversary $\mathcal{A}$, we show that there exist two SIM-CIRC distinguishers $\mathcal{D}_0$ and $\mathcal{D}_1$. Let $(\mathsf{pk}, \cdot) \leftarrow \mathsf{HE}.\mathtt{KGen}(1^\lambda)$ and let $\mathcal{D}_b$ maintain a transcript $\mathcal{T}_b$ for $b \in \{0, 1\}$ starting from empty. We handle the queries from $\mathcal{A}$ as follows.

- **On the query** m **to** $\mathcal{O}_{\mathsf{Enc}}$ **from** $\mathcal{A}$: For each $b \in \{0, 1\}$, we let $\mathcal{D}_b$ runs $c \leftarrow \mathsf{HE}.\mathtt{Enc}(\mathsf{pk}, m)$, append $\mathcal{T}_b$ with (m, c) and return c to $\mathcal{A}$.
- **On the query** (f_0, f_1, L_0, L_1) **to** $\mathcal{O}_{\mathsf{Eval}}$ **from** $\mathcal{A}$: For each $b \in \{0, 1\}$, we let $\mathcal{D}_b$ first compute $\hat{m}_b = f_b((m_l)_{l \in L_b})$. If $\hat{m}_0 \neq \hat{m}_1$, we let $\mathcal{D}_b$ return $\perp$ to $\mathcal{A}$. Otherwise, we let $\mathcal{D}_b$ interact with the system. In the real world, $\mathcal{D}_b$ gets $\hat{c} \leftarrow \mathsf{HE}.\mathtt{Eval}(f_b, \mathsf{pk}, (\mathcal{T}_b[l].m)_{l \in L_b})$ for $b \in \{0, 1\}$. In the ideal world, $\mathcal{D}_b$ gets $\hat{c}$ as the output from the simulator $\mathtt{Sim}(\mathsf{pk}, \hat{m}_b)$. Then $\mathcal{D}_b$ append $\mathcal{T}_b$ with $(\hat{m}, \hat{c})$ and return $\hat{c}$ to $\mathcal{A}$.
- **On the query** j **to** $\mathcal{O}_{\mathsf{Dec}}$ **from** $\mathcal{A}$: We simply let $\mathcal{D}_0$ returns $\mathcal{T}_0[l].m$ to $\mathcal{A}$.

Eventually, $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$. If $b' = 0$, we then let $\mathcal{D}_0$ return 1 and let $\mathcal{D}_1$ return 0; otherwise, we let $\mathcal{D}_0$ return 0 and let $\mathcal{D}_1$ return 1.

Observe that in the real world, $\mathcal{D}_b$ has the same view as $\mathrm{G}_{\mathsf{HE}}^{\text{IND-CIRC-}b}(\mathcal{A})$ for $b \in \{0, 1\}$. Also, in the ideal world, at each query to $\mathcal{O}_{\mathsf{Eval}}$, the message $\hat{m}_b$ passed to the simulator $\mathtt{Sim}$ for $b \in \{0, 1\}$ is the same. Thus in the ideal world, the distinguisher $\mathcal{D}_0$ and $\mathcal{D}_1$ has exactly the same view. Also, since we assume HE has perfect correctness, $T_b[l].m$ returned by $\mathcal{D}_b$ to $\mathcal{A}$ is the same as HE.Dec(sk, $T[l].c$). Thus we have that

$$\begin{aligned}\mathbf{Adv}^{\text{IND-CIRC}}_{\mathsf{HE}}(\mathcal{A}) &= \left|\Pr\left[\mathrm{G}^{\text{IND-CIRC-0}}_{\mathsf{HE}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}^{\text{IND-CIRC-1}}_{\mathsf{HE}}(\mathcal{A}) \Rightarrow 1\right]\right| \\ &\leq \Delta_{\mathcal{D}_0}(\mathtt{Sim}(\mathsf{pk}, \hat{m}_0), \hat{c}_0) - \Delta_{\mathcal{D}_1}(\mathtt{Sim}(\mathsf{pk}, \hat{m}_1), \hat{c}_1)\end{aligned}$$

which concludes the proof. □

Remark 6. In Figure 3.10, the check $\hat{m}_0 \neq \hat{m}_1$ is positioned in the oracle $\mathcal{O}_{\mathsf{Eval}}$ instead of $\mathcal{O}_{\mathsf{Dec}}$. This modification allows us to create the same view for distinguishers $\mathcal{D}_0$ and $\mathcal{D}_1$ when they interact with the ideal world, and thus enables us to build an implication from SIM-CIRC to IND-CIRC.

Game: $\mathrm{G}^{\text{IND-CIRC-}b}_{\mathsf{MGHE}}$

Procedure INIT

```
$\mathsf{pp} \leftarrow \mathsf{MGHE.PGen}(1^\lambda, 1^\kappa)$
$(\{\gamma_i\}_{i \in I_\mathcal{A}}), I_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{pp})$ // $I_\mathcal{A} \subseteq [n]$
for $i \in [n] \setminus I_\mathcal{A}$ do
    $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MGHE.KGen}(\mathsf{pp})$
for $i \in I_\mathcal{A}$ do
    $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MGHE.KGen}(\mathsf{pp}; \gamma_i)$
$\{I_1, \ldots, I_k\} \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{i \in [n]}, \{\mathsf{sk}_i\}_{i \in I_\mathcal{A}})$
for $j \in [k]$ do
    $\mathsf{jpk}_j \leftarrow \mathsf{MGHE.KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{i \in I_j})$
$T := [\,]$
// Elements in $T$ have structure $(I, \mathsf{pk}, m_0, m_1, c)$.
```

Oracle $\mathcal{O}_{\mathsf{Dec}}(l)$

```
for $i \in T[l].I$ do
    $d_i \leftarrow \mathsf{MGHE.PDec}(\mathsf{sk}_i\ T[l].c)$
return $\{d_i\}_{i \in T[l].I}$
```

Oracle $\mathcal{O}_{\mathsf{Enc}}(\mathsf{jpk}_j, m; \omega)$

```
$c \leftarrow \mathsf{MGHE.Enc}(\mathsf{jpk}_j, m; \omega)$
$T := T \bowtie (I_j, \mathsf{jpk}_j, m, c)$
return $c$
```

Oracle $\mathcal{O}_{\mathsf{Eval}}(f_0, f_1, L_0, L_1)$

```
$\hat{I} := \cup_{l \in L_b} T[l].I$
$\hat{m}_0 := f_0((T[l].m)_{l \in L_0})$
$\hat{m}_1 := f_1((T[l].m)_{l \in L_1})$
if $\hat{m}_0 \neq \hat{m}_1$ then
    return $\perp$
$\hat{c} \leftarrow \mathsf{MGHE.Eval}(f_b, (T[l].\mathsf{pk}, T[l].c)_{l \in L_b})$
$\hat{\mathsf{jpk}} \leftarrow \mathsf{MGHE.KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{i \in \hat{I}})$
$T := T \bowtie (\hat{I}, \hat{m}_0, \hat{c})$
return $\hat{c}$
```

Procedure FIN

```
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\{\mathsf{Enc,Eval,Dec}\}}}(\{\mathsf{pk}_i\}_{i \in [n]}, \{\mathsf{jpk}_j\}_{j \in [k]}, \{\mathsf{sk}_i\}_{i \in I_\mathcal{A}})$
return $b'$
```

Figure 3.11: IND-CIRC game for a multi-group homomorphic encryption scheme.

3.2.2.2 Extension to Multi-Group Setting

We extend this notion to a multi-group setting, considering a *semi-malicious* adversary who can choose the random coins used for encryption and key generation from arbitrary distribution. This adversary model is weaker than the one used in *malicious* circuit privacy [OPP14, DD22], where adversaries can generate ciphertexts and public (evaluation) keys in a fully malicious manner. However, for this part, it suffices to consider a semi-malicious adversary, as we later use a zkSNARK to ensure the well-formedness of ciphertexts and assume that the keys are honestly generated.

For completeness, we first extend the simulation-based definition for circuit privacy to the multi-group setting in Definition 22.

Definition 22 (Circuit Privacy (SIM-CIRC))**.** A multi-group homomorphic encryption scheme MGHE satisfies *circuit privacy* if for there exists a PPT simulator $\mathtt{Sim}$ such that for any $c_1, \ldots, c_\ell$ ciphertexts, it has

$$\mathtt{Sim}_{circ}(\{\mathsf{jpk}_1, \ldots, \mathsf{jpk}_\ell\}, f(m_1, \ldots, m_\ell)) \overset{s}{\approx} \mathsf{MGHE}.\mathtt{Eval}(f, (\mathsf{jpk}_j, c_j)_{j\in[\ell]})$$

where for $j \in [\ell]$, $c_j \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m_i; \omega_i)$, $\mathsf{jpk}_j \leftarrow \mathsf{MGHE}.\mathtt{KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{i\in I_j})$ and $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MGHE}.\mathtt{KGen}(\mathsf{pp}; \gamma_i)$ with $\mathsf{pp} \leftarrow \mathsf{MGHE}.\mathtt{PGen}(1^\lambda, 1^\kappa)$ for some security parameter λ and circuit depth κ, where ω_i and γ_i are random coins that may be selected by an adversary from an arbitrary distribution.

Now we consider the variant of IND-CIRC for MGHE. The adversary $\mathcal{A}$ is similarly allowed to corrupt selected clients in these groups by specifying corrupted client indices $I_\mathcal{A} \subseteq [n]$. Note that we no longer require $I_\mathcal{A}$ to be a *strict* subset of $[n]$, which means that $\mathcal{A}$ can choose to corrupt all n clients in the system. The adversary is similarly provided with an oracle $\mathcal{O}_{\mathsf{Enc}}$ for encryption and a decryption oracle $\mathcal{O}_{\mathsf{Dec}}$ for partial-decryptions. The challenge oracle is then changed to a left-or-right evaluation oracle $\mathcal{O}_{\mathsf{Eval}}$ that takes inputs (f_0, f_1, L_0, L_1) such that $f_0((\mathcal{T}[l].m)_{l\in L_0}) = f_1((\mathcal{T}[l].m)_{l\in L_1})$ for a transcript $\mathcal{T}$ that records the $\mathcal{A}$'s queries to the oracles $\mathcal{O}_{\mathsf{Enc}}$ and $\mathcal{O}_{\mathsf{Eval}}$.

Similarly, we allow the adversary to choose any public key (including those of corrupted groups) and the random coins when querying to $\mathcal{O}_{\mathsf{Enc}}$ to model the behavior of a *semi-malicious* adversary. To also account for the behavior of honest parties, we simply let the adversary query with honestly sampled random coins. We emphasize that the adversary is *not* allow to select the random coins at its queries to $\mathcal{O}_{\mathsf{Eval}}$. This restriction arises because we assume that the server remains uncorrupted when formalizing circuit privacy and, as such, the server selects random coins honestly during the evaluation and these random coins are not disclosed to the adversary.

Also note that we no longer impose the restriction preventing $\mathcal{A}$ from corrupting an entire group at the challenge oracle. As each tuple (f_0, f_1, L_0, L_1) passed to $\mathcal{O}_{\mathsf{Eval}}$ must yield that $\hat{m}_0 = \hat{m}_1$, it is equivalent for $\mathcal{A}$ to decrypt itself or query $\mathcal{O}_{\mathsf{Dec}}$. This would imply the security that the circuit should remain confidential even under the corruption of the entire group.

Definition 23. A multi-group homomorphic encryption scheme MGHE is said to satisfy (q_e, q_v, q_d, t, n, ε)*-circuit indistinguishability* (IND-CIRC), if for any adversaries $\mathcal{A}$ running in time t, making q_e encryption queries, q_v evaluation queries, q_d decryption queries, and corrupting n parties, the advantage $\mathbf{Adv}^{\text{IND-CIRC}}_{\mathsf{MGHE}}(\mathcal{A}) \leq \varepsilon$ where

$$\mathbf{Adv}^{\text{IND-CIRC}}_{\mathsf{MGHE}}(\mathcal{A}) := \left|\Pr\left[\mathrm{G}^{\text{IND-CIRC-0}}_{\mathsf{MGHE}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}^{\text{IND-CIRC-1}}_{\mathsf{MGHE}}(\mathcal{A}) \Rightarrow 1\right]\right|.$$

Remark 7. As defined in the oracle $\mathcal{O}_{\mathsf{Enc}}$ in Figure 3.11, we consider a security model where all parties providing inputs to the circuit are controlled by the adversary. This represents the worst-case scenario but remains a realistic possibility. Nonetheless, we also "preserve" honest parties in the model to better reflect real-world scenarios, even if their presence may have limited practical impact. To "mimic" the behavior of an honest party, the adversary can simply encrypt inputs using honest random coins and "pretend" to be unaware of the underlying inputs.

3.2.3 Threshold-Decryption Security

In this thesis, we focus on schemes that support partial decryption, which necessitates two additional security properties: partial decryption must not leak any information about the secret keys and the original message must be recoverable from enough partial decryptions.

3.2.3.1 Partial-Decryption Simulatability

To formalize the security property where partial decryption must not leak any information about the secret keys, in Definition 24, we introduce the notion of *partial-decryption simulatability* (SIM-PDEC). Note that although we introduce the notion in the context of multi-group HE, this definition is equally applicable to threshold and multi-key HE schemes.

The intuition behind this property is as follows: in a multi-party setting where a subset of parties is corrupted (denoted by $I_{\mathcal{A}}$), given the plaintext m, its ciphertext c, and the secret keys of the corrupted parties, there exists a simulator $\mathtt{Sim}_{th}$ capable of producing a partial decryption d for an honest party without knowledge of that party's secret key, such that this simulated d is *statistically* indistinguishable from the output of $\mathsf{MGHE}.\mathtt{PDec}(\cdot)$.

Definition 24 (Partial-Decryption Simulatability (SIM-PDEC))**.** Let I be an index set representing all the clients in a group and let $I_{\mathcal{A}} \subset I$ denote the index set of corrupted parties. A multi-group homomorphic encryption scheme MGHE satisfies *partial-decryption simulatability* if there exists a PPT simulator $\mathtt{Sim}_{th}$ such that

$$\forall i \in I \setminus I_{\mathcal{A}},\ \mathtt{Sim}_{th}(c, m, \{\mathsf{sk}_j\}_{j \in I_{\mathcal{A}}}) \overset{s}{\approx} \mathsf{MGHE}.\mathtt{PDec}(\mathsf{sk}_i, c)$$

where $m \leftarrow \mathsf{MGHE}.\mathsf{DistDec}(\{\mathsf{sk}_i\}_{i \in I}, c)$.

Remark 8 (Simulatability for schemes on approximate numbers)**.** We observe that the security of schemes operating on approximate numbers, such as [CKKS17], also aligns with this definition. In this case, we define m as the exact value of the message that was encrypted or evaluated. Intuitively, the partial decryption of a ciphertext generated by an (honest) party h in the real world takes the form

$$d_{\text{real}} = \mathtt{Ecd}(m) + \delta_{\mathsf{ct}} + \sigma + \eta + \psi(c, \{d_i\}_{i \neq h}),$$

where δ_{ct} represents the inherent error in the ciphertext, arising from encryption and accumulating through evaluation without additional sanitization or smudging. The term σ denotes noise introduced by the server, such as for circuit privacy via ciphertext sanitization, and η represents extra noise added by the honest party during partial decryption, for instance, through noise-smudging. The function ψ depends on the ciphertext c and the partial decryptions from other (potentially corrupted) parties. We note that in both exact and approximate schemes, the ciphertext c contains δ_{ct}. Thus the formalization holds for both types of the schemes.

3.2.3.2 Decryption Consistency

Decryption consistency (also known as *robustness* in some works) is a property specifically for threshold decryption that has been discussed in many prior works about threshold (HE) cryptosystems [BBH06, BGG$^+$18, BS23, BPR24]. Intuitively, an adversary may inject false partial decryptions to manipulate the reconstructed message. Decryption consistency ensures that if a player obtains sufficiently many honestly generated partial decryptions, the original message is correctly reconstructed, regardless of the adversary's influence. In Figure 3.12, we adapt the notion of decryption consistency for threshold HE introduced by Boudgoust and Scholl [BS23] and formalize it in the context of multi-group HE. Their definition considers decryption consistency only for ciphertexts that is the encryption of a message. However, in most applications, the focus should be more on evaluated ciphertexts, as clients primarily do a threshold decryption on these ciphertexts.

Similarly as in the security notions we introduced earlier, we allow $\mathcal{A}$ to define k groups by specifying index sets $I_1, \ldots, I_k$ and to corrupt a subset of clients, represented by the index set

Game: $\mathrm{G}^{\mathrm{DC}}_{\mathsf{MGHE}}$

Procedure INIT
```
1 : $\mathsf{pp} \leftarrow \mathsf{MGHE.PGen}(1^\lambda, 1^\kappa)$
2 : $(\{\gamma_i\}_{i \in I_\mathcal{A}}), I_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{pp})$ $/\!/ I_\mathcal{A} \subset [n]$
3 : for $i \in [n] \setminus I_\mathcal{A}$ do
4 :     $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MGHE.KGen}(\mathsf{pp})$
5 : for $i \in I_\mathcal{A}$ do
6 :     $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MGHE.KGen}(\mathsf{pp}; \gamma_i)$
7 : $\{I_1, \dots, I_k\} \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{i \in [n]}, \{\mathsf{sk}_i\}_{i \in I_\mathcal{A}})$
8 : for $j \in [k]$ do
9 :     $\mathsf{jpk}_j \leftarrow \mathsf{MGHE.KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{i \in I_j})$
10 : $\mathcal{T} := [\,]$
11 : $/\!/$Elements in $\mathcal{T}$ have structure $(I, \mathsf{pk}, m_0, m_1, c)$.
```

Procedure FIN
```
1 : $(l, \{d_i\}_{i \in I'}) \leftarrow \mathcal{A}^{\mathcal{O}_{\{\mathsf{Enc},\mathsf{Eval},\mathsf{Dec}\}}}(\{\mathsf{sk}_i\}_{i \in I_\mathcal{A}})$
2 : $/\!/ |\mathcal{T}[l].\mathbf{d} \cap \{d_i\}_{i \in I'}| \geq \frac{|\mathcal{T}[l].I|}{t}, I' \subseteq \mathcal{T}[l].I$
3 : $m \leftarrow \mathsf{MGHE.Combine}(\mathcal{T}[l].c, \{d_i\}_{i \in I'})$
4 : return $(m \neq \mathcal{T}[l].m)$
5 : return $||\mathcal{T}[l].m - m|| > \mathsf{Estimate}(\cdot)$
```

Oracle $\mathcal{O}_{\mathsf{Enc}}(\mathsf{jpk}_j, m; \omega)$
```
1 : $c \leftarrow \mathsf{MGHE.Enc}(\mathsf{jpk}_j, m; \omega)$
2 : $\mathcal{T} := \mathcal{T} \bowtie (I_j, \mathsf{jpk}_j, m, c, \perp)$
3 : return $c$
```

Oracle $\mathcal{O}_{\mathsf{Eval}}(f, L; \sigma)$
```
1 : $\hat{I} := \cup_{l \in L} \mathcal{T}[l].I$
2 : $\hat{m} := f((\mathcal{T}[l].m)_{l \in L})$
3 : $\hat{c} \leftarrow \mathsf{MGHE.Eval}(f, (\mathcal{T}[l].\mathsf{jpk}, \mathcal{T}[l].c)_{l \in L}; \sigma)$
4 : $\hat{\mathsf{jpk}} \leftarrow \mathsf{MGHE.KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{i \in \hat{I}})$
5 : $\mathcal{T} := \mathcal{T} \bowtie (\hat{I}, \hat{m}, \hat{c}, \perp)$
6 : return $\hat{c}$
```

Oracle $\mathcal{O}_{\mathsf{Dec}}(l; \{\eta_i\}_{i \in I_\mathcal{A} \cap \mathcal{T}[l].I})$
```
1 : for $i \in \mathcal{T}[l].I \setminus I_\mathcal{A}$ do
2 :     $d_i \leftarrow \mathsf{MGHE.PDec}(\mathsf{sk}_i, \mathcal{T}[l].c)$
3 : for $i \in \mathcal{T}[l].I \cap I_\mathcal{A}$ do
4 :     $d_i \leftarrow \mathsf{MGHE.PDec}(\mathsf{sk}_i, \mathcal{T}[l].c; \eta_i)$
5 : $\mathcal{T}.\mathbf{d} := \{d_i\}_{i \in \mathcal{T}[l].I}$
6 : return $\{d_i\}_{i \in \mathcal{T}[l].I}$
```

Figure 3.12: Decryption Consistency (DC) game for a multi-group homomorphic encryption scheme. The dot-boxed part is exclusively for MGHE on exact numbers and the highlighted part is exclusively for MGHE on approximate numbers.

$I_\mathcal{A}$, during the initialization phase. We maintain a lookup table $\mathcal{T}$ with a structure similar to that in Figures 3.9 and 3.11 to track message-ciphertext pairs and their associated groups, and the partial decryptions that are obtained honestly for the ciphertext.

The adversary $\mathcal{A}$ has access to an encryption oracle $\mathcal{O}_{\mathsf{Enc}}$, which encrypts a message m using the joint public key jpk_j of group I_j and a random coin r chosen by $\mathcal{A}$. Additionally, $\mathcal{A}$ can query an evaluation oracle $\mathcal{O}_{\mathsf{Eval}}$ to evaluate a circuit over ciphertexts produced by either $\mathcal{O}_{\mathsf{Enc}}$ or $\mathcal{O}_{\mathsf{Eval}}$. All oracle queries are recorded in $\mathcal{T}$. Moreover, $\mathcal{A}$ is given access to a decryption oracle $\mathcal{O}_{\mathsf{Dec}}$, which honestly computes partial decryptions on behalf of clients (including those corrupted by $\mathcal{A}$) and records them in $\mathcal{T}$. We note that we allow the adversary to select the random coins η_i's, e.g. for noise-smudging, for the corrupted parties at its query to $\mathcal{O}_{\mathsf{Dec}}$ and we still take this as a legitimate partial decryption.

In the finalization phase, $\mathcal{A}$ must output a set of partial decryptions $\{d_i\}_{i \in I'}$ along with an index l corresponding to a message in $\mathcal{T}$ that it wishes to reconstruct. We require that at least $\frac{\mathcal{T}[l].I}{t}$ of the partial decryptions in $\{d_i\}_{i \in I'}$ are honestly obtained,[7] where $\mathcal{T}[l].I$ denotes the group for which the message is intended. The adversary $\mathcal{A}$ wins the game if the partial decryptions $\{d_i\}_{i \in I'}$ reconstruct to a message different from the one stored in $\mathcal{T}$. For approximate schemes, the adversary wins the game if the distance between the actual message and the reconstructed message is beyond the estimated error.

[7]An adversary may still instruct a corrupted client to output an honest partial decryption.

Definition 25 (Decryption Consistency)**.** A homomorphic encryption scheme HE is said to satisfy (t, ε)-*decryption consistency* (DC), if for any adversaries $\mathcal{A}$ that runs in time t, the advantage $\mathbf{Adv}^{\mathrm{DC}}_{\mathsf{MGHE}}(\mathcal{A}) \leq \varepsilon$ where

$$\mathbf{Adv}^{\mathrm{DC}}_{\mathsf{MGHE}}(\mathcal{A}) \coloneqq \Pr\left[\mathrm{G}^{\mathrm{DC}}_{\mathsf{MGHE}}(\mathcal{A}) \Rightarrow 1\right].$$

3.3 UC View for Confidentiality

In our system, we consider n clients, denoted by $\{\mathcal{P}_{\mathsf{cli}_i}\}_{i\in[n]}$. At a state, a subset of these clients $\{\mathcal{P}_{\mathsf{cli}_i}\}_{i\in I}$ with $I \subseteq [n]$ may organize into k groups to perform a specific computation task. Each group is represented as a "virtual entity" $\mathcal{P}_{\mathsf{grp}_j}$. For notation simplicity, we use the notation $\mathsf{grp}_j = \{\mathsf{cli}_i\}_{i\in I_j}$ to express the relationship between the group's party ID and its constituent clients, with $I_j \subseteq I$ for $j \in [k]$ denoting the corresponding index set. Notably, a single client can belong to multiple groups.

A server $\mathcal{P}_{\mathsf{srv}}$ processes the encrypted messages by evaluating them over a circuit. All the associated clients of an evaluated ciphertext forms a new group $\hat{\mathsf{grp}}$ that collaboratively decrypts the evaluated results.

3.3.1 Global Key Registry $\mathcal{G}_{\mathsf{KRK}}$

Inspired by the *key registration with knowledge* (KRK) setup introduced by Barak et al. [BCNP04], we define a global functionality $\mathcal{G}_{\mathsf{KRK}}$ for public parameter and key registration in multi-group setting, as illustrated in Figure 3.13.

We remark that $\mathcal{G}_{\mathsf{KRK}}$ is Π-regular for every protocol Π since it does not invoke subprotocols nor passes outputs to any ITM that did not query it. $\mathcal{G}_{\mathsf{KRK}}$ is also subroutine-respecting since it does not communicate with a party outside the session. Thus the precondition of Corollary 1 is satisfied.

Note for simplicity, we let the `JoinKey` action to be handled by the algorithm $\mathsf{MGHE}.\mathtt{KJoin}(\cdot)$ instead of running an MPC protocol. If an MPC functionality is needed, then it should be taken out from $\mathcal{G}_{\mathsf{KRK}}$ and considered separately.

Parameter Setup & Retrieval. We let the system begin when a client $\mathcal{P}_{\mathsf{cli}}$ sends the input $\langle\mathtt{RegParam} : [\mathsf{sid}, \lambda, d]\rangle$ to $\mathcal{G}_{\mathsf{KRK}}$, specifying a security parameter λ and a maximum circuit depth λ, to register a public parameter pp for a session identified by sid. If a public parameter is already initialized for the session, i.e., $\mathsf{Params}[\mathsf{sid}] \neq \perp$, then $\mathcal{G}_{\mathsf{KRK}}$ aborts on the input. Otherwise, it generates the public parameter as $\mathsf{pp} \leftarrow \mathsf{MGHE}.\mathtt{PGen}(1^\lambda, 1^\kappa)$, stores it in the lookup table Params under the session ID sid, and fixes it for the session's duration.

When any party $\mathcal{P}_{\mathsf{pid}}$ queries $\mathcal{G}_{\mathsf{KRK}}$ for the public parameter of session sid using the input $\langle\mathtt{RegParam} : [\mathsf{sid}]\rangle$, we let $\mathcal{G}_{\mathsf{KRK}}$ check the record in Params. If found, it outputs the message $\{\mathtt{"param"} : [\mathsf{sid}, \mathsf{pp}]\}$ to the inputing party; otherwise, it outputs $\perp$ to indicate non-existence.

Individual Key Generation. Once the public parameter is set, a client $\mathcal{P}_{\mathsf{cli}}$ or $\mathcal{S}$ in the name of a corrupted client can send the input $\langle\mathtt{RegKey} : [\mathsf{sid}, \mathsf{pp}]\rangle$ to $\mathcal{G}_{\mathsf{KRK}}$ to register an individual key pair. If the provided public parameter pp does not match the stored parameter for the session, i.e., $\mathsf{Params}[\mathsf{sid}] \neq \mathsf{pp}$, then $\mathcal{G}_{\mathsf{KRK}}$ aborts on this input. Then if the input is from an honest client $\mathcal{P}_{\mathsf{cli}}$, we run $\mathsf{MGHE}.\mathtt{KGen}(\mathsf{pp})$ assuming that a random coin is sampled honestly following a specific distribution and used as mentioned in Remark 1. Otherwise, we let $\mathcal{S}$ specify a (possibly malicious) random coin for key generation in the name of the corrupted client. We then generate a key pair as $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{MGHE}.\mathtt{KGen}(\mathsf{pp}; \eta)$ and registers it in the lookup table Keys under the index $(\mathsf{cli}, \mathsf{sid})$.

Functionality: $\mathcal{G}_{\mathsf{KRK}}$

Participants: n clients $\mathcal{P}_{\mathsf{cli}_1}, \mathcal{P}_{\mathsf{cli}_2}, \ldots, \mathcal{P}_{\mathsf{cli}_n}$. At a state, a subset of clients $\{\mathcal{P}_{\mathsf{cli}_i}\}_{i \in I}$ with $I \subseteq [n]$ form k groups $\mathcal{P}_{\mathsf{grp}_j}$ with $\mathsf{grp}_j = \{\mathsf{cli}_i\}_{i \in I_j}$ with $I_j \subseteq I$ for $j \in [k]$. A server $\mathcal{P}_{\mathsf{srv}}$, and an adversary $\mathcal{A}/\mathcal{S}$.

Initialization

```
$\mathsf{Params} := []$ //Record for public parameter
$\mathsf{Keys} := []$ //Record for key pairs
$\mathsf{Corrupt} := \varnothing$ //A set of corrupted parties
```

Corruption

```
In $\langle \texttt{Corrupt} : [\mathsf{pid}] \rangle \leftarrow \mathcal{A}/\mathcal{S}$
    $\mathsf{Corrupt} := \mathsf{Corrupt} \cup \{\mathsf{pid}\}$
```

Parameter Setup

```
In $\langle \texttt{RegParam} : [\mathsf{sid}, \lambda, \kappa] \rangle \leftarrow \mathcal{P}_{\mathsf{cli}}$
    if $\mathsf{Params}[\mathsf{sid}] \neq \perp$ then
        Abort
    else
        $\mathsf{pp} \leftarrow \mathsf{MGHE}.\texttt{PGen}(1^\lambda, 1^\kappa)$
        $\mathsf{Params}[\mathsf{sid}] := \mathsf{pp}$
```

```
In $\langle \texttt{RetParam} : [\mathsf{sid}] \rangle \leftarrow \mathcal{P}_{\mathsf{cli}}$
    if $\mathsf{Params}[\mathsf{sid}] = \perp$ then
        Out $\{\texttt{"param"} : [\mathsf{sid}, \perp]\} \rightarrow \mathcal{P}_{\mathsf{cli}}$
    else
        $\mathsf{pp} := \mathsf{Params}[\mathsf{sid}]$
        Out $\{\texttt{"param"} : [\mathsf{sid}, \mathsf{pp}]\} \rightarrow \mathcal{P}_{\mathsf{cli}}$
```

Key Generation

```
In $\left\langle \texttt{RegKey} : [\mathsf{cli}, \mathsf{sid}, \mathsf{pp}; \boxed{\gamma}] \right\rangle \leftarrow \{\mathcal{P}_{\mathsf{cli}}, \mathcal{A}/\mathcal{S}\}$
    if $\mathsf{Params}[\mathsf{sid}] \neq \mathsf{pp}$ then
        Abort
    if $\mathsf{cli} \notin \mathsf{Corrupt}$
      $\wedge \langle \texttt{RegKey} : [\cdot] \rangle \leftarrow \mathcal{A}/\mathcal{S}$ then
        Abort
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{MGHE}.\texttt{KGen}(\mathsf{pp}; \boxed{\gamma})$
    $\mathsf{Keys}[(\mathsf{cli}, \mathsf{sid})] := (\mathsf{pk}, \mathsf{sk}; \boxed{\gamma})$
```

```
In $\langle \texttt{JoinKey} : [\mathsf{grp}, \mathsf{sid}] \rangle \leftarrow \mathcal{P}_{\mathsf{grp}}$
    if $\exists \mathsf{cli}_i \in \mathsf{grp} : \mathsf{Params}[(\mathsf{cli}_i, \mathsf{sid})] = \perp$
        Abort
    for $\mathsf{cli}_i \in \mathsf{grp}$ do
        $(\mathsf{pk}_i, \mathsf{sk}_i) := \mathsf{Keys}[(\mathsf{cli}_i, \mathsf{sid})]$
    $\mathsf{jpk} \leftarrow \mathsf{MGHE}.\texttt{KJoin}((\mathsf{pk}_i, \mathsf{sk}_i)_{\mathsf{cli}_i \in \mathsf{grp}})$
    $\mathsf{Keys}[(\mathsf{grp}, \mathsf{sid})] := (\mathsf{jpk}, \perp)$
```

```
In $\langle \texttt{RetKey} : [\mathsf{pid}', \mathsf{sid}] \rangle \leftarrow \{\mathcal{P}_{\mathsf{pid}}, \mathcal{A}/\mathcal{S}\}$
    if $\mathsf{Keys}[(\mathsf{pid}', \mathsf{sid})] = \perp$ then
        Out $\{\texttt{"key"} : [\mathsf{pid}, \mathsf{sid}, \perp]\} \rightarrow \mathcal{P}_{\mathsf{pid}}$
    else
        $(\mathsf{pk}, \mathsf{sk}) := \mathsf{Keys}[(\mathsf{pid}', \mathsf{sid})]$
        if $\mathsf{pid}' = \mathsf{pid}$ then
            Out $\{\texttt{"key"} : [\mathsf{pid}, \mathsf{sid}, (\mathsf{pk}, \mathsf{sk})]\} \rightarrow \mathcal{P}_{\mathsf{pid}}$
        elseif $\mathsf{pid}' \in \mathsf{Corrupt}$ then
            Out $\{\texttt{"key"} : [\mathsf{pid}, \mathsf{sid}, (\mathsf{pk}, \mathsf{sk})]\} \rightarrow \mathcal{A}/\mathcal{S}$
        else
            Out $\left\{\texttt{"key"} : \left[\mathsf{pid}', \mathsf{sid}, \mathsf{pk}\right]\right\} \rightarrow \mathcal{P}_{\mathsf{pid}}$
```

Figure 3.13: A global key registry functionality $\mathcal{G}_{\mathsf{KRK}}$. The dot-boxed parts are exclusive inputs from the adversary $\mathcal{A}$ or $\mathcal{S}$.

Group Key Aggregation. A client group, represented as a "virtual party" $\mathcal{P}_{\mathsf{grp}}$, is formed once all individual clients in the group have registered their keys. When $\mathcal{P}_{\mathsf{grp}}$ sends the input $\langle \texttt{JoinKey} : [\mathsf{grp}, \mathsf{sid}] \rangle$ to $\mathcal{G}_{\mathsf{KRK}}$, we let $\mathcal{G}_{\mathsf{KRK}}$ first verify that all required individual public keys for the session sid are available. We note that if non-interactive key aggregation (like the one proposed by Kwak et al. [KLSW24] is possible, then we can have a client $\mathcal{P}_{\mathsf{cli}}$ or the server $\mathcal{P}_{\mathsf{srv}}$ send this input instead of having the virtual group entity. If any key is missing, then $\mathcal{G}_{\mathsf{KRK}}$ aborts. Otherwise, it aggregates the individual key pairs using $\mathsf{MGHE}.\texttt{KJoin}(\cdot)$ to generate a joint public key jpk, which is then registered in Keys under the index $(\mathsf{grp}, \mathsf{sid})$.

As discussed in Remark 4, the key aggregation process must be executed using an MPC protocol since it requires the secret keys as inputs. Notably, this "sub-functionality" of $\mathcal{G}_{\mathsf{KRK}}$ aligns with the functionality of a (simple) MPC functionality $\mathcal{F}_{\mathsf{MPC}}$ introduced in prior works

[CD05, LTV12]. For simplicity, we directly include this sub-functionality as a part of $\mathcal{G}_{\mathsf{KRK}}$ in this thesisto reduce the number of hybrid games used in proofs. It is trivial to see that the security of the key aggregation process can be directly reduced to the security of the underlying MPC protocol.

Key Retrieval. A party $\mathcal{P}_{\mathsf{pid}}$ (the client or the server) and the adversary $\mathcal{S}$ can retrieve registered keys by sending the input $\langle \texttt{RegKey} : [\mathsf{pid}', \mathsf{sid}] \rangle$ to $\mathcal{G}_{\mathsf{KRK}}$. If no record exists in Keys, we let $\mathcal{G}_{\mathsf{KRK}}$ return $\perp$. Otherwise, it extracts the key pair $(\mathsf{pk}, \mathsf{sk})$ or the joint public key jpk if $\mathsf{pid}' = \mathsf{grp}$ from Keys. If the inputing party $\mathcal{P}_{\mathsf{pid}}$ is retrieving its own key pair (i.e., $\mathsf{pid} = \mathsf{pid}'$), or if the adversary $\mathcal{S}$ inputs the key pair of a corrupted party, then $\mathcal{G}_{\mathsf{KRK}}$ returns $\{\texttt{"key"} : [\mathsf{pid}, \mathsf{sid}, (\mathsf{pk}, \mathsf{sk})]\}$. Otherwise, it only returns the public key $\{\texttt{"key"} : [\mathsf{pid}, \mathsf{sid}, \mathsf{pk}]\}$ to the inputing party.

Corruption. The adversary $\mathcal{S}$ can send the input $\langle \texttt{Corrupt} : [\mathsf{pid}] \rangle$ to $\mathcal{G}_{\mathsf{KRK}}$ to corrupt a party. We assume that $\mathcal{G}_{\mathsf{KRK}}$ follows the pid-wise corruption [Can20]. Upon receiving this input, we let $\mathcal{G}_{\mathsf{KRK}}$ append the pid in the set $\mathsf{Corrupt}$. Note that during the key retrieval, if a party is corrupted, then we assume that its secret keys for all the instances of $\mathcal{G}_{\mathsf{KRK}}$ is retrievable by the adversary.

Remark 9. We note that the global key registry functionality introduced by Canetti et al. [CDPW07] does not align with our requirements. Their definition mandates that a party $\mathcal{P}_{\mathsf{pid}}$ must use the same public key across all sessions throughout its lifetime. However, in Section 5.4, we utilize Naor-Yung paradigm in our on-the-fly MPC protocol, which requires two *independent* public keys for two instances of the MGHE functionality. Thus we adopt the UC framework with global routines [BCH^{+}20] (UCGS), which allows us to create multiple sessions of the global functionality, rather than the generalized UC framework (GUC).

3.3.2 Ideal Functionality $\mathcal{F}_{\mathsf{MGHE}}$

In Figures 3.14 to 3.17, we present the ideal functionality $\mathcal{F}_{\mathsf{MGHE}}$ for a multi-group homomorphic encryption scheme.

Functionality: $\mathcal{F}_{\mathsf{MGHE}}$ (Part I)

Participants: n clients $\mathcal{P}_{\mathsf{cli}_1}, \mathcal{P}_{\mathsf{cli}_2}, \ldots, \mathcal{P}_{\mathsf{cli}_n}$. At a state, a subset of clients $\{\mathcal{P}_{\mathsf{cli}_i}\}_{i \in I}$ with $I \subseteq [n]$ form k groups $\mathcal{P}_{\mathsf{grp}_j}$ with $\mathsf{grp}_j = \{\mathsf{cli}_i\}_{i \in I_j}$ with $I_j \subseteq I$ for $j \in [k]$. A server $\mathcal{P}_{\mathsf{srv}}$, and an ideal-process adversary $\mathcal{S}$.

Initialization

```
1 : $\mathsf{MsgRec} := [\,]$ // Record for plaintext-ciphertext pairs indexed by $(\mathsf{grp}, \mathsf{sid})$.
2 : $\mathsf{Corrupt} := \varnothing$ // A set of corrupted party.
```

Corruption

```
1 : In $\langle \texttt{Corrupt} : [\mathsf{pid}] \rangle \leftarrow \mathcal{S}$
2 :     $\mathsf{Corrupt} := \mathsf{Corrupt} \cup \{\mathsf{pid}\}$
```

Figure 3.14: Ideal functionality $\mathcal{F}_{\mathsf{MGHE}}$ (Part I – Initialization and Corruption) for a multi-group homomorphic encryption (MGHE) scheme.

Initialization, Corruption, and Setup. Before detailing the functionality of each part in the system, in Figure 3.14, we first describe the internal variables used by $\mathcal{F}_{\mathsf{MGHE}}$, and handling of corruption. The following variables are defined at the initialization phase of the system, and their views are restricted exclusively to the functionality $\mathcal{F}_{\mathsf{MGHE}}$.

– *Message Record* MsgRec*:* A lookup table indexed by $(\mathsf{grp}, \mathsf{sid})$, where each entry is a list (initially empty). We use MsgRec to track the messages registered for a group grp during a session sid.

– *Corrupted Party Set* $\mathsf{Corrupt}$*:* A set (initialized as $\varnothing$) that tracks parties corrupted by the adversary.

We similarly assume pid-wise corruption as in $\mathcal{G}_{\mathsf{KRK}}$. Also, we assume *static corruption* of clients and the server after their initialization (including parameter and key generation) and before any computation. Thus after a party is corrupted, all messages intended for it are then instead delivered to $\mathcal{S}$, which means no messages are output to a party prior to its corruption. To simplify, this behavior is omitted from the description of corruption.

For setup and key generation, we utilize the key registration functionality illustrated in Figure 3.13. For each input to generate the public parameter or the key pair, or to join the individual public keys to obtain a joint public key for a group, we let the respective party send inputs to $\mathcal{G}_{\mathsf{KRK}}$.

Functionality: $\mathcal{F}_{\mathsf{MGHE}}$ (Part II)

Encryption

```
In $\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}, m; \boxed{\omega}] \rangle \leftarrow \{\mathcal{P}_{\mathsf{cli}}, \mathcal{S}\}$
    Out $\langle \texttt{RetKey} : [\mathsf{grp}, \mathsf{sid}] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
    In $\{ \texttt{"key"} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}'] \} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
    if $\mathsf{jpk}' \neq \mathsf{jpk}$ then
        Abort
    if $|\mathsf{grp} \cap \mathsf{Corrupt}| < |\mathsf{grp}|/t$
      $\wedge \langle \texttt{Encrypt} : [\cdot] \rangle \leftarrow \mathcal{P}_{\mathsf{cli}} \wedge \mathsf{cli} \notin \mathsf{Corrupt}$ then
        Out $\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}, |m|] \rangle \rightarrow \mathcal{S}$
        In $\{ \texttt{"ct"} : [\mathsf{grp}, \mathsf{sid}, |m|; \omega, c] \} \leftarrow \mathcal{S}$
    elseif $\langle \texttt{Encrypt} : [\cdot] \rangle \leftarrow \mathcal{S}$ then
        Out $\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}, m; \omega] \rangle \rightarrow \mathcal{S}$
        In $\{ \texttt{"ct"} : [\mathsf{grp}, \mathsf{sid}, m; \omega, c] \} \leftarrow \mathcal{S}$
    else
        Out $\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}, m] \rangle \rightarrow \mathcal{S}$
        In $\{ \texttt{"ct"} : [\mathsf{grp}, \mathsf{sid}, m; \omega, c] \} \leftarrow \mathcal{S}$
    if $(\cdot, c) \in \mathsf{MsgRec}$ then
        Abort
    $\delta := \mathsf{GetNoise}(\omega)$
    $\mathsf{MsgRec}[(\mathsf{grp}, \mathsf{sid})] := \mathsf{MsgRec}[(\mathsf{grp}, \mathsf{sid})] \bowtie (m; \delta, c)$
    Out $\{ \texttt{"ct"} : [\mathsf{grp}, \mathsf{sid}, c] \} \rightarrow \{\mathcal{P}_{\mathsf{cli}}, \mathcal{S}\}$
```

Figure 3.15: Ideal functionality $\mathcal{F}_{\mathsf{MGHE}}$ (Part II – Encryption) for a multi-group homomorphic encryption (MGHE) scheme. The dot-boxed parts are exclusive inputs from $\mathcal{S}$. The highlighted parts are exclusively for schemes on exact numbers.

Encryption. In Figure 3.15, we present the functionality for encryption. A client $\mathcal{P}_{\mathsf{cli}}$ or the adversary $\mathcal{S}$ may send a input

$$\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}, m] \rangle$$

to ask $\mathcal{F}_{\mathsf{MGHE}}$ to encrypt a message m for a group grp. The functionality first verifies the public key jpk by querying $\mathcal{G}_{\mathsf{KRK}}$ to retrieve the registered public key jpk' for the group and comparing it with jpk. Based on the corruption state of the group grp and the party who sends the input, we consider the following three cases:

- It has that $\mathsf{jpk}' = \mathsf{jpk}$, fewer than $\frac{|\mathsf{grp}|}{t}$ clients in the group grp are corrupted, and the party sending input to $\mathcal{F}_{\mathsf{MGHE}}$ is an *uncorrupted* client. In this case, the adversary should only learn the message length, which models the *client-side confidentiality* property of the MGHE scheme. In this case, $\mathcal{F}_{\mathsf{MGHE}}$ forwards the input to $\mathcal{S}$ but only includes message length $|m|$.
- The adversary $\mathcal{S}$ provides input to $\mathcal{F}_{\mathsf{MGHE}}$ (on behalf of a corrupted client) [8]. We allow $\mathcal{S}$ to specify the message m, as well as the random coins ω to be used for encryption (as in the dotted-box in Figure 3.15). In this case, we let $\mathcal{F}_{\mathsf{MGHE}}$ send back this message to $\mathcal{S}$, which then allows $\mathcal{S}$ to simulate the behavior of a semi-malicious adversary in the real world (which will be further discussed in Sections 3.3.3.3 and 3.3.3.4).
- The inputs contains an adversarially-controlled key, or the adversary has corrupted a sufficient number of clients in grp to enable decryption, but the input is from an *honest* client $\mathcal{P}_{\mathsf{cli}}$. Since the adversary has controlled enough resource to decrypt, we let $\mathcal{F}_{\mathsf{MGHE}}$ forward the input to $\mathcal{S}$ including the actual content of m.

We then let $\mathcal{S}$ return the ciphertext c. In case of approximate schemes, we let $\mathcal{S}$ additionally return the random coins ω used during encryption. We let $\mathcal{F}_{\mathsf{MGHE}}$ check if an entry $(\cdot, c)$ has already existed in the message record MsgRec. If it does, the input is aborted; otherwise, the pair (m, c) is added to MsgRec for further reference. In case of approximate schemes, we define a function GetNoise to get the actual noise δ from the random coin ω. (Indeed, ω can contain the noise and other randomness). Then we append $(m; \delta, c)$ to track the noise incurred during the execution.

Evaluation. In Figure 3.16, we describe the evaluation process in $\mathcal{F}_{\mathsf{MGHE}}$. A server $\mathcal{P}_{\mathsf{srv}}$ (or the adversary $\mathcal{S}$ on the behalf of a corrupted server) may send the input

$$\left\langle \texttt{Evaluate} : \left[\mathsf{sid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j \in [\ell]}\right] \right\rangle$$

to input $\mathcal{F}_{\mathsf{MGHE}}$ to evaluate a circuit f over ℓ ciphertexts. First, $\mathcal{F}_{\mathsf{MGHE}}$ verifies that all ℓ ciphertexts belong to the same session, i.e., $\forall i, j \in [\ell], \mathsf{sid}_i = \mathsf{sid}_j$, and that the session identifier matches the one input by $\mathcal{P}_{\mathsf{srv}}$, i.e., $\forall j \in [\ell], \mathsf{sid}_j = \mathsf{sid}$. Additionally, $\mathcal{F}_{\mathsf{MGHE}}$ checks for the presence of an entry (m_j, c_j) in $\mathsf{MsgRec}[(\mathsf{grp}_j, \mathsf{sid}_j)]$ for each $j \in [\ell]$. If any of these checks fail, $\mathcal{F}_{\mathsf{MGHE}}$ aborts on the input.

For each $j \in [\ell]$, the corresponding plaintext m_j (and the existing noise δ_j in the respective ciphertext in case of approximate schemes) is retrieved from the message record MsgRec, and the evaluation result in plaintext is computed as $\hat{m} = f(m_1, \ldots, m_\ell)$. A new group of clients is then formed with ID $\hat{\mathsf{grp}} = \cup_{j \in [\ell]} \mathsf{grp}_j$, representing the union of all client IDs involved in the evaluation. Each grp_j may correspond to an initial group of clients or an aggregated group from

[8]We use notation $\langle \cdot \rangle \twoheadleftarrow \mathcal{S}$ to indicate that $\mathcal{S}$ sends a backdoor message to the functionality for the instruction of the corrupted party $\mathcal{P}_{\mathsf{pid}}$. The functionality behaves as if receiving that input from $\mathcal{P}_{\mathsf{pid}}$.

Functionality: $\mathcal{F}_{\mathsf{MGHE}}$ (Part III)

Evaluation

```
In $\langle \texttt{Evaluate} : [\mathsf{sid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j\in[\ell]}; \boxed{\sigma}] \rangle \leftarrow \{\mathcal{P}_{\mathsf{srv}}, \mathcal{S}\}$
  if $\exists i, j \in [\ell] : (\mathsf{sid}_i \neq \mathsf{sid}_j \vee \mathsf{sid}_j \neq \mathsf{sid})$ then
    Abort
  if $\exists\, j \in [\ell] : (\cdot, c_j) \notin \mathsf{MsgRec}[(\mathsf{grp}_j, \mathsf{sid}_j)]$
    Abort
  for $j = 1, \ldots, \ell$ do
    Out $\langle \texttt{RetKey} : [\mathsf{grp}_j, \mathsf{sid}] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
    In $\{ \texttt{"key"} : [\mathsf{grp}_j, \mathsf{sid}, \mathsf{jpk}'_j] \} \rightarrow \mathcal{G}_{\mathsf{KRK}}$
    if $\mathsf{jpk}_j \neq \mathsf{jpk}'_j$ then
      Abort
    $(m_j; \delta_j, c_j) := \mathsf{MsgRec}[(\mathsf{grp}_j, \mathsf{sid})]$
  $\hat{\mathsf{grp}} := \cup_{j\in[\ell]} \mathsf{grp}_j$
  $\hat{m} := f(m_1, \ldots, m_\ell)$
  //Server is corrupted. input from $\mathcal{S}$.
  if $\langle \texttt{Evaluate} : [\cdot] \rangle \leftarrow \mathcal{S} \wedge \mathsf{srv} \in \mathsf{Corrupt}$ then
    Out $\langle \texttt{Evaluate} : [\mathsf{sid}, f, \hat{m}, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j\in[\ell]}; \sigma] \rangle \rightarrow \mathcal{S}$
    In $\{ \texttt{"ct"} : [\hat{\mathsf{grp}}, \mathsf{sid}, \hat{m}; \sigma, \hat{c}] \} \leftarrow \mathcal{S}$
    $\hat{\delta} := \mathsf{GetNoise}(f, \{\delta\}_{j\in[\ell]}, \sigma)$
  else
    Out $\langle \texttt{Evaluate} : [\mathsf{sid}, \sqcup, \hat{m}, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j\in[\ell]}] \rangle \rightarrow \mathcal{S}$
    In $\{ \texttt{"ct"} : [\hat{\mathsf{grp}}, \mathsf{sid}, (\hat{m}, f^*, \sigma), \hat{c}] \} \leftarrow \mathcal{S}$
    $\hat{\delta} := \mathsf{GetNoise}(f^*, \{\delta\}_{j\in[\ell]}, \sigma)$
  if $(\cdot, \hat{c}) \in \mathsf{MsgRec}$ then
    Abort
  $\mathsf{MsgRec}[(\hat{\mathsf{grp}}, \mathsf{sid})] := \mathsf{MsgRec}[(\hat{\mathsf{grp}}, \mathsf{sid})] \bowtie (\hat{m}; \hat{\delta}, \hat{c})$
  Out $\{ \texttt{"ct"} : [\hat{\mathsf{grp}}, \mathsf{sid}, \hat{c}] \} \rightarrow \{\mathcal{P}_{\mathsf{srv}}, \mathcal{S}\}$
```

Figure 3.16: Ideal functionality $\mathcal{F}_{\mathsf{MGHE}}$ (Part III – Circuit Evaluation) for a multi-group homomorphic encryption scheme. The dot-boxed parts are exclusive inputs from $\mathcal{S}$. The highlighted parts are exclusively for schemes on exact numbers.

previous evaluations, maintaining a recursive record of all participants who contributed to the computation.

If the input is from $\mathcal{S}$ and the server is corrupted, we let $\mathcal{S}$ additionally specify the random coin σ that will be used for evaluation (note that here the random coin is basically noise e.g. for ciphertext sanitization, so we directly use σ to represent the noise.). Then we simply have $\mathcal{F}_{\mathsf{MGHE}}$ send back this input $\mathcal{S}$ to let it simulate a real-world semi-malicious adversary to obtain the evaluated ciphertext $\hat{c}$. In case of approximate schemes, we then calculate the noise in $\hat{c}$ after this evaluation as $\hat{\delta} := \mathsf{GetNoise}(f^*, \{\delta\}_{j\in[\ell]}, \sigma)$ where $\mathsf{GetNoise}(\cdot)$ is an efficiently computable function that depends on the circuit f, current errors in the ciphertexts $\{\delta\}_{j\in[\ell]}$ and additional sanitizing noise σ.

Otherwise, if the input is from an honest server $\mathcal{P}_{\mathsf{srv}}$, we then let $\mathcal{F}_{\mathsf{MGHE}}$ forward this input to $\mathcal{S}$. Note that we let $\mathcal{F}_{\mathsf{MGHE}}$ include $\hat{m}$ when forwarding since we use MGHE as a building block for a MPC protocol and the adversary is allowed to learn the result by the security definition of a MPC protocol. To model circuit privacy, we let $\mathcal{F}_{\mathsf{MGHE}}$ hide the circuit f when forwarding (we use $\sqcup$ as a placeholder for that). That is, given the evaluation in plaintext and without knowing the circuit, $\mathcal{S}$ should be able to simulate an evaluated ciphertext $\hat{c}$ that is indistinguishable. In case of approximate schemes, we let $\mathcal{S}$ additionally send back the random circuit f^* and the random coins σ that it uses to generate $\hat{c}$. Similarly, we calculate the noise in $\hat{c}$ after this evaluation as $\hat{\delta} := \mathsf{GetNoise}(f^*, \{\delta\}_{j\in[\ell]}, \sigma)$ but with the random circuit f^* instead this time.

Similarly upon receiving the ciphertext $\hat{c}$ from $\mathcal{S}$, we let $\mathcal{F}_{\mathsf{MGHE}}$ check if there exists an entry $(\cdot, \hat{c})$ in the lookup table MsgRec. If yes, then we let $\mathcal{F}_{\mathsf{MGHE}}$ abort on this input. Otherwise, we let $\mathcal{F}_{\mathsf{MGHE}}$ append $(\hat{m}, \hat{c})$ (in case of approximate schemes, also $\hat{\delta}$) to MsgRec and send it back to the party sending the input.

Decryption. In Figure 3.17, we outline the decryption process in $\mathcal{F}_{\mathsf{MGHE}}$. This process is divided into two phases. A client first retrieves its partial-decryption, and then it combines partial-decryptions from other clients to recover the final result.

– To input a partial-decryption of a ciphertext c from $\mathcal{F}_{\mathsf{MGHE}}$, a client $\mathcal{P}_{\mathsf{cli}}$ or the ideal-process adversary $\mathcal{S}$ (on the behalf of a corrupted client) sends the input

$$\langle \mathtt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c] \rangle.$$

Upon receiving this input, $\mathcal{F}_{\mathsf{MGHE}}$ first checks whether there is an entry (m, c) for some plaintext m in the lookup table MsgRec that is for the group grp. If no matching exists, we let $\mathcal{F}_{\mathsf{MGHE}}$ abort on this input. Otherwise, it retrieves the plaintext message m associated with c from MsgRec. We note that for both exact and approximate schemes, we only extract the exact value m (without error) for simulation. Based on the party who sends the input, we consider the following two cases:

- If the client $\mathcal{P}_{\mathsf{cli}}$ is not corrupted, we then let $\mathcal{F}_{\mathsf{MGHE}}$ forward the input to $\mathcal{S}$ but also includes the plaintext m as a *trapdoor*. The ideal-process adversary $\mathcal{S}$, without the knowledge of the honest client's secret key sk, should use the plaintext m to generate a partial-decryption d that is indistinguishable from one produced during a real execution. In case of approximate schemes, we let $\mathcal{S}$ additionally return the random coin η (e.g. for noise-smudging) used to generated the partial decryption.
- If the client is corrupted, we let $\mathcal{S}$ additionally select a random coin η to be used during partial decryption (similarly, here η is primarily the noise for smudging thus we directly use η to represent the noise). Then we let $\mathcal{F}_{\mathsf{MGHE}}$ send this input back to $\mathcal{S}$ (without including m). Note that $\mathcal{S}$ can retrieve the secret keys of the corrupted clients from $\mathcal{G}_{\mathsf{KRK}}$, which allows it to run $\mathsf{MGHE}.\mathtt{PDec}(\cdot)$ to obtain the partial decryption. We note that in this case, we still consider it as a valid partial decryption and record it in ShRec.

Once d is obtained, we let $\mathcal{F}_{\mathsf{MGHE}}$ check if there is an entry $(\cdot, \cdot, d)$ in a lookup table ShRec that tracks the partial-decryptions generated by each client. If yes, we then let $\mathcal{F}_{\mathsf{MGHE}}$ abort on this input. Otherwise, we let $\mathcal{F}_{\mathsf{MGHE}}$ append the tuple (c, m, d) (also η in case of approximate schemes) to ShRec, and then outputs d to the client $\mathcal{P}_{\mathsf{cli}}$ or the adversary $\mathcal{S}$.

We note that only the partial decryption of an honest party is simulated. This is because, in the real world, the adversary can directly perform $\mathsf{MGHE}.\mathtt{PDec}(\cdot)$ using the secret key of the corrupted party. Therefore, the distinction between the ideal world and the real world lies in the partial-decryptions performed by the honest parties.

Functionality: $\mathcal{F}_{\mathsf{MGHE}}$ (Part IV)

Decryption

```
In $\left\langle \texttt{Share} : \left[\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c; \boxed{\eta}\right] \right\rangle \leftarrow \{\mathcal{P}_{\mathsf{cli}}, \mathcal{S}\}$
    if $(\cdot, c) \notin \mathsf{MsgRec}[(\mathsf{grp}, \mathsf{sid})]$ then
        $\texttt{Abort}$
    $(m, c) := \mathsf{MsgRec}[(\mathsf{grp}, \mathsf{sid})]$
    if $\langle \texttt{Share} : [\cdot] \rangle \leftarrow \mathcal{P}_{\mathsf{cli}} \land \mathsf{cli} \notin \mathsf{Corrupt}$ then
        Out $\langle \texttt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c, m] \rangle \rightarrow \mathcal{S}$
        In $\{\texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c, d, \eta]\} \leftarrow \mathcal{S}$
    else
        Out $\langle \texttt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c; \eta] \rangle \rightarrow \mathcal{S}$
        In $\{\texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c, d, \eta]\} \leftarrow \mathcal{S}$
    if $(\cdot, \cdot, d) \in \mathsf{ShRec}$ then
        $\texttt{Abort}$
    $\mathsf{ShRec}[(\mathsf{grp}, \mathsf{sid})] := \mathsf{ShRec}[(\mathsf{grp}, \mathsf{sid})] \bowtie (c, m, d, \eta)$
    Out $\{\texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c, d]\} \rightarrow \{\mathcal{P}_{\mathsf{cli}}, \mathcal{S}\}$
In $\langle \texttt{Combine} : [\mathsf{grp}, \mathsf{sid}, c, \{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}'}] \rangle \leftarrow \{\mathcal{P}_{\mathsf{cli}}, \mathcal{S}\}$
    if $(\cdot, c) \notin \mathsf{MsgRec}[(\mathsf{grp}, \mathsf{sid})]$ then
        $\texttt{Abort}$
    // Not enough partial decryptions. Abort.
    if $(\mathsf{grp}' \not\subseteq \mathsf{grp}) \lor (|\mathsf{grp}' \cap \mathsf{grp}| < |\mathsf{grp}|/t)$ then
        $\texttt{Abort}$
    // All shares are obtained legitimately. Output the original message.
    if $\exists \mathsf{cli}_1, \ldots, \mathsf{cli}_{|\mathsf{grp}|/t} \in \mathsf{grp}' : (c, m, d_i, \eta) \in \mathsf{ShRec}[(\mathsf{grp}, \mathsf{sid})]$ then
        $(m; \delta, c) := \mathsf{MsgRec}[(\mathsf{grp}, \mathsf{sid})]$
        $\hat{\delta} := \delta + \sum_{i=1, \mathsf{cli}_i \in \mathsf{Corrupt}}^{|\mathsf{grp}|/t} \eta_i + \sum_{i=1, \mathsf{cli}_i \notin \mathsf{Corrupt}}^{|\mathsf{grp}|/t} \eta_i - \delta$
        Out $\left\{\texttt{"pt"} : \left[\mathsf{grp}, \mathsf{sid}, c, m + \hat{\delta}\right]\right\} \rightarrow \{\mathcal{P}_{\mathsf{cli}}, \mathcal{S}\}$
    else
        In $\{\texttt{"pt"} : [\mathsf{grp}, \mathsf{sid}, c, m^*]\} \leftarrow \mathcal{S}$
        Out $\{\texttt{"pt"} : [\mathsf{grp}, \mathsf{sid}, c, m^*]\} \rightarrow \mathcal{P}_{\mathsf{cli}}$
```

Figure 3.17: Ideal functionality $\mathcal{F}_{\mathsf{MGHE}}$ (Part IV – Decryption) for a multi-group homomorphic encryption scheme. The dot-boxed part is exclusively for schemes on exact numbers and the highlighted part is exclusively for schemes on approximate numbers.

– Finally, to combine these partial-decryptions, we let a client $\mathcal{P}_{\mathsf{cli}}$ send to $\mathcal{F}_{\mathsf{MGHE}}$ the input[9]

$$\left\langle \texttt{Combine} : [\mathsf{grp}, \mathsf{sid}, c, \{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}'}] \right\rangle.$$

We then let $\mathcal{F}_{\mathsf{MGHE}}$ verify that there exists a ciphertext c in MsgRec and that the client has obtained a sufficient number of partial decryptions from clients in the group grp, i.e.,

[9] By $\{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}'}$, we assume the client receives the partial decryption in the format $\{\texttt{"sh"} : [\mathsf{cli}_i, \mathsf{grp}, \mathsf{sid}, c, d]\}$ for $\mathsf{cli}_i \in \mathsf{grp}'$.

$\mathsf{grp}' \subseteq \mathsf{grp}$ and $|\mathsf{grp}' \cap \mathsf{grp}| \geq \frac{|\mathsf{grp}|}{t}$. If this condition is not met, the original message cannot be recovered, and we let $\mathcal{F}_{\mathsf{MGHE}}$ abort on this input [10].

Otherwise, $\mathcal{F}_{\mathsf{MGHE}}$ checks whether at least $\frac{|\mathsf{grp}|}{t}$ partial decryptions provided by clients in grp' were obtained honestly, meaning they correspond to tuples (c, m, d_i) recorded in ShRec. If both conditions hold, $\mathcal{F}_{\mathsf{MGHE}}$ retrieves the message m from ShRec. Depending if MGHE is designed to be on exact or approximate number, we consider the following two cases.

- If MGHE is on exact numbers, we have $\mathcal{F}_{\mathsf{MGHE}}$ directly output m to $\mathcal{P}_{\mathsf{cli}}$.
- Otherwise, we define an error term $\hat{\delta} := \delta + \sum_{i=1, \mathsf{cli}_i \in \mathsf{Corrupt}}^{|\mathsf{grp}|/t} \eta_i + \sum_{i=1, \mathsf{cli}_i \notin \mathsf{Corrupt}}^{|\mathsf{grp}|/t} \eta_i - \delta$ where δ represents the existing noise in the ciphertext up to this step (including noise added at encryption, propagated through evaluation, and added for sanitization), and each η_i corresponds to the noise added during partial decryption by the clients. We note that if the server or a client is corrupted, then σ and η is chosen by the adversary as defined earlier. Otherwise, they are the ones used by $\mathcal{S}$ during simulation.

  We note that in case of simulated partial decryption, we add the term $\eta_i - \delta$ since that the partial decryptions are simulated without the term δ. This models the general case that there may be more than one honest client who provides the partial decryption. Observe that in case of the only one honest client, this simply give us the summation of smudging error, as also formalized by Hwang et al. [HHK$^+$25] where the functionality outputs the summation of sanitization and smudging error without disclosing the actual ciphertext error. By the indistinguishability between this two worlds, we guaranteed that the reconstructed message does not disclose any information about the existing error in the ciphertext, thereby preventing attacks like the one proposed by Li and Micciancio [LM21].

If the condition is not satisfied, it means the adversary has corrupted a sufficient number of clients in the group and has output at least $\frac{|\mathsf{grp}|}{t}$ fake partial decryptions through these clients. In this case, the original message is reconstructed incorrectly. Thus, we let $\mathcal{S}$ directly inject a message m^*, which is then delivered to $\mathcal{P}_{\mathsf{cli}}$ at a later stage.

Additionally, note that in this phase of combining partial-decryptions, $\mathcal{S}$ is not invoked (unless it needs to inject a fake message). This highlights that combining partial-decryptions is a *non-interactive* process.

Remark 10 (Liberal versus standard MPC security)**.** We remark that we let $\mathcal{F}_{\mathsf{MGHE}}$ provide the exact message value to the simulator $\mathcal{S}$ to achieve a more general security notion applicable to both exact and approximate schemes. However, as noted by Hwang et al. [HHK$^+$25], this captures *liberal security* model introduced by Feigenbaum et al. [FIM$^+$01]. Specifically, in the liberal security model, the ideal-process adversary $\mathcal{S}$ has access to the exact computation result, whereas in the standard security model, only an approximate result is provided for simulation.

To formalize standard security for approximate schemes, $\mathcal{F}_{\mathsf{MGHE}}$ can be modified accordingly. In particular, in Figure 3.16, we omit $\hat{m}$ when forwarding the input to $\mathcal{S}$. Similarly, in Figure 3.17, we have $\mathcal{F}_{\mathsf{MGHE}}$ sample an error δ from a sufficiently large distribution. Then when we have $\mathcal{S}$ simulate the partial decryption, we send $\mathcal{S}$ with $m + \delta$. Similarly, we have $\mathcal{F}_{\mathsf{MGHE}}$ output $m + \delta$ to the client when partial decryptions are combined.

Recently, Hwang et al. [HHK$^+$25] observed that if each party independently samples its own error for sanitization and smudging during threshold decryption, only liberal security can be

[10]We refer to Remark 12 for why we abort the protocol if there are not enough partial decryptions.

achieved. To address this, they proposed a distributed sampling technique in which a single error term δ is first sampled and then additively shared among all parties, who use their respective shares for sanitization and smudging. This ensures that the total error remains independent of the number of parties, thereby achieving standard MPC security.

Remark 11. It is assumed that an honest party does not reveal its partial-decryption for a ciphertext that is the encryption of another party's input. Instead, it only provides the partial-decryption for evaluated ciphertexts generated by the server. However, since we are defining a functionality for an MGHE *scheme*, in Figure 3.17, we consider the general case where all ciphertexts are eligible to be reconstructed from partial-decryptions for completeness. In Section 5.4, we will refine the protocol to differentiate between two distinct types of messages.

3.3.3 Realizing $\mathcal{F}_{\mathsf{MGHE}}$ with $\mathcal{G}_{\mathsf{KRK}}$

In this section, we show the UC-realization of the functionality $\mathcal{F}_{\mathsf{MGHE}}$ with a protocol Π_{MGHE} against a *semi-malicious* adversary who honestly executes the protocol but is allowed to choose the random coins for encryption from an arbitrary random coins space chosen by the adversary. We additionally allow the adversary to output fake partial decryption through corrupted clients to capture decryption consistency.

3.3.3.1 Protocol Π_{MGHE}

We present the protocol Π_{MGHE} for multi-group homomorphic encryption, as illustrated in Figure 3.18. Specifically, when a client $\mathcal{P}_{\mathsf{cli}}$ is activated with the input to generate public parameters or create individual keys, the client interacts with the global key registry $\mathcal{G}_{\mathsf{KRK}}$ to process the corresponding input. Similarly, when a virtual group entity $\mathcal{P}_{\mathsf{grp}}$ is activated to generate the joint public key, it forwards the input to $\mathcal{G}_{\mathsf{KRK}}$. For message encryption, the client $\mathcal{P}_{\mathsf{cli}}$ first query $\mathcal{G}_{\mathsf{KRK}}$ to verify the correctness of the public key. If the key is correct, then it invokes $\mathsf{MGHE}.\mathtt{Enc}(\cdot)$ and then outputs the resulting ciphertext. Otherwise, it aborts the protocol.

When a server $\mathcal{P}_{\mathsf{srv}}$ is activated to evaluate a circuit f over ℓ ciphertexts, it first similarly checks that all the public keys are correct with respect to the groups by querying $\mathcal{G}_{\mathsf{KRK}}$ and checks that all the ciphertexts to be evaluated on are in the same session. If not, then it aborts the protocol. Otherwise, the server $\mathcal{P}_{\mathsf{srv}}$ then aggregates the group IDs as $\hat{\mathsf{grp}} = \cup_{j \in [\ell]} \mathsf{grp}_j$ and invokes $\mathsf{MGHE}.\mathtt{Eval}(\cdot)$ to get the evaluated ciphertext $\hat{c}$ and outputs it.

To obtain a partial decryption, a client $\mathcal{P}_{\mathsf{cli}}$ first inputs its secret key sk from $\mathcal{G}_{\mathsf{KRK}}$ and then executes $\mathsf{MGHE}.\mathtt{PDec}(\cdot)$ to generate a partial decryption d, which it outputs later. To combine partial decryptions, the client first verifies whether the set of partial decryptions $\{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}'}$ it has collected satisfies the following conditions: (1) all partial decryptions are from clients in the group, i.e., $\mathsf{grp}' \subseteq \mathsf{grp}$, and (2) the number of collected partial decryptions meets the threshold requirement, i.e., $|\mathsf{grp}' \cap \mathsf{grp}| \geq \frac{|\mathsf{grp}|}{t}$. If either condition is not met, the client aborts the protocol. Otherwise, it invokes the algorithm $\mathsf{MGHE}.\mathtt{Combine}(\cdot)$ to recover the final result and outputs it.

Remark 12 (Abortion when insufficient partial decryptions)**.** In Figure 3.18, we specify that a client (or the semi-malicious adversary) aborts the protocol if $\mathsf{grp}' \not\subseteq \mathsf{grp}$ or $|\mathsf{grp}' \cap \mathsf{grp}| < \frac{|\mathsf{grp}|}{t}$ when activated with the input to combine partial decryptions from clients in grp' for a message intended for grp.

We note that by IND-CPA$^{\mathrm{pD}}$, it implies that the adversary does not learn the underlying message when it does not have a sufficient number of partial decryptions (observe that allowing corruption already grants the adversary access to the partial decryptions of corrupted clients). Thus, we do not introduce an additional hybrid game for this security.

Protocol: Π_{MGHE}

Participants: n clients $\mathcal{P}_{\mathsf{cli}_1}, \mathcal{P}_{\mathsf{cli}_2}, \ldots, \mathcal{P}_{\mathsf{cli}_n}$. At a state, a subset of clients $\{\mathcal{P}_{\mathsf{cli}_i}\}_{i \in I}$ with $I \subseteq [n]$ form ℓ groups $\mathcal{P}_{\mathsf{grp}_j}$ with $\mathsf{grp}_j = \{\mathsf{cli}_i\}_{i \in I_j}$ with $I_j \subseteq I$ for $j \in [\ell]$. A server $\mathcal{P}_{\mathsf{srv}}$.

Setup & Key Generation

```
$\mathcal{P}_{\mathsf{cli}} \leftarrow \langle \texttt{RegParam} : [\mathsf{sid}] \rangle$
    Out $\langle \texttt{RegParam} : [\mathsf{sid}] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$

$\mathcal{P}_{\mathsf{grp}} \leftarrow \langle \texttt{JoinKey} : [\mathsf{grp}, \mathsf{sid}] \rangle$
    Out $\langle \texttt{JoinKey} : [\mathsf{grp}, \mathsf{sid}] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$

$\mathcal{P}_{\mathsf{cli}} \leftarrow \langle \texttt{RegKey} : [\mathsf{cli}, \mathsf{sid}] \rangle$
    Out $\langle \texttt{RetParam} : [\mathsf{sid}] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
    In $\{\texttt{"param"} : [\mathsf{sid}, \mathsf{pp}]\} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
    Out $\langle \texttt{RegKey} : [\mathsf{cli}, \mathsf{sid}, \mathsf{pp}] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
```

Encryption

```
$\mathcal{P}_{\mathsf{cli}} \leftarrow \langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}, m] \rangle$
    Out $\langle \texttt{RetKey} : [\mathsf{grp}, \mathsf{sid}] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
    In $\{\texttt{"key"} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}']\} \rightarrow \mathcal{G}_{\mathsf{KRK}}$
    if $\mathsf{jpk} \neq \mathsf{jpk}'$ then
        Abort
    $c \leftarrow \mathsf{MGHE.Enc}(\mathsf{jpk}, m)$
    Out $\{\texttt{"ct"} : [\mathsf{grp}, \mathsf{sid}, c]\}$
```

Evaluation

```
$\mathcal{P}_{\mathsf{srv}} \leftarrow \langle \texttt{Evaluate} : [\mathsf{sid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j \in [\ell]}] \rangle$
    for $j = 1, \ldots, \ell$ do
        Out $\langle \texttt{RetKey} : [\mathsf{grp}_j, \mathsf{sid}] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
        In $\{\texttt{"key"} : [\mathsf{grp}_j, \mathsf{sid}, \mathsf{jpk}'_j]\} \rightarrow \mathcal{G}_{\mathsf{KRK}}$
        if $\mathsf{jpk}_j \neq \mathsf{jpk}'_j$ then
            Abort
    if $\exists i, j \in [\ell] : \mathsf{sid}_i \neq \mathsf{sid}_j \vee \mathsf{sid}_j \neq \mathsf{sid}$ then
        Abort
    $\hat{\mathsf{grp}} := \cup_{j \in [\ell]} \mathsf{grp}_j$
    $\hat{c} \leftarrow \mathsf{MGHE.Eval}(f, (\mathsf{jpk}_j, c_j)_{j \in [\ell]})$
    Out $\{\texttt{"ct"} : [\hat{\mathsf{grp}}, \mathsf{sid}, c]\}$
```

Decryption

```
$\mathcal{P}_{\mathsf{cli}} \leftarrow \langle \texttt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c] \rangle$
    Out $\langle \texttt{RetKey} : [\mathsf{cli}, \mathsf{sid}] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
    In $\{\texttt{"key"} : [\mathsf{cli}, \mathsf{sid}, (\mathsf{pk}, \mathsf{sk})]\} \rightarrow \mathcal{G}_{\mathsf{KRK}}$
    $d \leftarrow \mathsf{MGHE.PDec}(\mathsf{sk}, c)$
    Out $\{\texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c, d]\}$

$\mathcal{P}_{\mathsf{cli}} \leftarrow \langle \texttt{Combine} : [\mathsf{grp}, \mathsf{sid}, c, \{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}'}] \rangle$
    if $(\mathsf{grp}' \not\subseteq \mathsf{grp})$
     $\vee\ (|\mathsf{grp}' \cap \mathsf{grp}| < |\mathsf{grp}|/t)$ then
        Abort
    $m \leftarrow \mathsf{MGHE.Combine}(c, \{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}'})$
    Out $\{\texttt{"pt"} : [\mathsf{grp}, \mathsf{sid}, c, m]\}$
```

Figure 3.18: Protocol Π_{MGHE} of a multi-group homomorphic encryption.

3.3.3.2 Simulator $\mathcal{S}_{\mathsf{MGHE}}$

In Figures 3.19 to 3.22, we describe the simulator $\mathcal{S}_{\mathsf{MGHE}}$ attached to the functionality $\mathcal{F}_{\mathsf{MGHE}}$.

```
Simulator: $\mathcal{S}_{\mathsf{MGHE}}$ (Part I)

Initialization
$\mathsf{Corrupt} := \varnothing$ // A set of corrupted party
$\mathsf{CtrlKey} := [\,]$ // A list of keys of corrupted parties.

Corruption
In $\langle \mathtt{Corrupt} : [\mathsf{pid}] \rangle \leftarrow \mathcal{Z}$
    Out $\langle \mathtt{Corrupt} : [\mathsf{pid}] \rangle \rightarrow \mathcal{F}_{\mathsf{MGHE}}$
    Out $\langle \mathtt{Corrupt} : [\mathsf{pid}] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
    $\mathsf{Corrupt} := \mathsf{Corrupt} \cup \{\mathsf{pid}\}$
    if $\mathsf{pid} = \mathsf{cli}_i$ then
        // sid represents the current session.
        Out $\langle \mathtt{RetKey} : [\mathsf{cli}_i, \mathsf{sid}] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
        In $\{\texttt{"key"} : [\mathsf{cli}_i, \mathsf{pk}, \mathsf{sk}]\} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
        $\mathsf{CtrlKey}[(\mathsf{cli}_i, \mathsf{sid})] := \mathsf{sk}$
```

Figure 3.19: Simulator $\mathcal{S}_{\mathsf{MGHE}}$ (Part I - Initialization and Corruption) attached to the functionality $\mathcal{F}_{\mathsf{MGHE}}$.

Initialization & Corruption. Firstly, upon a message from the environment $\mathcal{Z}$ to corrupt a party $\mathcal{P}_{\mathsf{pid}}$, we let $\mathcal{S}$ forward this message to $\mathcal{F}_{\mathsf{MGHE}}$ and $\mathcal{G}_{\mathsf{KRK}}$. Then if it is a client to be corrupted, we then let $\mathcal{S}$ send the identifier for the current session sid to specify the secret keys in which session of $\mathcal{G}_{\mathsf{KRK}}$ it would like to access. Then we let $\mathcal{S}_{\mathsf{MGHE}}$ send the message to $\mathcal{G}_{\mathsf{KRK}}$ to retrieve the secret keys of the corrupted party and store it for later use. For all the other message sent by $\mathcal{Z}$ to a corrupted party, we simply have $\mathcal{S}$ forward this message (as a backdoor message) to $\mathcal{F}_{\mathsf{MGHE}}$.

Encryption. In the encryption phase, there are two types of inputs from $\mathcal{F}_{\mathsf{MGHE}}$ to $\mathcal{S}_{\mathsf{MGHE}}$: either with the message length $|m|$ or with the actual message m. If the input includes the message length $|m|$, we let $\mathcal{S}_{\mathsf{MGHE}}$ run $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}, 0^{|m|})$. Otherwise, we let $\mathcal{S}_{\mathsf{MGHE}}$ simply encrypt m to get c. Then we let $\mathcal{S}_{\mathsf{MGHE}}$ return c to $\mathcal{F}_{\mathsf{MGHE}}$. Observe that if a input for encryption from $\mathcal{F}_{\mathsf{MGHE}}$ does not specify the random coin, it means that the input is given by an honest client. Thus we let $\mathcal{S}$ use *honestly* sampled random coins from the random coins space for encryption to simulate the behavior of honest clients. (We note that, when building reduction, this random coin should be the same as in the real world. Indeed, in the left-or-right oracle, the same random coin is used for encryption of m_0 and m_1.) Otherwise, if the random coin is specified, we then let $\mathcal{S}$ use the random coin of its choice.

Note that the environment $\mathcal{Z}$'s input to a corrupted client $\mathcal{P}_{\mathsf{cli}}$ to encrypt m with a random coins ω chosen by $\mathcal{Z}$ is also delivered to $\mathcal{S}$ instead. We then let $\mathcal{S}$ forward this input to $\mathcal{F}_{\mathsf{MGHE}}$ [11] and output the value from $\mathcal{F}_{\mathsf{MGHE}}$ as the output of $\mathcal{P}_{\mathsf{cli}}$.

Evaluation. In the evaluation phase, the evaluation in plaintext $\hat{m}$ is sent by $\mathcal{F}_{\mathsf{MGHE}}$ to $\mathcal{S}_{\mathsf{MGHE}}$. We then consider two cases depending on if MGHE uses approximate or exact evaluation. If MGHE uses approximate evaluation, we let $\mathcal{S}_{\mathsf{MGHE}}$ sample a circuit f^* uniformly at random

[11] Intuitively, $\mathcal{S}$ could compute $\mathsf{MGHE}.\mathtt{Enc}(\cdot)$ directly without forwarding the input to $\mathcal{F}_{\mathsf{MGHE}}$. However, since we also need to log this message in MsgRec, we let $\mathcal{S}$ forward the input instead.

Simulator: $\mathcal{S}_{\mathsf{MGHE}}$ (Part II)

Encryption

```
// Simulating an honest client submitting message under an uncorrupted group.
In $\langle$Encrypt : [grp, sid, jpk, $|m|$]$\rangle \leftarrow \mathcal{F}_{\mathsf{MGHE}}$
    $\omega \leftarrow_{\mathcal{D}} \Omega$
    $c \leftarrow \mathsf{MGHE.Enc}(\mathsf{jpk}, 0^{|m|}; \omega)$
    Out {"ct" : [grp, sid, $|m|; \omega$, $c$]} $\rightarrow \mathcal{F}_{\mathsf{MGHE}}$
// Simulating an honest client submitting message under an corrupted group.
In $\langle$Encrypt : [grp, sid, jpk, $m$]$\rangle \leftarrow \mathcal{F}_{\mathsf{MGHE}}$
    $\omega \leftarrow_{\mathcal{D}} \Omega$
    $c \leftarrow \mathsf{MGHE.Enc}(\mathsf{jpk}, m; \omega)$
    Out {"ct" : [grp, sid, $m; \omega$, $c$]} $\rightarrow \mathcal{F}_{\mathsf{MGHE}}$
// Simulating corrupted client.
In $\langle$Encrypt : [grp, sid, jpk, $m; \omega$]$\rangle \leftarrow \mathcal{F}_{\mathsf{MGHE}}$
    $c \leftarrow \mathsf{MGHE.Enc}(\mathsf{jpk}, m; \omega)$
    Out {"ct" : [grp, sid, $|m|; \omega$, $c$]} $\rightarrow \mathcal{F}_{\mathsf{MGHE}}$
// Input to a corrupted client from $\mathcal{Z}$ is sent to $\mathcal{S}$. Forward the input to $\mathcal{F}_{\mathsf{MGHE}}$.
In $\langle$Encrypt : [grp, sid, jpk, $m; \omega$]$\rangle \leftarrow (\mathcal{P}_{\mathsf{cli}} \leftarrow \mathcal{Z})$
    Out $\langle$Encrypt : [grp, sid, jpk, $m; \omega$]$\rangle \rightarrow \mathcal{F}_{\mathsf{MGHE}}$
    In {"ct" : [grp, sid, $c$]} $\leftarrow \mathcal{F}_{\mathsf{MGHE}}$
    Out {"ct" : [grp, sid, $c$]} as $\mathcal{P}_{\mathsf{cli}}$
```

Figure 3.20: Simulator $\mathcal{S}_{\mathsf{MGHE}}$ (Part II - Encryption) attached to the functionality $\mathcal{F}_{\mathsf{MGHE}}$. The highlighted part is exclusively for schemes on approximate numbers.

from the circuit space $\mathcal{F}_\kappa$ i.e., all the circuits of maximum depth κ. Then $\mathcal{S}_{\mathsf{MGHE}}$ computes $\hat{c} \leftarrow \mathsf{MGHE.Eval}(f^*, (\mathsf{jpk}_j, c_j)_{j\in[\ell]})$ and outputs it to $\mathcal{F}_{\mathsf{MGHE}}$. We note that simulating with the same set of group public key-ciphertext tuples $(\mathsf{jpk}_j, c_j)_{j\in[\ell]}$ more closely reflects real-world cases, particularly in multi-party settings. This is because the clients and the server can establish a consensus on how each client's input should be assigned to specific positions in the circuit. As a result, the topology of the circuit is inherently revealed to the adversary (through some corrupted clients). As discussed in Section 3.2.2.1, this form of simulation allows us to establish the privacy of each gate value in the circuit. Otherwise, if it is a scheme with exact evaluation, we let $\mathcal{S}_{\mathsf{MGHE}}$ invoke the circuit-privacy simulator as in Definition 22 to obtain $\hat{c} \leftarrow \mathtt{Sim}_{circ}(\{\mathsf{jpk}_j\}_{j\in[\ell]}, \hat{m})$ and return $\hat{c}$ to $\mathcal{F}_{\mathsf{MGHE}}$.

In the case where the input also includes the circuit f and the random coin σ for evaluation, which means that the input is provided through a corrupted client. We then let $\mathcal{S}$ run $\mathsf{MGHE.Eval}(\cdot)$ with f and σ and return the evaluated ciphertext $\hat{c}$ to $\mathcal{F}_{\mathsf{MGHE}}$.

Similarly, $\mathcal{Z}$'s input to the corrupted server to evaluate a circuit will be delivered to $\mathcal{S}_{\mathsf{MGHE}}$ instead and we let $\mathcal{S}_{\mathsf{MGHE}}$ forward it to $\mathcal{F}_{\mathsf{MGHE}}$ and output the value from $\mathcal{F}_{\mathsf{MGHE}}$ as the output of the server $\mathcal{P}_{\mathsf{srv}}$.

Decryption. In the decryption phase, upon receiving a input from $\mathcal{F}_{\mathsf{MGHE}}$ to generate a partial-decryption d for a given ciphertext c and its underlying plaintext m, we let $\mathcal{S}_{\mathsf{MGHE}}$ include the secret keys sk_i's of the corrupted clients and forward query to the threshold simulator $\mathtt{Sim}_{th}$. In case of approximate schemes, we slightly abuse the notation to let $\mathtt{Sim}_{th}$ to additionally output the noise η that is added by $\mathtt{Sim}_{th}$. Given a ciphertext c and its corresponding plaintext m, and the

Simulator: $\mathcal{S}_{\mathsf{MGHE}}$ (Part III)

Evaluation

```
//Simulating an honest server.
In $\langle \texttt{Evaluate} : [\hat{\mathsf{grp}}, \mathsf{sid}, \perp, \hat{m}, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j \in [\ell]}] \rangle \leftarrow \mathcal{F}_{\mathsf{MGHE}}$
    $\boxed{\hat{c} \leftarrow \texttt{Sim}_{circ}(\{\mathsf{jpk}_j\}_{j \in [\ell]}, \hat{m})}$
    $f^* \leftarrow\$ \mathcal{F}_{\kappa}$
    $\sigma \leftarrow_{\mathcal{D}} \Sigma$
    $\hat{c} \leftarrow \mathsf{MGHE}.\texttt{Eval}(f^*, (\mathsf{jpk}_j, c_j)_{j \in [\ell]}; \sigma)$
    Out $\{\texttt{"ct"} : [\mathsf{grp}, \mathsf{sid}, (\hat{m}, f^*, \sigma), \hat{c}]\} \rightarrow \mathcal{F}_{\mathsf{MGHE}}$
//Simulating a corrupted server.
In $\langle \texttt{Evaluate} : [\hat{\mathsf{grp}}, \mathsf{sid}, f, \hat{m}, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j \in [\ell]}; \sigma] \rangle \leftarrow \mathcal{F}_{\mathsf{MGHE}}$
    $\hat{c} \leftarrow \mathsf{MGHE}.\texttt{Eval}(f, (\mathsf{jpk}_j, c_j)_{j \in [\ell]}; \sigma)$
    Out $\{\texttt{"ct"} : [\mathsf{grp}, \mathsf{sid}, \hat{c}]\} \rightarrow \mathcal{F}_{\mathsf{MGHE}}$
//Input to a corrupted client is sent to $\mathcal{S}$. Forward the input to $\mathcal{F}_{\mathsf{MGHE}}$
In $\langle \texttt{Evaluate} : [\mathsf{sid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j \in [\ell]}; \sigma] \rangle \leftarrow (\mathcal{P}_{\mathsf{srv}} \leftarrow \mathcal{Z})$
    Out $\langle \texttt{Evaluate} : [\mathsf{sid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j \in [\ell]}; \sigma] \rangle \rightarrow \mathcal{F}_{\mathsf{MGHE}}$
    In $\{\texttt{"ct"} : [\mathsf{grp}, \mathsf{sid}, \hat{c}]\} \leftarrow \mathcal{F}_{\mathsf{MGHE}}$
    Out $\{\texttt{"ct"} : [\mathsf{grp}, \mathsf{sid}, \hat{c}]\}$ as $\mathcal{P}_{\mathsf{srv}}$
```

Figure 3.21: Simulator $\mathcal{S}_{\mathsf{MGHE}}$ (Part III - Evaluation) attached to the functionality $\mathcal{F}_{\mathsf{MGHE}}$. The highlighted part is exclusively for schemes on approximate numbers. The framed part is exclusively for schemes on exact numbers.

secret keys $\{\mathsf{sk}_i\}_{i \in I_{\mathcal{A}}}$ of the corrupted client, the simulator $\texttt{Sim}_{th}$ outputs the partial-decryption d for an honest party. Then we let $\mathcal{S}_{\mathsf{MGHE}}$ forward this d to $\mathcal{F}_{\mathsf{MGHE}}$.

If the input does not include the underlying message m, we then let $\mathcal{S}$ first call $\mathcal{G}_{\mathsf{KRK}}$ to retrieve the secret key sk of the corrupted client. Then $\mathcal{S}$ simply runs $d \leftarrow \mathsf{MGHE}.\texttt{PDec}(\mathsf{sk}, c)$ to get the partial decryption and return d to $\mathcal{F}_{\mathsf{MGHE}}$.

Similarly, all the other inputs from $\mathcal{Z}$ to the corrupted clients to let it obtain the decryption share and combine to recover the result are then forward by $\mathcal{S}_{\mathsf{MGHE}}$ to $\mathcal{F}_{\mathsf{MGHE}}$ respectively. Upon receiving outputs from $\mathcal{F}_{\mathsf{MGHE}}$, we let $\mathcal{S}_{\mathsf{MGHE}}$ write it as the output of the corrupted client.

Additionally, we allow the environment to directly inject a partial-decryption (which may be fake). Note this partial decryption is not included in the lookup table ShRec by $\mathcal{F}_{\mathsf{MGHE}}$ but it is just output by the corrupted party.

3.3.3.3 Adversarial Model

For this part, we consider a *Partial-Decryption-Manipulating Semi-Malicious Adversary*, an extension of the *semi-malicious adversary* model introduced by Asharov et al. [AJL$^+$12]. A semi-malicious adversary falls between a *semi-honest* adversary, which follows the protocol honestly but attempts to learn information about honest parties' inputs, and a *malicious* adversary, which has no restrictions on its behavior and actively attacks the execution of the protocol. Unlike a standard semi-malicious adversary, we consider an adversary hat is additionally allowed to manipulate the decryption process by injecting false partial decryptions through corrupted parties,

Simulator: $\mathcal{S}_{\mathsf{MGHE}}$ (Part IV)

Decryption

```
1 : //Simulating the honest client.
2 : In ⟨Share : [cli, grp, sid, $c$, $m$]⟩ ↞ $\mathcal{F}_{\mathsf{MGHE}}$
3 :     for $\mathsf{cli}_i \in$ Corrupt ∩ grp do
4 :         $\mathsf{sk}_i :=$ CtrlKey[($\mathsf{cli}_i$, sid)]
5 :     ($\eta$,$d$) ← $\mathtt{Sim}_{th}(c, m, \{\mathsf{sk}_i\}_{\mathsf{cli}_i \in \mathsf{Corrupt} \cap \mathsf{grp}})$
6 :     Out {"sh" : [cli, grp, sid, $d$, $\eta$]} ↠ $\mathcal{F}_{\mathsf{MGHE}}$
```

```
1 : //Simulating the corrupted client.
2 : In ⟨Share : [cli, grp, sid, $c$; $\eta$]⟩ ↞ $\mathcal{F}_{\mathsf{MGHE}}$
3 :     sk := CtrlKey[(cli, sid)]
4 :     if sk = ⊥ then
5 :         Abort
6 :     $d$ ← MGHE.PDec(sk, $c$; $\eta$)
7 :     Out {"sh" : [cli, grp, sid, $d$, $\eta$]} ↠ $\mathcal{F}_{\mathsf{MGHE}}$
```

```
1 : //Input to a corrupted client from $\mathcal{Z}$ is sent to $\mathcal{S}$.
2 : In ⟨Share : [cli, grp, sid, $c$; $\eta$]⟩ ↞ ($\mathcal{P}_{\mathsf{cli}}$ ↞ $\mathcal{Z}$)
3 :     Out ⟨Share : [cli, grp, sid, $c$]⟩ ↠ $\mathcal{F}_{\mathsf{MGHE}}$
4 :     In {"sh" : [cli, grp, sid, $d$]} ↞ $\mathcal{F}_{\mathsf{MGHE}}$
5 :     Out {"sh" : [cli, grp, sid, $d$]} as $\mathcal{P}_{\mathsf{cli}}$
```

```
1 : //The adversary directly injects a partial decryption.
2 : In {"sh" : [cli, grp, sid, $d$]} ↞ ($\mathcal{P}_{\mathsf{cli}}$ ↞ $\mathcal{Z}$)
3 :     Out {"sh" : [cli, grp, sid, $d$]} as $\mathcal{P}_{\mathsf{cli}}$
```

```
1 : //Input to a corrupted client from $\mathcal{Z}$ is sent to $\mathcal{S}$. Forwad the input to $\mathcal{F}_{\mathsf{MGHE}}$.
2 : In ⟨Combine : [grp, sid, $c$, $\{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}'}$]⟩ ↞ ($\mathcal{P}_{\mathsf{cli}}$ ↞ $\mathcal{Z}$)
3 :     Out ⟨Combine : [grp, sid, $c$, $\{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}'}$]⟩ ↠ $\mathcal{F}_{\mathsf{MGHE}}$
4 :     In {"pt" : [grp, sid, $c$, $m$]} ↞ $\mathcal{F}_{\mathsf{MGHE}}$
5 :     Out {"pt" : [grp, sid, $c$, $m$]} as $\mathcal{P}_{\mathsf{cli}}$
```

Figure 3.22: Simulator $\mathcal{S}_{\mathsf{MGHE}}$ (Part IV - Decryption) attached to the functionality $\mathcal{F}_{\mathsf{MGHE}}$. The highlighted part is exclusively for schemes on approximate numbers.

potentially influencing the final reconstructed message.

A semi-malicious adversary is an *interactive Turing machine* (ITM) with an additional *witness tape*. When the adversary sends a message msg on behalf of a corrupted party $\mathcal{P}_{\mathsf{pid}}$, it must record on the witness tape the corresponding input-random coins pair $(x; r)$. These pairs must be consistent with the messages sent by the adversary, as if the protocol were being honestly executed. We stress that the random coin r should be in a valid range defined by the scheme even though that the adversary can choose it from an arbitrary distribution.

We assume that clients and servers are subject to *static corruption*. Following prior works [LTV12, MW16, Sma23, KLSW24], we say corruption is *static* if the adversary non-adaptively selects the subset of corrupted parties before any computation begins and after all the n clients and the server have been initialized. Additionally, we assume the adversary is *rushing* [CD05], meaning it can *adaptively* decide its messages after observing the protocol messages from honest parties in the current and all previous rounds.

3.3.3.4 Proof of Secure Realization

We show the security of the protocol with respect to two possible types of semi-malicious adversary. In Theorem 2, we first consider an adversary that corrupts both the server and a subset of clients. Note since the server is corrupted, we only need to consider client-side confidentiality in this case. This corresponds to the result established in prior literature [LTV12, MW16, KLSW24], that is, with MGHE alone, we can obtain an MPC that is secure under a semi-malicious adversary that corrupts the server and the a subset of clients (where only client-side confidentiality is guaranteed).

In Theorem 3 and Lemma 5, we then consider a semi-malicious adversary who corrupts a subset of clients in a group but the server remains also honest. Thus in this case, we need to show both client-side and server-side confidentiality.

In Figure 3.23, we show the high-level idea of hybrid games in the security proofs and reduction to the respective security notion for each hop between hybrid games.

$$\begin{tikzcd}
\boxed{\mathsf{REAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}} \arrow[r, bend right, "\substack{\text{DC}\\ \text{Claim 2.1}}"'] & \boxed{\mathsf{HYB}^{1\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}} \arrow[r, bend right, "\substack{\{\text{IND},\text{SIM}\}\text{-CIRC}\\ \text{Claims 3.1 and 3.3}}"'] \arrow[rr, bend left, "\substack{\text{IND-CPA}^{\text{pD}}\\ \text{Claim 2.2}}"] & \boxed{\mathsf{HYB}^{2\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}} \arrow[r, bend right, "\substack{\text{IND-CPA}^{\text{pD}}\\ \text{Claims 3.2 and 3.4}}"'] & \boxed{\mathsf{HYB}^{3\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}} \arrow[r, bend right, "\substack{\text{SIM-PDEC}\\ \text{Claim 2.3}}"'] & \boxed{\mathsf{IDEAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\mathcal{F}_{\mathsf{MGHE}},\mathcal{S},\mathcal{Z}}}
\end{tikzcd}$$

Figure 3.23: An illustration of hybrid games and reduction in the security proofs for $\mathcal{F}_{\mathsf{MGHE}}$ realization. The dot-boxed hybrid game is exclusively for Theorem 3 and Lemma 5 where the server is not corrupted.

Theorem 2. *Let* MGHE *be a multi-group homomorphic encryption scheme with threshold factor t. Then, the protocol Π_{MGHE} UC-realizes the functionality $\mathcal{F}_{\mathsf{MGHE}}$ in the presence of a global key registry $\mathcal{G}_{\mathsf{KRK}}$ against a partial-decryption manipulating semi-malicious adversary that non-adaptively corrupts parties, provided that:*

- MGHE *satisfies Decryption Consistency, and the adversary corrupts fewer than $\frac{|\mathsf{grp}|}{t}$ clients in any group.*
- MGHE *is* IND-CPA$^{\text{pD}}$*-secure, and there exists at least one uncorrupted client who provides input to the computation, such that the group* grp *under which it provides input contains no more than $\frac{|\mathsf{grp}|}{t}$ corrupted clients.*
- MGHE *satisfies* SIM-PDEC *security with respect to a simulator* $\mathtt{Sim}_{th}$.

Proof. We prove that the protocol Π_{MGHE} is correct, and it is secure and robust against corruption up to the threshold.

CORRECTNESS. The correctness of the protocol follows directly from the correctness of the MGHE scheme. For correctness, we assume that no adversary launches any attack and no party in the system is corrupted (we use $\perp$ to indicate no presence of adversary).

When a client $\mathcal{P}_{\mathsf{cli}}$ encrypts a message m, the plaintext-ciphertext pair (m, c) is recorded in MsgRec. When the server $\mathcal{P}_{\mathsf{srv}}$ evaluates a circuit f over the ciphertexts $\{c_j\}_{j\in[\ell]}$, the corresponding plaintexts $\{m_j\}_{j\in[\ell]}$ are retrieved from MsgRec, and the plaintext result is computed as $\hat{m} = f(\{m_j\}_{j\in[\ell]})$. The pair $(\hat{m}, \hat{c})$ is then recorded in MsgRec. When a client $\mathcal{P}_{\mathsf{cli}}$ obtains the partial-decryption d for a ciphertext c, the pair (m, c) is retrieved from MsgRec, and the tuple (c, m, d) is recorded in ShRec.

In $\mathcal{F}_{\mathsf{MGHE}}$, when a client $\mathcal{P}_{\mathsf{cli}}$ inputs to combine partial-decryptions to obtain the final result, if all partial-decryptions $(c, d, \cdot)$ are available, the original plaintext m is directly output to the client by retrieving m from the tuple (c, d, m). Thus, in the ideal world, correctness is independent of the ciphertext c and partial-decryption d; these only serve to identify the plaintext m in MsgRec and ShRec. Consequently, following Definition 9 for the correctness in the real world, we have that

$$\Delta_{\mathcal{Z}}\left(\mathsf{REAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\perp,\mathcal{Z}}, \mathsf{IDEAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\mathcal{F}_{\mathsf{MGHE}},\mathcal{S}_{\mathsf{MGHE}},\mathcal{Z}}\right) \leq \varepsilon \tag{3.1}$$

where ε represents the correctness parameter of the MGHE scheme.

SECURITY. We consider n clients $\mathcal{P}_{\mathsf{cli}_1}, \ldots, \mathcal{P}_{\mathsf{cli}_n}$ and a server $\mathcal{P}_{\mathsf{srv}}$ in the system. In the setup phase, we let a client $\mathcal{P}_{\mathsf{cli}_i}$, for some $i \in [n]$, be first activated with the input $\langle \texttt{RegParam} : [\mathsf{sid}] \rangle$ to initialize the system's public parameter at $\mathcal{G}_{\mathsf{KRK}}$. Following this, each client $\mathcal{P}_{\mathsf{cli}_i}$ receives the input $\langle \texttt{RegKey} : [\mathsf{cli}, \mathsf{sid}] \rangle$ to generate their respective key pairs $(\mathsf{pk}_i, \mathsf{sk}_i)$ at $\mathcal{G}_{\mathsf{KRK}}$.

Next, we form k initial groups $\mathsf{grp}_1, \ldots, \mathsf{grp}_k$, where each group $\mathsf{grp}_j = \{\mathsf{cli}_i\}_{i \in I_j}$ includes a subset of clients indexed by $I_j \subseteq [n]$ for $j \in [k]$. For every group, the corresponding "virtual" group entity $\mathcal{P}_{\mathsf{grp}_j}$ is provided with input $\left\langle \texttt{JoinKey} : [\mathsf{grp}_j, \mathsf{sid}] \right\rangle$ to compute joint public key jpk_j at $\mathcal{G}_{\mathsf{KRK}}$.

Then the adversary corrupts a subset of clients by sending backdoor messages and queries $\mathcal{G}_{\mathsf{KRK}}$ to access the secret keys sk_i of the corrupted clients. Notably, during this setup phase, $\mathcal{G}_{\mathsf{KRK}}$ does these operations for both of the real and ideal worlds. Thus these two worlds remain indistinguishable to the environment $\mathcal{Z}$ at this stage.

To prove the security of the protocol, we then define the following sequence of hybrid games:

- $\mathbf{REAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$: This represents the execution of the protocol Π_{MGHE} in the real world and an adversary $\mathcal{A}$ attacking the protocol.
- $\mathbf{HYB}^{1\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$: In this hybrid game, we keep track of the message-ciphertext tuples (m, c) that are encrypted and evaluated, as well as the partial decryptions d that are honestly generated with respect to (m, c). When a client combines the partial decryptions, if it has obtained a sufficient number of honest partial decryptions, it outputs m. For schemes on approximate number, it outputs $m + \delta_{\mathsf{ct}} + \sigma + \sum_{i=1}^{|\mathsf{grp}|/t} \eta_i$.
- $\mathbf{HYB}^{2\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$: In this hybrid game, whenever an honest client $\mathcal{P}_{\mathsf{cli}}$ is activated with the input $\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}, m] \rangle$, the client outputs $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}, 0^{|m|})$.
- $\mathbf{IDEAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\mathcal{F}_{\mathsf{MGHE}}, \mathcal{S}_{\mathsf{MGHE}}, \mathcal{Z}}$: This is the ideal execution where the ideal functionality $\mathcal{F}_{\mathsf{MGHE}}$ is attached to the simulator $\mathcal{S}_{\mathsf{MGHE}}$ and interacts with dummy client $\mathcal{P}_{\mathsf{cli}_1}, \ldots, \mathcal{P}_{\mathsf{cli}_n}$.

Claim 2.1. *If* MGHE *satisfies Decryption Consistency, then* $\mathbf{REAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$ *and* $\mathbf{HYB}^{1\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$ *are computationally indistinguishable.*

Proof. We prove the claim by showing that if there is an environment $\mathcal{Z}$ that distinguishes between $\mathbf{REAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$ and $\mathbf{HYB}^{1\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$, then we can use it to construct an adversary $\mathcal{A}_{dc}$ that breaks the decryption consistency of MGHE. For this, we let $\mathcal{A}_{cpa}$ simulate the interaction between the environment $\mathcal{Z}$ and the two worlds.

We let the adversary $\mathcal{A}_{dc}$ form the same groups $\mathsf{grp}_1, \ldots, \mathsf{grp}_k$ by specifying the index sets $I_1, \ldots, I_k$ in the $\mathrm{G}^{\mathrm{DC}}_{\mathsf{MGHE}}$ game and corrupts the same clients by specifying the index set $I_{\mathcal{A}}$ as in the setup phase. When a client queries $\mathcal{G}_{\mathsf{KRK}}$ to retrieve the group public key, we let $\mathcal{A}_{dc}$ provides the one generated by $\mathrm{G}^{\mathrm{DC}}_{\mathsf{MGHE}}$. Similarly, when the adversary $\mathcal{A}$ inputs $\mathcal{G}_{\mathsf{KRK}}$ for the key pairs of corrupted clients, we let $\mathcal{A}_{dc}$ respond with $(\mathsf{pk}_i, \mathsf{sk}_i)$ for $i \in I_{\mathcal{A}}$, which is also generated by the game $\mathrm{G}^{\mathrm{DC}}_{\mathsf{MGHE}}$.

- **On input** $\left\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}_j, m] \right\rangle$ **to an honest client** $\mathcal{P}_{\mathsf{cli}}$: We then let $\mathcal{A}$ honestly sample a random coin r from the distribution and query its oracle $\mathcal{O}_{\mathsf{Enc}}$ with $(m; \omega)$. We observe that $\mathcal{O}_{\mathsf{Enc}}$ does $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m; \omega)$, which is exactly the behavior of an honest client in both of these worlds. Thus we write c from $\mathcal{O}_{\mathsf{Enc}}$ as output of $\mathcal{P}_{\mathsf{cli}}$.

– **On input** $\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}_j, m; \omega] \rangle$ **to a corrupted client** $\mathcal{P}_{\mathsf{cli}}$: We similarly let $\mathcal{A}$ query its oracle $\mathcal{O}_{\mathsf{Enc}}$ with $(m; \omega)$ where ω is the random coin selected by $\mathcal{Z}$. Then upon getting c from $\mathcal{O}_{\mathsf{Enc}}$, we write it as output of $\mathcal{P}_{\mathsf{cli}}$.

– **On input** $\langle \texttt{Evaluate} : [\mathsf{sid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j \in [\ell]}; \sigma] \rangle$ **to a corrupted server** $\mathcal{P}_{\mathsf{srv}}$: We observe by definition of these two worlds, all ciphertexts to be evaluated must either be encrypted by the clients or result from an evaluation performed by the server. This implies that in $\mathrm{G}^{\mathrm{DC}}_{\mathsf{MGHE}}$, these ciphertexts are all entries in the lookup table $\mathcal{T}$. Thus, we let $\mathcal{A}_{dc}$ identify these ciphertexts in the lookup table $\mathcal{T}$ by specifying an index set L and query the oracle $\mathcal{O}_{\mathsf{Eval}}$ with $(f, L; \sigma)$. Observe that the oracle $\mathcal{O}_{\mathsf{Eval}}$ returns the ciphertext $\hat{c} \leftarrow \mathsf{MGHE}.\texttt{Eval}(f, (\mathcal{T}[l].\mathsf{jpk}, \mathcal{T}[l].c)_{l \in L}; \sigma)$. This is exactly the behavior of the server in both of these worlds. Thus we write $\hat{c}$ as the output of $\mathcal{P}_{\mathsf{srv}}$.

– **On input** $\langle \texttt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c] \rangle$ **to an honest** $\mathcal{P}_{\mathsf{cli}}$: Observe that we require that c must be either encrypted by clients or an evaluation performed by the server. Thus we similarly let $\mathcal{A}_{dc}$ identify the index l of c in the lookup table $\mathcal{T}$ of $\mathrm{G}^{\mathrm{DC}}_{\mathsf{MGHE}}$. Then we let $\mathcal{A}_{dc}$ query $\mathcal{O}_{\mathsf{Dec}}$ with l. Note that $\mathcal{O}_{\mathsf{Dec}}$ does $d \leftarrow \mathsf{MGHE}.\texttt{PDec}(\mathsf{sk}, c)$ for where sk is the secret key of the client, which reflects the behavior of the client in these two worlds. Thus we write d as the output of $\mathcal{P}_{\mathsf{cli}}$.

– **On input** $\langle \texttt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c; \eta] \rangle$ **to a corrupted** $\mathcal{P}_{\mathsf{cli}}$: We similarly let $\mathcal{A}_{dc}$ identify the index l of c in the lookup table $\mathcal{T}$ of $\mathrm{G}^{\mathrm{DC}}_{\mathsf{MGHE}}$. Then we let $\mathcal{A}_{dc}$ query $\mathcal{O}_{\mathsf{Dec}}$ with l and also with η. Note that $\mathcal{O}_{\mathsf{Dec}}$ does $d \leftarrow \mathsf{MGHE}.\texttt{PDec}(\mathsf{sk}, c; \eta)$ for where sk is the secret key of the client, which reflects the behavior of the client in these two worlds. Thus we write d as the output of $\mathcal{P}_{\mathsf{cli}}$.

We observe that in $\mathbf{HYB1}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$, when an (honest) client $\mathcal{P}_{\mathsf{cli}}$ is activated to combine the partial decryptions output by clients in a group grp' for a message m that is intended for the group grp, where $\mathsf{grp}' \subseteq \mathsf{grp}$ and $|\mathsf{grp}' \cap \mathsf{grp}| > \frac{|\mathsf{grp}|}{t}$. The message m is output if at least $\frac{|\mathsf{grp}|}{t}$ partial decryptions are honestly obtained and output by clients in grp'.

It follows that the environment $\mathcal{Z}$ can only distinguish between these two worlds if the algorithm $\mathsf{MGHE}.\texttt{Combine}(\cdot)$ reconstructs a message different from m using the same set of partial decryptions from grp' in $\mathbf{REAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$. This corresponds to the winning condition of $\mathrm{G}^{\mathrm{DC}}_{\mathsf{MGHE}}$. Thus we let $\mathcal{A}_{dc}$ to output the same partial decryptions from grp' in the game $\mathrm{G}^{\mathrm{DC}}_{\mathsf{MGHE}}$.

$$\Delta_{\mathcal{Z}} \left(\mathbf{REAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}, \mathbf{HYB1}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}} \right) \leq \mathbf{Adv}^{\mathrm{DC}}_{\mathsf{MGHE}}(\mathcal{A}_{dc}). \tag{3.2}$$

□

Claim 2.2. *If* MGHE *is* IND-CPA$^{\mathrm{pD}}$ *secure, then* $\mathbf{HYB1}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$ *and* $\mathbf{HYB2}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$ *are computationally indistinguishable.*

Proof. We prove the claim by demonstrating that if an environment $\mathcal{Z}$ can distinguish between $\mathbf{HYB1}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$ and $\mathbf{HYB2}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}}, \mathcal{A}, \mathcal{Z}}$, then we can use $\mathcal{Z}$ to construct an adversary $\mathcal{A}_{cpa}$ to break the IND-CPA$^{\mathrm{pD}}$ security of MGHE. For this, we let $\mathcal{A}_{cpa}$ simulate the interaction between the environment $\mathcal{Z}$ and the two worlds. We let $\mathcal{A}_{cpa}$ follow a similar setup as in proof of Claim 2.1 but interacts with the game $\mathrm{G}^{\mathrm{IND\text{-}CPA^{pD}}}_{\mathsf{MGHE}}$ instead. Then $\mathcal{A}_{cpa}$ handles its oracle queries as follows to simulate the responses based on the messages that the environment $\mathcal{Z}$ delivers to each party.

- **On input $\langle$`Encrypt` : $[\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}_j, m]\rangle$ to an honest client $\mathcal{P}_{\mathsf{cli}}$**: We consider the following two cases for this input.
    1. (1) If the group grp_j is not corrupted (i.e., $|\mathsf{grp}_j \cap \mathsf{Corrupt}| < \frac{|\mathsf{grp}_j|}{t}$), we let $\mathcal{A}_{cpa}$ query its encryption oracle $\mathcal{O}_{\mathsf{Enc}}$ with the tuple $(\mathsf{jpk}_j, m_0, m_1)$, where $m_0 = m$ and $m_1 = 0^{|m|}$ for the m chosen by $\mathcal{Z}$. When $\mathcal{O}_{\mathsf{Enc}}$ responds with $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m)$, this simulates the behavior of an honest client in the world $\mathbf{HYB}1^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi\mathsf{MGHE},\mathcal{A},\mathcal{Z}}$, where m is encrypted, and the encryption is output by $\mathcal{P}_{\mathsf{cli}}$. On the other hand, if $\mathcal{O}_{\mathsf{Enc}}$ returns $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, 0^{|m|})$, it represents the scenario where the dummy party $\mathcal{P}_{\mathsf{cli}}$ outputs the encryption of $0^{|m|}$, which is returned from $\mathcal{F}_{\mathsf{MGHE}}$ and generated by $\mathcal{S}_{\mathsf{MGHE}}$. Accordingly, we write c returned by $\mathcal{O}_{\mathsf{Enc}}$ as the output of the client $\mathcal{P}_{\mathsf{cli}}$.
    2. (2) Otherwise, if grp_j is corrupted, we let $\mathcal{A}_{cpa}$ query its encryption oracle $\mathcal{O}_{\mathsf{Enc}}$ with the tuple $(\mathsf{jpk}_j, m; \omega)$, where ω is *honestly* sampled from the distribution since we are simulating the behavior of an *honest* client. Note that the oracle $\mathcal{O}_{\mathsf{Enc}^*}$ returns $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m; \omega)$. In $\mathbf{HYB}1^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, this is exactly the behavior of an honest client. Also, in $\mathbf{HYB}2^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, this is what the simulator $\mathcal{S}_{\mathsf{MGHE}}$ does when it is given the actual message m by $\mathcal{F}_{\mathsf{MGHE}}$. Thus, we then write c from $\mathcal{O}_{\mathsf{Enc}^*}$ as the output of the client $\mathcal{P}_{\mathsf{cli}}$.

- **On input $\langle$`Encrypt` : $[\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}_j, m; \omega]\rangle$ to a corrupted client $\mathcal{P}_{\mathsf{cli}}$**: We let $\mathcal{A}_{cpa}$ query its oracle with $(m; \omega)$, where ω is the random coins chosen by $\mathcal{Z}$. The oracle $\mathcal{O}_{\mathsf{Enc}^*}$ returns $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m; \omega)$. In $\mathbf{HYB}1^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, this input from $\mathcal{Z}$ is delivered to the adversary $\mathcal{A}$, who then computes $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m; \omega)$ using the random coins r provided by $\mathcal{Z}$, and writes c as the output of a corrupted client. Similarly, in $\mathbf{HYB}2^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, when the simulator $\mathcal{S}_{\mathsf{MGHE}}$ sends the input to $\mathcal{F}_{\mathsf{MGHE}}$, the functionality also performs the encryption honestly using the random coins chosen by $\mathcal{Z}$. The simulator $\mathcal{S}_{\mathsf{MGHE}}$ then writes c received from $\mathcal{F}_{\mathsf{MGHE}}$ as the output of the corrupted client. Accordingly, we write c from $\mathcal{O}_{\mathsf{Enc}^*}$ as the output of a corrupted client $\mathcal{P}_{\mathsf{cli}}$.

- **On input $\langle$`Evaluate` : $\big[\mathsf{sid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j\in[\ell]}; \sigma\big]\rangle$ to a corrupted server $\mathcal{P}_{\mathsf{srv}}$**: Note that by definition, all ciphertexts to be evaluated must either be encrypted by the clients or result from an evaluation performed by the server. This implies that in $\mathrm{G}^{\mathrm{IND\text{-}CPA^{pD}}}_{\mathsf{MGHE}}$, these ciphertexts are all entries in the lookup table $\mathcal{T}$. Thus, we let $\mathcal{A}_{cpa}$ first identify these ciphertexts in the lookup table $\mathcal{T}$ by specifying an index set L and query the oracle $\mathcal{O}_{\mathsf{Eval}}$ with the circuit f, the index set L, and the random coin σ. If $\perp$ is returned, $\mathcal{A}_{dc}$ aborts, and nothing is written on any tape. Otherwise, the oracle $\mathcal{O}_{\mathsf{Eval}}$ returns the ciphertext $\hat{c} \leftarrow \mathsf{MGHE}.\mathtt{Eval}(f, (\mathcal{T}[l].\mathsf{jpk}, \mathcal{T}[l].c)_{l\in L})$.

  In $\mathbf{HYB}1^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, since we consider a semi-malicious adversary, the adversary $\mathcal{A}$ evaluates the circuit f honestly to obtain the ciphertext $\hat{c}$. Similarly, in $\mathbf{HYB}2^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, by the definition of $\mathcal{F}_{\mathsf{MGHE}}$, when the server is corrupted, the functionality also evaluates the circuit honestly. After $\hat{c}$ is obtained, it is written as the output of a corrupted client. Thus, the behavior of these two executions is identical for this input and aligns with the behavior of $\mathcal{O}_{\mathsf{Eval}}$. Accordingly, we write $\hat{c}$ from $\mathcal{O}_{\mathsf{Eval}}$ as the output of the corrupted server $\mathcal{P}_{\mathsf{srv}}$.

- **On input $\langle$`Share` : $[\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c]\rangle$ to a honest $\mathcal{P}_{\mathsf{cli}}$**: We let $\mathcal{A}_{cpa}$ query it oracle $\mathcal{O}_{\mathsf{Dec}}$ with an index l identifying the position of c in the lookup table $\mathcal{T}$. If the oracle returns $\perp$, then $\mathcal{A}_{cpa}$ aborts the query. This indicates that the messages being decrypted in $\mathbf{HYB}1^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$

and $\mathsf{HYB2}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ differ. Consequently, after the execution, the environment $\mathcal{Z}$ can trivially distinguish between these two worlds by inspecting the final output tape of a client.

Otherwise, the oracle $\mathcal{O}_{\mathsf{Dec}}$ returns $\{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}}$ where $d_i \leftarrow \mathsf{MGHE}.\mathtt{PDec}(\mathsf{sk}_i, c)$ for $\mathsf{cli}_i \in \mathsf{grp}$. Note that in both of the worlds $\mathsf{HYB1}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{HYB2}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, the partial-decryptions output by clients are $\mathsf{MGHE}.\mathtt{PDec}(\cdot)$ in both two worlds. Therefore, we can then write the d_i from $\mathcal{O}_{\mathsf{Dec}}$ as the outputs of the clients.

– **On input** $\langle \mathtt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c; \eta] \rangle$ **to a corrupted** $\mathcal{P}_{\mathsf{cli}}$: We note that for corrupted client, the adversary $\mathcal{A}_{cpa}$ has the secret keys of the corrupted client. Thus we let $\mathcal{A}_{cpa}$ directly run $d \leftarrow \mathsf{MGHE}.\mathtt{PDec}(\mathsf{sk}_i, c; \eta)$. Then we write d as the output of the corrupted client.

– **On input** $\langle \mathtt{Combine} : [\mathsf{grp}, \mathsf{sid}, c, \{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}'}] \rangle$ **to a (honest or corrupted) client** $\mathcal{P}_{\mathsf{cli}}$: Note that in both $\mathsf{HYB1}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{HYB2}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, if insufficient partial decryption is obtained, then $\mathcal{P}_{\mathsf{cli}}$ outputs $\perp$. Otherwise, it outputs the original message m (plus an error term if approximate).

Observe that by decryption consistency, $m' \leftarrow \mathsf{MGHE}.\mathtt{Combine}(c, \{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}'})$ is indistinguishable from m. Thus, we write $\perp$ if not enough partial decryptions are obtained; otherwise, we write m' as the output of $\mathcal{P}_{\mathsf{cli}}$.

Finally, when $\mathcal{Z}$ outputs a bit b, we let $\mathcal{A}_{cpa}$ also output b. Thus $\mathcal{Z}$'s distinguishing advantage between these two worlds directly translates to $\mathcal{A}_{cpa}$'s advantage in the IND-CPA$^{\mathrm{pD}}$ game i.e.,

$$\Delta_{\mathcal{Z}}\left(\mathsf{HYB1}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}, \mathsf{HYB2}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}\right) \leq \mathbf{Adv}^{\text{IND-CPA}^{\mathrm{pD}}}_{\mathsf{MGHE}}(\mathcal{A}_{cpa}). \tag{3.3}$$

□

Claim 2.3. *If* MGHE *is* SIM-PDEC *secure with respect to a threshold simulator* $\mathtt{Sim}_{th}$*, then* $\mathsf{HYB2}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ *and* $\mathsf{IDEAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\mathcal{F}_{\mathsf{MGHE}},\mathcal{S}_{\mathsf{MGHE}},\mathcal{Z}}$ *are statistically indistinguishable.*

Proof. By the definition of SIM-PDEC, there exists a threshold simulator $\mathtt{Sim}_{th}$ such that a partial-decryption generated by an honest client using $d \leftarrow \mathsf{MGHE}.\mathtt{PDec}(\mathsf{sk}_i, c)$ is statistically indistinguishable from a share generated via $d \leftarrow \mathtt{Sim}_{th}(c, m, \{\mathsf{sk}_i\}_{i \in I_{\mathcal{A}}})$. For corrupted clients, the decryption process remains the same in both hybrid games since they are both computed as $d \leftarrow \mathsf{MGHE}.\mathtt{PDec}(\mathsf{sk}_i, c; \eta)$, or they directly injects a fake partial decryption. Additionally, for approximate schemes, we know that δ_{ct} and δ are indistinguishable by previous hop. Thus we have that the reconstruct messages in these two worlds are also indistinguishable from each other in case of approximate schemes..

Therefore, we have that the partial-decryptions output by any client $\mathcal{P}_{\mathsf{cli}}$ (whether honest or corrupted) in both of $\mathsf{HYB2}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\mathcal{F}_{\mathsf{MGHE}},\mathcal{S}_{\mathsf{MGHE}},\mathcal{Z}}$ are statistically indistinguishable. □

Therefore, by combing Claims 2.1 to 2.3, we can then conclude the proof for the theorem. □

In Theorem 3 and Lemma 5, we then move to the security with respect to a semi-malicious adversary who only corrupts a subset of the clients and the server remains honest. Thus we need to additionally consider a hybrid game for server-side confidentiality.

Theorem 3. *Let* MGHE *be a multi-group homomorphic encryption scheme on approximate number with threshold factor* t*. Then, the protocol* Π_{MGHE} *UC-realizes the functionality* $\mathcal{F}_{\mathsf{MGHE}}$ *in the presence of a global key registry* $\mathcal{G}_{\mathsf{KRK}}$ *against a partial-decryption manipulating semi-malicious adversary that non-adaptively corrupts parties, provided that:*

- MGHE *satisfies Decryption Consistency, and the adversary corrupts fewer than* $\frac{|\mathsf{grp}|}{t}$ *clients in any group.*
- MGHE *is* IND-CIRC *secure and the server is not corrupted.*
- MGHE *is* IND-CPA$^{\mathrm{pD}}$*-secure, and there exists at least one uncorrupted client who provides input to the computation, such that the group* grp *under which it provides input contains no more than* $\frac{|\mathsf{grp}|}{t}$ *corrupted clients.*
- MGHE *satisfies* SIM-PDEC *security with respect to a simulator* $\mathtt{Sim}_{th}$*.*

Proof. Note that the correctness of the protocol follows Theorem 2. We only show security in the lemma. We consider the same setting as in the proof of Theorem 2.To show the security of Π_{MGHE}, we consider the following sequence of hybrid games:

– $\mathbf{REAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$: This represents the execution of the protocol Π_{MGHE} in the real world, involving the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$.

– $\mathbf{HYB}^{1\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$: In this hybrid game, we keep track of the message-ciphertext tuples (m, c) that are encrypted and evaluated, as well as the partial decryptions d that are honestly generated with respect to (m, c). When a client combines the partial decryptions, if it has obtained a sufficient number of honest partial decryptions, it outputs $m+\delta_{\mathsf{ct}}+\sigma+\sum_{i=1}^{|\mathsf{grp}|/t}\eta_i$.

– $\mathbf{HYB}^{2\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$: In this hybrid game, when the server $\mathcal{P}_{\mathsf{srv}}$ is activated with the input $\langle\mathtt{Evaluate} : [\mathsf{sid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j\in[\ell]}]\rangle$, we let $\mathcal{P}_{\mathsf{srv}}$ output the ciphertext

$$\hat{c}^* \leftarrow \mathsf{MGHE}.\mathtt{Eval}(f^*, (\mathsf{jpk}_j, c_j)_{j\in[\ell]})$$

where f^* is a randomly sampled circuit from the circuit space $\mathfrak{F}_\kappa$. The noise δ_{ct} is then re-calculated with f^*.

– $\mathbf{HYB}^{3\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$: In this hybrid game, whenever an honest client $\mathcal{P}_{\mathsf{cli}}$ is activated with the input $\langle\mathtt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}, m]\rangle$, the client outputs $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}, 0^{|m|})$.

– $\mathbf{IDEAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\mathcal{F}_{\mathsf{MGHE}},\mathcal{S}_{\mathsf{MGHE}},\mathcal{Z}}$: This is the ideal execution where the ideal functionality $\mathcal{F}_{\mathsf{MGHE}}$ is attached to the simulator $\mathcal{S}_{\mathsf{MGHE}}$ and interacts with dummy client $\mathcal{P}_{\mathsf{cli}_1}, \dots, \mathcal{P}_{\mathsf{cli}_n}$.

Note that by Claim 2.1, we know that $\mathbf{REAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{HYB}^{1\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ are computational indistinguishable by the decryption consistency of MGHE. Now we claim that:

Claim 3.1. *If* MGHE *is* IND-CIRC *secure, then* $\mathbf{HYB}^{1\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ *and* $\mathbf{HYB}^{2\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ *are computationally indistinguishable.*

Proof. We show that if there exists an environment $\mathcal{Z}$ that can distinguish between $\mathbf{HYB}^{1\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{HYB}^{2\,\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, we can construct an IND-CIRC adversary $\mathcal{A}_{circ}$ against MGHE using this environment. To do so, we let $\mathcal{A}_{circ}$ simulate the interaction between $\mathcal{Z}$ and these two worlds. Specifically, $\mathcal{A}_{circ}$ forms the same groups by specifying the index set $I_1, \dots, I_k$ in the game $\mathrm{G}^{\text{IND-CIRC}}_{\mathsf{MGHE}}$ and corrupts the same clients by defining the index set $I_{\mathcal{A}}$ as in the setup phase. Then, $\mathcal{A}_{cpa}$ handles its oracle queries based on the input provided by the environment $\mathcal{Z}$ to each party.

- **On input** $\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}_j, m] \rangle$ **to an honest client** $\mathcal{P}_{\mathsf{cli}}$: We let $\mathcal{A}_{cpa}$ query its oracle $\mathcal{O}_{\mathsf{Enc}}$ with $(\mathsf{jpk}_j, m; \omega)$ where r is an honestly sampled random coins. Note that in both $\mathbf{HYB}^{1}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{HYB}^{2}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$. An honest client runs and outputs $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m; \omega)$ for an honestly sampled random coins ω, which is exactly the behavior of the oracle $\mathcal{O}_{\mathsf{Enc}}$. Thus we write c output from $\mathcal{O}_{\mathsf{Enc}}$ as the output of an honest client $\mathcal{P}_{\mathsf{cli}}$.

- **On input** $\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}_j, m; \omega] \rangle$ **to a corrupted client** $\mathcal{P}_{\mathsf{cli}}$: We let $\mathcal{A}_{cpa}$ query its oracle $\mathcal{O}_{\mathsf{Enc}}$ with $(\mathsf{jpk}_j, m; \omega)$, where ω is the random coins provided by $\mathcal{Z}$ to a corrupted client. In both hybrid games, since we consider a semi-malicious adversary, the ciphertext is generated as $c \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m; \omega)$ and is written as the output of a corrupted client. This behavior corresponds to that of the oracle $\mathcal{O}_{\mathsf{Enc}}$. Therefore, we write the ciphertext c output by $\mathcal{O}_{\mathsf{Enc}}$ as the output of a corrupted client $\mathcal{P}_{\mathsf{cli}}$.

- **On input** $\langle \texttt{Evaluate} : [\mathsf{sid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j\in[\ell]}] \rangle$ **to an honest server** $\mathcal{P}_{\mathsf{srv}}$: Similarly as in the proof of Theorem 2, we let $\mathcal{A}_{cpa}$ first identify these ciphertexts in the lookup table $\mathcal{T}$ of $\mathrm{G}^{\text{IND-CIRC}}_{\mathsf{MGHE}}$ by specifying an index set L. Then, we let $\mathcal{A}_{circ}$ queries the oracle $\mathcal{O}_{\mathsf{Eval}}$ with circuit-index set pair (f, L, f^*, L) where f^* is a circuit sampled uniformly at random from the circuit space $\mathcal{F}_\kappa$. If the oracle outputs $\perp$, we let $\mathcal{A}_{cpa}$ abort this query and nothing is written on any tape. This ensures that the output of a client $\mathcal{P}_{\mathsf{cli}}$ during decryption remains the same in both worlds, thereby preventing trivial distinguishing. Otherwise, the evaluated ciphertext output from $\mathcal{O}_{\mathsf{Eval}}$ is either

$$\hat{c} \leftarrow \mathsf{MGHE}.\mathtt{Eval}(f, (\mathcal{T}[l].\mathsf{jpk}, \mathcal{T}[l].c)_{l\in L})$$

or

$$\hat{c}^* \leftarrow \mathsf{MGHE}.\mathtt{Eval}(f^*, (\mathcal{T}[l].\mathsf{jpk}, \mathcal{T}[l].c)_{l\in L}).$$

Observe that in $\mathbf{HYB}^{1}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, an honest server $\mathcal{P}_{\mathsf{srv}}$ outputs $\hat{c}$, while in $\mathbf{HYB}^{2}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, $\hat{c}^*$ corresponds to the output of an honest server. Therefore, we write $\hat{c}$ or $\hat{c}^*$ returned by $\mathcal{O}_{\mathsf{Eval}}$ as the output of an honest server $\mathcal{P}_{\mathsf{srv}}$.

- **On input** $\langle \texttt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c] \rangle$ **to an honest client** $\mathcal{P}_{\mathsf{cli}}$: This is handled the same way as in the proof of Claim 2.2.

- **On input** $\langle \texttt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c; \eta] \rangle$ **to an corrupted client** $\mathcal{P}_{\mathsf{cli}}$: This is handled the same way as in the proof of Claim 2.2.

- **On input** $\langle \texttt{Combine} : [\mathsf{grp}, \mathsf{sid}, c, \{d_i\}_{\mathsf{cli}_i \in \mathsf{grp}'}] \rangle$ **to a client** $\mathcal{P}_{\mathsf{cli}}$: This is handled the same way as in the proof of Claim 2.2.

Finally, we let $\mathcal{A}_{circ}$ output the same bit output by $\mathcal{Z}$. Thus, the distinguishing advantage of $\mathcal{Z}$ translates directly to the advantage of $\mathcal{A}_{circ}$ in the IND-CIRC game:

$$\Delta_{\mathcal{Z}}\left(\mathbf{HYB}^{1}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}, \mathbf{HYB}^{2}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}\right) \leq \mathbf{Adv}^{\text{IND-CIRC}}_{\mathsf{MGHE}}(\mathcal{A}_{circ}). \tag{3.4}$$

□

Claim 3.2. *If* MGHE *is* IND-CPA$^{\mathrm{pD}}$ *secure, then* $\mathbf{HYB}^{2}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ *and* $\mathbf{HYB}^{3}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ *are computationally indistinguishable.*

Proof. We consider the same IND-CPA$^{\text{PD}}$ adversary $\mathcal{A}_{cpa}$ as defined in the proof of Theorem 3, but instead let $\mathcal{A}_{cpa}$ query its oracle $\mathcal{O}_{\mathsf{Eval}}$ with a circuit f^* sampled from the circuit space $\mathcal{F}_\kappa$ at random. Note that In $\mathbf{HYB}^{2\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, the ciphertext $\hat{c}$ is computed as

$$\hat{c} \leftarrow \mathsf{MGHE}.\mathtt{Eval}(f^*, \{\mathsf{jpk}\}_{j\in[\ell]}, f(\{m_j\}_{j\in[\ell]}))$$

for some circuit f, where each m_j is encrypted by either an honest or a corrupted client. In contrast, in $\mathbf{HYB}^{3\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, the ciphertext $\hat{c}$, the ciphertext $\hat{c}$ is computed as

$$\hat{c} \leftarrow \mathsf{MGHE}.\mathtt{Eval}(f^*, \{\mathsf{jpk}\}_{j\in[\ell]}, f(\{m_j\}_{j\in[\ell]}))$$

where each m'_j is either $0^{|m_j|}$ or some plaintext m_j if it is from an honest client or m_j if it is from a corrupted client or encrypted for a corrupted group. The remaining part follows the proof of Claim 2.2. ☐

Also, by Claim 2.3, we have that $\mathbf{HYB}^{3\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{IDEAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\mathcal{F}_{\mathsf{MGHE}},\mathcal{S}_{\mathsf{MGHE}},\mathcal{Z}}$ statistically indistinguishable by SIM-PDEC of MGHE. Therefore, by combining Claims 3.1 and 3.2 and Claims 2.1 and 2.3, we can conclude the proof for this theorem. ☐

We can then consider a stronger security by requiring the statistical indistinguishability of the circuit for schemes with exact evaluation. In Lemma 5, we show the server-side confidentiality with the SIM-CIRC security.

Lemma 5. *Let* MGHE *be a multi-group homomorphic encryption scheme on exact number with threshold factor t. Then, the protocol* Π_{MGHE} *UC-realizes the functionality* $\mathcal{F}_{\mathsf{MGHE}}$ *in the presence of a global key registry* $\mathcal{G}_{\mathsf{KRK}}$ *against a partial-decryption manipulating semi-malicious adversary that non-adaptively corrupts parties, provided that:*

- MGHE *satisfies Decryption Consistency, and the adversary corrupts fewer than* $\frac{|\mathsf{grp}|}{t}$ *clients in any group.*
- MGHE *is* SIM-CIRC *secure and the server is not corrupted.*
- MGHE *is* IND-CPA$^{\text{PD}}$*-secure, and there exists at least one uncorrupted client who provides input to the computation, such that the group* grp *under which it provides input contains no more than* $\frac{|\mathsf{grp}|}{t}$ *corrupted clients.*
- MGHE *satisfies* SIM-PDEC *security with respect to a simulator* $\mathtt{Sim}_{th}$.

Proof. We consider the same setting as in the proof of Theorem 2. From Theorem 3, we only replace the last two games. For completeness, we also all of the hybrid worlds here.

– $\mathbf{REAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$: This represents the execution of the protocol Π_{MGHE} in the real world, involving the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$.

– $\mathbf{HYB}^{1\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$: In this hybrid game, we keep track of the message-ciphertext tuples (m, c) that are encrypted and evaluated, as well as the partial decryptions d that are honestly generated with respect to (m, c). When a client combines the partial decryptions, if it has obtained a sufficient number of honest partial decryptions, it outputs m.

– $\mathbf{HYB}^{2\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$: In this hybrid game, whenever a server $\mathcal{P}_{\mathsf{srv}}$ is activated with the input $\langle \mathtt{Evaluate} : [\mathsf{sid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j\in[\ell]}] \rangle$, we let $\mathcal{P}_{\mathsf{srv}}$ output the ciphertext $\hat{c} \leftarrow \mathtt{Sim}_{circ}(\{\mathsf{jpk}_j\}_{j\in[\ell]}, \hat{m})$ produced by the circuit privacy simulator $\mathtt{Sim}_{circ}$ where $\hat{m} = f(m_1, \ldots, m_j)$ and $c_j \leftarrow \mathsf{MGHE}.\mathtt{Enc}(\mathsf{jpk}_j, m_j)$ for $j \in [\ell]$.

– $\mathsf{HYB}^{3}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$: In this hybrid game, whenever an honest client $\mathcal{P}_{\mathsf{cli}}$ is activated with the input $\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}, m]\rangle$, the client outputs $c \leftarrow \mathsf{MGHE}.\texttt{Enc}(\mathsf{jpk}, 0^{|m|})$.

– $\mathsf{IDEAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\mathcal{F}_{\mathsf{MGHE}},\mathcal{S}_{\mathsf{MGHE}},\mathcal{Z}}$: This is the ideal execution where the ideal functionality $\mathcal{F}_{\mathsf{MGHE}}$ is attached to the simulator $\mathcal{S}_{\mathsf{MGHE}}$ and interacts with dummy client $\mathcal{P}_{\mathsf{cli}_1}, \ldots, \mathcal{P}_{\mathsf{cli}_n}$.

Similarly, by Claim 2.1, we know that $\mathsf{REAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{HYB}^{1}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ are computational indistinguishable by the decryption consistency of MGHE. Now we claim that:

Claim 3.3. *If* MGHE *is* SIM-CIRC *secure with respect to a circuit-privacy simulator* $\texttt{Sim}_{circ}$, *then* $\mathsf{HYB}^{1}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ *and* $\mathsf{HYB}^{2}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ *are statistically indistinguishable.*

Proof. By the definition of SIM-CIRC, there exists a circuit-privacy simulator $\texttt{Sim}_{circ}$ such that an evaluated ciphertext output by $\hat{c} \leftarrow \mathsf{MGHE}.\texttt{Eval}(f, (jpk_j, c_j)_{j\in[\ell]})$ is statistically indistinguishable from $c \leftarrow \texttt{Sim}_{circ}(\{jpk_j\}_{j\in[\ell]}, \hat{m}$ where $\hat{m} = f(m_1, \ldots, m_\ell)$ and $c_j \leftarrow \mathsf{MGHE}.\texttt{Enc}(\mathsf{jpk}_j, m_j)$ for $j \in [\ell]$. Observe that these two outputs correspond to the outputs of an honest server when it is activated to evaluated a circuit f in these two worlds. Thus we have that the distributions of the two worlds are statistically indistinguishable. □

Claim 3.4. *If* MGHE *is* IND-CPA$^{\mathrm{pD}}$ *secure, then* $\mathsf{HYB}^{3}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ *and* $\mathsf{IDEAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\mathcal{F}_{\mathsf{MGHE}},\mathcal{S}_{\mathsf{MGHE}},\mathcal{Z}}$ *are computationally indistinguishable.*

Proof. We consider the same IND-CPA$^{\mathrm{pD}}$ adversary $\mathcal{A}_{cpa}$ as defined in the proof of Theorem 3, but with a change in how $\mathcal{A}_{cpa}$ handles queries to $\mathcal{O}_{\mathsf{Eval}}$. Note that In $\mathsf{HYB}^{2}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, the ciphertext $\hat{c}$ is computed as

$$\hat{c} \leftarrow \texttt{Sim}_{circ}(\{\mathsf{jpk}\}_{j\in[\ell]}, f(\{m_j\}_{j\in[\ell]}))$$

for some circuit f, where each m_j is encrypted by either an honest or a corrupted client. In contrast, in $\mathsf{HYB}^{3}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$, the ciphertext $\hat{c}$, the ciphertext $\hat{c}$ is computed as

$$\hat{c} \leftarrow \texttt{Sim}_{circ}(\{\mathsf{jpk}\}_{j\in[\ell]}, f(\{m'_j\}_{j\in[\ell]}))$$

where each m'_j is either $0^{|m_j|}$ or some plaintext m_j if it is from an honest client or m_j if it is from a corrupted client or encrypted for a corrupted group.

Furthermore, by Claim 3.3, we know that the ciphertext $\hat{c}$ obtained from $\mathsf{MGHE}.\texttt{Eval}(\cdot)$ and $\texttt{Sim}_{circ}(\cdot)$ are statistically indistinguishable. Thus, the output of $\mathcal{P}_{\mathsf{srv}}$ in these two worlds corresponds to the behavior of $\mathcal{O}_{\mathsf{Eval}}$ when $b = 0$ or $b = 1$ in the IND-CPA$^{\mathrm{pD}}$ game, respectively. Upon receiving $\hat{c}$ from $\mathcal{O}_{\mathsf{Eval}}$, we can then write it as the output of $\mathcal{P}_{\mathsf{srv}}$. The remaining part follows the proof of Claim 2.2. □

Also, by Claim 2.3, we have that $\mathsf{HYB}^{3}{}^{\mathcal{G}_{\mathsf{KRK}}}_{\Pi_{\mathsf{MGHE}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}^{\mathcal{G}_{\mathsf{KRK}}}_{\mathcal{F}_{\mathsf{MGHE}},\mathcal{S}_{\mathsf{MGHE}},\mathcal{Z}}$ statistically indistinguishable by SIM-PDEC of MGHE. Therefore, by combing Claims 3.3 and 3.4, and Claims 2.1 and 2.3, we can then conclude the proof for this lemma. □

Remark 13 (Security against semi-honest adversary)**.** It is worth noting that MGHE does not need to (nor can) satisfy SIM-CIRC security when considering a semi-honest real-world adversary $\mathcal{A}$ who only observes network traffic. In this case, the adversary does not have knowledge of the evaluated plaintext $\hat{m}$. When $\mathcal{F}_{\mathsf{MGHE}}$ forwards only the length $|\hat{m}|$ to $\mathcal{S}_{\mathsf{MGHE}}$, it executes $\mathsf{MGHE}.\texttt{Eval}(\cdot)$ using a circuit sampled uniformly at random from the circuit space. While $\mathcal{F}_{\mathsf{MGHE}}$ could include $\hat{m}$ as a trapdoor for $\mathcal{S}_{\mathsf{MGHE}}$, this is unnecessary, as the real-world adversary $\mathcal{A}$ does not know $\hat{m}$. Consequently, as in the proof of Theorem 3, we can bridge two hybrid games using an IND-CIRC adversary.

Remark 14 (Necessity of IND-CPA$^{\text{pD}}$ plus SIM-PDEC)**.** We acknowledge that the combination of SIM-PDEC with IND-CPA security has also been considered in previous works [MW16, KLSW24], where the game hopping for SIM-PDEC occurs before that of IND-CPA. However, we emphasize that IND-CPA together with SIM-PDEC does not necessarily imply IND-CPA$^{\text{pD}}$. Indeed, the noise-smudging technique used to ensure SIM-PDEC is also a primary technique in mitigating IND-CPA$^{\text{D}}$ attacks. Yet we note that the attack on bootstrapping failure presented by Cheon et al. [CCP+24] may not be captured under this combination, as their attack remains effective even against CKKS with the noise-smudging technique. Furthermore, Cheon et al. [CCP+24] extended their attack to the threshold setting, leveraging decryption failures that arise after partial decryptions are combined. This suggests that an adversary might still be able to exploit the reconstructed message for such an attack while the partial decryptions are simulatable. We remark that this may be subject to a case-by-case study. Here we to choose to consider both security notions to cover a broader range of scenarios.

We also note that IND-CPA$^{\text{pD}}$ alone does not guarantee UC-security, as the simulator $\mathcal{S}$ (without the knowledge of secret keys) cannot generate simulated partial decryptions for honest clients if MGHE does not satisfies SIM-PDEC.

However, achieving IND-CPA$^{\text{pD}}$ alone still makes a sense. Boudgoust and Scholl [BS23] argued that achieving IND-CPA$^{\text{pD}}$ security remains meaningful. They pointed out that a major drawback of using the noise-flooding (smudging) technique to ensure partial-decryption simulatability is that it requires the ratio between the flooding noise and the ciphertext noise to be *superpolynomial* in the security parameter, leading to significant overhead in ciphertext size. In contrast, achieving IND-CPA$^{\text{pD}}$ security allows for a polynomial noise size at the cost of slightly weaker security.

Chapter 4

Integrity Layer

In this chapter, we examine the integrity layer of the system, focusing on achieving integrity through *verifiability* by leveraging *zero-knowledge succinct arguments of knowledge* (zkSNARKs). In Section 4.1, we revisit the concept of *non-interactive arguments* (NARGs) in the *random oracle model* (ROM), highlighting key properties such as knowledge soundness and zero-knowledge. Subsequently, in Section 4.3, we define the functionality of a zkSNARK and demonstrate its UC-realization with a global random oracle that is both observable and programmable.

4.1 Non-Interactive Argument (NARG)

Non-interactive arguments are typically instantiated in the *Common Reference String Model* (CRS) or the *Random Oracle Model* (ROM). Several other constructions [LR22a, LR22b, GKO^{+}23] also operate in a model that combines a common reference string with a random oracle. As noted by Ganesh et al. [GKO^{+}23], certain properties necessary for achieving UC-security can only be realized under the assumption of a global random oracle. Additionally, Chiesa and Fenzi [CF24] demonstrated that several well-known zkSNARKs [Mic00, BCS16] are UC-secure in the *plain* random oracle model. Thus, in this thesis, we focus on analyzing security in the (plain) random oracle model for generality.

4.1.1 Random Oracle Model (ROM)

Random Oracle Model (ROM) is a heuristic model, firstly used by Bellare and Rogway [BR93] for cryptographic proof, to simulate an idealized hash function. In a *Random Oracle* (RO), every input is mapped to a random output, independently of any other inputs. This oracle is accessible to all parties, both honest and malicious. In Definition 26, we adopt the definition for a random oracle provided by Chiesa and Yogev [CY24].

Definition 26 (Random Oracle [CY24])**.** Let $\lambda \in \mathbb{N}$, we let $\mathcal{F}_\lambda$ denote the set of all functions of the form $f : \{0,1\}^* \to \{0,1\}^\lambda$. If $\mathcal{O}_{\mathsf{RO}}$ is sampled from uniformly at random $\mathcal{F}_\lambda$, then for every input x, the output $y := \mathcal{O}_{\mathsf{RO}}(x)$ is a uniformly random λ-bit string, where the output for each input is sampled independently. We refer to $\mathcal{O}_{\mathsf{RO}} \leftarrow\!\!\$\ \mathcal{F}_\lambda$ as a *random oracle* with output size λ.

For certain security notions, we need to give the adversary the "superpower" to program the random oracle. Specifically, if a query-answer pair (x, v) is specified and x has not been queried before, then any subsequent queries to x will consistently yield v as the response from the random oracle.

Definition 27 (Programmable Random Oracle)**.** A *programmable random oracle* $\mathcal{O}_{\mathsf{pRO}}$ is a random oracle that can be programmed with a set of query-answer pairs pg. For every $(x, v) \in \mathsf{pg}$, if x has not yet been queried to $\mathcal{O}_{\mathsf{pRO}}$, then the oracle defines v as the output for later queries on x. We denote $v := \mathcal{O}_{\mathsf{pRO}}[\mathsf{pg}](x)$ as that a programmable random oracle $\mathcal{O}_{\mathsf{pRO}}$ programmed with a set of query-answer pairs pg returns v upon receiving the query x. We use the notation $b := \mathcal{O}_{\mathsf{RO}}[(x, v)](\cdot)$ to represent the attempt to program the oracle with (x, v). If the programming succeeds, $b = 1$ is returned; otherwise, $b = 0$ is returned.

4.1.2 Syntax of NARG

In this section, we revisit the syntax of NARG in random oracle model (ROM). In Definition 28, we define a non-interactive argument following the work of Chiesa and Yogev [CY24].

Definition 28 (Non-Interactive Argument [CY24])**.** Let $\mathcal{O}_{\mathsf{RO}}$ be a random oracle with output length λ. A *non-interactive argument* is a tuple of algorithms, denoted as $\mathsf{NARG} = (\mathtt{Prv}, \mathtt{Vfy})$, where

– $\pi \leftarrow \mathtt{Prv}^{\mathcal{O}_{\mathsf{RO}}}(x, w)$: Given an instance x, a witness w, and access to a random oracle $\mathcal{O}_{\mathsf{RO}}$, the *argument prover* algorithm outputs an argument string π.

– $b \leftarrow \mathtt{Vfy}^{\mathcal{O}_{\mathsf{RO}}}(x, \pi)$: Given an instance x, an argument string π, and access to the random oracle $\mathcal{O}_{\mathsf{RO}}$, outputs a bit $b \in \{0, 1\}$, where $b = 1$ denotes acceptance and $b = 0$ denotes rejection.

To be efficient, a NARG must also satisfy *succinctness*.[1] Observe that setting the argument π as the witness w yields a trivial NARG. Thus to be succinct, it means that the argument size of a NARG should be smaller than the witness size.

Definition 29 (Succinctness)**.** A non-interactive argument $\mathsf{NARG} = (\mathtt{Prv}, \mathtt{Vfy})$ is *succinct* if for any output size $\lambda \in \mathbb{N}$ of the random oracle $\mathcal{O}_{\mathsf{RO}}$, any instance-witness pair $(x, w) \in R$ and $\pi \leftarrow \mathtt{Prv}^{\mathcal{O}_{\mathsf{RO}}}(x, w)$, then size of π is given by

$$|\pi| = \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(|x| + |w|).$$

Definition 30 (SNARG)**.** A non-interactive argument $\mathsf{NARG} = (\mathtt{Prv}, \mathtt{Vfy})$ is a *succinct non-interactive argument* (SNARG) if it also satisfies succinctness.

4.2 Security Notions for Verifiability

In this section, we revisit the security notions for NARG. Specifically, we consider completeness, knowledge soundness, and zero-knowledge. We first revisit the basic definitions of these notions following the work by Chiesa and Yogev [CY24]. We then discuss the limitations of these basic definitions for UC security and illustrate the UC-friendly security notions proposed by Chiesa and Fenzi [CF24].

4.2.1 Completeness

For a NARG to be "correct", it needs to satisfy (*perfect*) *completeness*. That means, if an instance-argument (x, π) is generated by an honest prover, then it is always accepted by an honest verifier. In Definition 32, we define the completeness of an NARG.

[1]Chiesa and Yogev[CY24] also discussed succinctness in verification. Since it is less related to the discussion in this thesis, we omit that part.

Definition 31 (Perfect Completeness [CY24])**.** A non-interactive argument $\mathsf{NARG} = (\texttt{Prv}, \texttt{Vfy})$ for a relation R has *perfect completeness* if for any output size $\lambda \in \mathbb{N}$ of the random oracle $\mathcal{O}_{\mathsf{RO}}$ and any instance-witness pair $(x, w) \in R$, it has that

$$\Pr\left[\texttt{Vfy}^{\mathcal{O}_{\mathsf{RO}}}(x,\pi) = 1 \;\middle|\; \begin{array}{l} \mathcal{O}_{\mathsf{RO}} \leftarrow\!\!\$\; \mathcal{F}_\lambda \\ \pi \leftarrow \texttt{Prv}^{\mathcal{O}_{\mathsf{RO}}}(x,w) \end{array}\right] = 1.$$

The probability is taken over $\mathcal{O}_{\mathsf{RO}}$ and any random coins of the argument prover $\texttt{Prv}$ and the argument verifier $\texttt{Vfy}$.

However, as also indicated by Chiesa and Fenzi [CF24], enabling perfect completeness does not necessarily mean a NARG is complete in the UC framework. That is because, the environment $\mathcal{Z}$ is allowed to program the RO (at any time), which may yield trivial incompleteness. Thus, they introduced a UC-friendly completeness which allows the adversary to program the RO.

```
Game: $\mathrm{G}^{\mathrm{Comp}}_{\mathsf{NARG}}$

Procedure Init
1 : $\mathcal{O}_{\mathsf{RO}} \leftarrow\!\!\$\; \mathcal{F}_\lambda$
2 : $\mathsf{pg}_a := [\,]$
3 : $Q := [\,]$

Oracle $\mathcal{O}_{\mathsf{Prv}}(x, w)$
1 : if $(x, w) \notin R \vee |x| > n$ then
2 :     return $\perp$
3 : $\pi \leftarrow \texttt{Prv}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}_a]}(x, w)$
4 : $Q := Q \cup \{(x, \pi)\}$
5 : return $\pi$

Procedure Fin
1 : return $\mathsf{win}$

Oracle $\mathcal{O}_{\mathsf{Prog}}(x, v)$
1 : $b := \mathcal{O}_{\mathsf{RO}}[(x, v)](\cdot)$
2 : if $b = 1$ then
3 :     $\mathsf{pg}_a := \mathsf{pg}_a \bowtie (x, v)$
4 : return $b$

Oracle $\mathcal{O}_{\mathsf{Vfy}}(x, \pi)$
1 : if $|x| > n$ then
2 :     return $\perp$
3 : $b \xleftarrow{\mathsf{tr}_v} \texttt{Vfy}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}_a]}(x, \pi)$
4 : if $(x, \pi) \in Q \wedge b = 0$
5 :  $\wedge\, \mathsf{tr}_v \cap \mathsf{pg}_a = \varnothing$ then
6 :     $\mathsf{win} := 1$
7 : return $b'$
```

Figure 4.1: Completeness (Comp) game for a non-interactive argument NARG introduced by Chiesa and Fenzi [CF24].

In Figure 4.1, we illustrate the UC-friendly completeness introduced by Chiesa and Fenzi [CF24]. The adversary $\mathcal{A}$ can *adaptively* program $\mathcal{O}_{\mathsf{RO}}$ by querying the oracle $\mathcal{O}_{\mathsf{Prog}}$.[2] When the adversary queries the oracle $\mathcal{O}_{\mathsf{Prv}}$ with an instance-witness pair (x, w), the respective instance-argument pair (x, π) will be added to the lookup table Q. When the adversary queries the oracle $\mathcal{O}_{\mathsf{Vfy}}$ with (x, π), the verifier algorithm $\texttt{Vfy}$ is invoked to output a bit b for acceptance or rejection. The winning condition is defined as $b = 0$ yet $(x, \pi) \in Q$, and the query trace made by $\texttt{Vfy}$ does not intersect with the points programmed by the adversary i.e., $tr_v \cap \mathsf{pg}_a =$. The last condition is to prevent the adversary from trivially winning the game.

Definition 32 (Completeness)**.** A non-interactive argument $\mathsf{NARG} = (\texttt{Prv}, \texttt{Vfy})$ for a relation R has *completeness* error ε in *explicitly-programmable random oracle model* (EPROM) if for

[2]The oracle $\mathcal{O}_{\mathsf{Prog}}$ is simply used to track the points programmed by $\mathcal{A}$.

any output size $\lambda \in \mathbb{N}$ of the random oracle $\mathcal{O}_{\mathsf{RO}}$, any instance bound $n \in \mathbb{N}$, any admissible adversary $\mathcal{A}$ that makes at most $q_{ro} \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{RO}}$, $q_{pg} \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{Prog}}$, $q_p \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{Prv}}$, and $q_v \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{Vfy}}$, it has that $\mathbf{Adv}^{\mathrm{Comp}}_{\mathsf{NARG}}(\mathcal{A}) \leq \varepsilon(\lambda, q_{ro}, q_{pg}, q_p, q_v, n)$ where

$$\mathbf{Adv}^{\mathrm{Comp}}_{\mathsf{NARG}}(\mathcal{A}) \coloneqq \Pr\left[\mathrm{G}^{\mathrm{Comp}}_{\mathsf{NARG}}(\mathcal{A}) \Rightarrow 1\right]$$

where $\mathcal{A}$ is admissible if it always outputs $(x, w) \in R$ and $|x| \leq n$.

Remark 15. We remark that, Chiesa and Fenzi [CF24, Lemma 5.9] showed that satisfying perfect completeness, *monotone proofs*, and *unpredictable queries*, which are generally satisfied by zero-knowledge NARG.

4.2.2 Zero-Knowledge

The privacy for an (honest) argument prover is formalized with the notion of *Zero-Knowledge* (ZK). Specifically, for an instance-argument pair (x, π), zero-knowledge guarantees that an adversary gains no knowledge about the witness w through this instance-argument pair.

The security of zero-knowledge is formalized using a "real-world vs. ideal-world" approach. In the real world, the argument π is produced by the argument prover $\mathtt{Prv}$ with inputs (x, w). In the ideal world, π is generated by a simulator $\mathtt{Sim}$ with only the instance x as input. If these two worlds are *statistically* indistinguishable, it means that an argument π reveals no information about the witness w.

Definition 33 ((Adaptive) Zero-Knowledge [CY24])**.** A non-interactive argument $\mathsf{NARG} = (\mathtt{Prv}, \mathtt{Vfy})$ for a relation R has *(adaptive) zero-knowledge* error ε in *explicitly-programmable random oracle model* (EPROM) if there exists a probabilistic polynomial-time simulator $\mathtt{Sim}$ such that, for any output size $\lambda \in \mathbb{N}$ of the random oracle $\mathcal{O}_{\mathsf{RO}}$, any instance bound $n \in \mathbb{N}$, any admissible adversary $\mathcal{A}$ that makes at most q queries to $\mathcal{O}_{\mathsf{RO}}$, the following two distribution are $\varepsilon(\lambda, q, n)$-close in statistical distance:

$$\mathcal{D}_{\mathrm{real}} \coloneqq \left\{ \mathsf{out} \;\middle|\; \begin{array}{l} \mathcal{O}_{\mathsf{RO}} \leftarrow\!\!\$\; \mathcal{F}_\lambda \\ (x, w, \mathsf{aux}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{RO}}} \\ \pi \leftarrow \mathtt{Prv}^{\mathcal{O}_{\mathsf{RO}}}(x, w) \\ \mathsf{out} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{RO}}}(\mathsf{aux}, \pi) \end{array} \right\} \text{ and } \mathcal{D}_{\mathrm{sim}} \coloneqq \left\{ \mathsf{out} \;\middle|\; \begin{array}{l} \mathcal{O}_{\mathsf{RO}} \leftarrow\!\!\$\; \mathcal{F}_\lambda \\ (x, w, \mathsf{aux}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{RO}}} \\ (\pi, \mathsf{pg}) \leftarrow \mathtt{Sim}^{\mathcal{O}_{\mathsf{RO}}}(x) \\ \mathsf{out} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}]}(\mathsf{aux}, \pi) \end{array} \right\}$$

where $\mathcal{A}$ is admissible if it always outputs $(x, w) \in R$ and $|x| \leq n$.

Observe that in Definition 35, the adversary is not allowed to program the RO. Thus in Figure 4.2, the adversary $\mathcal{A}$ is similarly given the access to $\mathcal{O}_{\mathsf{Prog}}$ to program RO. Additionally, any instance-argument pair generated by querying $\mathcal{O}_{\mathsf{Prv}}$ is accepted at the oracle $\mathcal{O}_{\mathsf{Vfy}}$. This covers a presumption of completeness, which we will later use in the security proof.

Definition 34 (Zero-Knowledge)**.** A non-interactive argument $\mathsf{NARG} = (\mathtt{Prv}, \mathtt{Vfy})$ for a relation R has *(adaptive) zero-knowledge* error ε in *explicitly-programmable random oracle model* (EPROM)[3] if there exists a probabilistic polynomial-time simulator $\mathtt{Sim}$ such that, for any output size $\lambda \in \mathbb{N}$ of the random oracle $\mathcal{O}_{\mathsf{RO}}$, any instance bound $n \in \mathbb{N}$, any admissible adversary $\mathcal{A}$ that makes at most $q_{ro} \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{RO}}$, $q_{pg} \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{Prog}}$, $q_p \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{Prv}}$, and $q_v \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{Vfy}}$, it has that $\mathbf{Adv}^{\mathrm{ZK}}_{\mathsf{NARG}}(\mathcal{A}) \leq \varepsilon(\lambda, q_{ro}, q_{pg}, q_p, q_v, n)$ where

$$\mathbf{Adv}^{\mathrm{ZK}}_{\mathsf{NARG}}(\mathcal{A}) \coloneqq \left|\Pr\left[\mathrm{G}^{\mathrm{ZK\text{-}0}}_{\mathsf{NARG}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathrm{G}^{\mathrm{ZK\text{-}1}}_{\mathsf{NARG}}(\mathcal{A}) \Rightarrow 1\right]\right|.$$

[3]We note that this contrasts with the *fully-programmable random oracle model* (FPROM), where the simulator internally implements the RO and can adaptively determine responses to oracle queries.

```
Game: $\mathrm{G}^{\text{ZK-0}}_{\mathsf{NARG}}$ $\boxed{\mathrm{G}^{\text{ZK-1}}_{\mathsf{NARG}}}$

Procedure INIT
1 : $\mathcal{O}_{\mathsf{RO}} \leftarrow\!\!\$\ \mathcal{F}_\lambda$
2 : $\mathsf{pg}_a \coloneqq [\,]$
3 : $\mathsf{pg}_{all} \coloneqq [\,]$
4 : $Q \coloneqq [\,]$

Oracle $\mathcal{O}_{\mathsf{Prv}}(x, w)$
1 : if $(x, w) \notin R \vee |x| > n$ then
2 :     return $\perp$
3 : $\pi \leftarrow \mathtt{Prv}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}_{all}]}(x, w)$
4 : $\boxed{\begin{array}{l} 1: (\pi, \mathsf{pg}_s) \leftarrow \mathtt{Sim}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}_{all}]}(x) \\ 2: \mathsf{pg}_{all} \coloneqq \mathsf{pg}_{all} \bowtie \mathsf{pg}_s \end{array}}$
5 : $Q \coloneqq Q \cup \{(x, \pi)\}$
6 : return $\pi$

Oracle $\mathcal{O}_{\mathsf{Prog}}(x, v)$
1 : $b \coloneqq \mathcal{O}_{\mathsf{RO}}[(x, v)](\cdot)$
2 : if $b = 1$ then
3 :     $\mathsf{pg}_a \coloneqq \mathsf{pg}_a \bowtie (x, v)$
4 :     $\mathsf{pg}_{all} \coloneqq \mathsf{pg}_{all} \bowtie \mathsf{pg}_a$
5 : return $b$

Oracle $\mathcal{O}_{\mathsf{Vfy}}(x, \pi)$
1 : if $|x| > n$ then
2 :     return $\perp$
3 : if $(x, \pi) \in Q$ then
4 :     return $1$
5 : $b \xleftarrow{\mathsf{tr}_v} \mathtt{Vfy}^{\mathcal{O}_{\mathsf{RO}}}(x, \pi)$
6 : return $(b \wedge \mathsf{tr}_v \cap \mathsf{pg}_a = \varnothing)$
```

Figure 4.2: Zero-Knowledge (ZK) game for a non-interactive argument NARG. We present a simplified version modified from the one introduced by Chiesa and Fenzi [CF24]. The dot-boxed part is exclusive for $\mathrm{G}^{\text{ZK-1}}_{\mathsf{NARG}}$.

where $\mathcal{A}$ is admissible if it always outputs $(x, w) \in R$ and $|x| \leq n$.

Remark 16 (Necessity of RO programmability)**.** In this thesis, we let the RO be programmable. Specifically, in Definitions 33 and 38, we allow the simulator `Sim` to output a query-answer list pg, enabling it to program the RO for the generation of a simulated proof. As noted by Chiesa and Yogev [CY24, Lemma 6.6.1], this configuration results in a weaker but still meaningful security definition. Intuitively, without the ability to program the RO, the simulator gains no structural advantage over a malicious prover. Consequently, if a NARG is zero-knowledge with respect to a simulator in a *non-programming* ROM, that simulator could then be used to decide the language. Since the simulator is intended to be efficient, this implies that the language would be trivial, and its zero-knowledge proof would lack practical significance.

4.2.3 Soundness

Another property that a NARG should satisfy is *soundness*. This property means that an instance x that is not in the language of the binary R should not be accepted. In Definition 35. we define the soundness of a NARG.

Definition 35 ((Adaptive) Soundness [CY24])**.** A non-interactive argument $\mathsf{NARG} = (\mathtt{Prv}, \mathtt{Vfy})$ for a relation R has *(adaptive) soundness error* ε if for any output size $\lambda \in \mathbb{N}$ of the random oracle $\mathcal{O}_{\mathsf{RO}}$, any instance size bound $n \in \mathbb{N}$, any malicious argument prover $\mathcal{A}$ that makes at most $q \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{RO}}$, it has

$$\Pr\left[\begin{array}{l} |x| \leq n \\ \wedge\, x \notin \mathcal{L}(R) \\ \wedge\, \mathtt{Vfy}^{\mathcal{O}_{\mathsf{RO}}}(x, \pi) = 1 \end{array} \middle| \begin{array}{l} \mathcal{O}_{\mathsf{RO}} \leftarrow\!\!\$\ \mathcal{F}_\lambda \\ (x, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{RO}}} \end{array}\right] \leq \varepsilon(\lambda, q, n).$$

The probability is taken over $\mathcal{O}_{\mathsf{RO}}$ and any random coins of the argument verifier `Vfy`.

Knowledge Soundness. *Knowledge Soundness* (KS) is a stronger security notion than Definition 35, requiring that a prover should be able to provide a witness w for any instance-argument pair (x, π) that has been verified as valid. KS is defined via the introduction of an *extractor*. Depending on the extractor's behavior, two types are KS are defined: *Straightline* KS and *Rewinding* KS. In straightline KS, the extractor must retrieve the witness in a *single* execution of the argument prover `Prv`. Rewinding KS, however, relaxes this constraint by allowing the extractor to run the prover multiple times. In both types, we assume the extractor's access to the argument prover is *black-box* i.e., it cannot access internal details, such as code and random coins, used by the prover.

In this thesis, we focus on *straightline* knowledge soundness. As indicated by Ganesh et al. [GKO+23], in the UC framework, the environment $\mathcal{Z}$ acts as an interactive distinguisher between the real execution protocol and the ideal process. Thus the extractor cannot rewind the environment $\mathcal{Z}$ for this purpose. On the top of this, black-box access is also required since the extractor should not have access to the internal code of $\mathcal{Z}$.

Definition 36 (Straightline Knowledge Soundness [CY24])**.** A non-interactive argument $\mathsf{NARG} = (\mathtt{Prv}, \mathtt{Vfy})$ for a relation R has *straight knowledge soundness* error ε if there is a polynomial-time determinisitc algorithm `Ext` (the extractor) such that for any output size $\lambda \in \mathbb{N}$ of the random oracle $\mathcal{O}_{\mathsf{RO}}$, any instance size bound $n \in N$, any deterministic adversary $\mathcal{A}$ that makes at most $q \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{RO}}$, it has

$$\Pr\left[\begin{array}{l|l} |x| \le n & \mathcal{O}_{\mathsf{RO}} \leftarrow\$ \mathcal{F}_\lambda \\ \wedge\, (x, w) \notin R & (x, \pi) \xleftarrow{\mathsf{tr}} \mathcal{A}^{\mathcal{O}_{\mathsf{RO}}} \\ \wedge\, \mathtt{Vfy}^{\mathcal{O}_{\mathsf{RO}}}(x, \pi) = 1 & w \leftarrow \mathtt{Ext}(x, \pi, \mathsf{tr}) \end{array}\right] \le \varepsilon(\lambda, q, n).$$

4.2.3.1 Simulation Security

We observe that when formalizing zero-knowledge, we use a simulator `Sim` who does not take the witness as input to generate the argument. For UC security, we prove security through a sequence of hybrid games. During game hopping, we replace the argument generated by `Prv` with one produced by `Sim`. However, given the access to `Sim`, the adversary $\mathcal{A}$ can use it to generate an argument without knowledge or the existence of a witness. Thus we need to consider security with respect to such simulated arguments to guarantee the knowledge or existence of witness.

Simulation Soundness. We first consider the notion of *simulation soundness* (SS), introduced by Sahai [Sah99], which is the "simulation version" of (adaptive) soundness in Definition 35. This notion requires that even after the adversary has seen simulated arguments on arbitrary instances, then it is guaranteed that the instance is in the language of the relation if the adversary constructs a new valid argument on any instance. As also indicated by Sahai [Sah99], simulation soundness implies the *non-malleability* [DDN91] of the non-interactive argument.

In Figure 4.3, we present the security game for simulation soundness. The adversary $\mathcal{A}$ has access to the random oracle $\mathcal{O}_{\mathsf{RO}}$ and a simulated-argument generation oracle $\mathcal{O}_{\mathsf{Sim}}$, which internally implements a simulator `Sim`. For each query made to $\mathcal{O}_{\mathsf{Sim}}$, the simulator `Sim` produces a list pg_s to *honestly* program the random oracle and generates a simulated proof π. This is to align with that of the simulator in the ideal world. Additionally, the adversary $\mathcal{A}$ can *adaptively* program $\mathcal{O}_{\mathsf{RO}}$ by querying the oracle $\mathcal{O}_{\mathsf{Prog}}$ with a query-answer pair (x, v). The winning condition is defined such that for an instance-argument pair (x, π) output by the adversary $\mathcal{A}$, the instance x has not been proved, the verifier algorithm `Vfy` decides to accept π, but x is not in the

Game: $\mathrm{G}^{\mathrm{SS}}_{\mathsf{NARG}}$

```
Procedure INIT
1 : $\mathcal{O}_{\mathsf{RO}} \leftarrow\!\!\$\ \mathcal{F}_\lambda$
2 : $Q := \varnothing$
3 : $\mathsf{pg}_a \leftarrow [\,]$
4 : $\mathsf{pg}_{all} \leftarrow [\,]$
```

```
Oracle $\mathcal{O}_{\mathsf{Prog}}(x, v)$
1 : $b := \mathcal{O}_{\mathsf{RO}}[(x, v)](\cdot)$
2 : if $b = 1$ then
3 :     $\mathsf{pg}_a := \mathsf{pg}_a \bowtie (x, v)$
4 :     $\mathsf{pg}_{all} := \mathsf{pg}_{all} \bowtie \mathsf{pg}_a$
5 : return $b$
```

```
Procedure FIN
1 : $(x, \pi) \xleftarrow{\mathsf{tr}_a} \mathcal{A}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}_{all}], \mathcal{O}_{\mathsf{Sim}}}$
2 : $b \xleftarrow{\mathsf{tr}_v} \mathtt{Vfy}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}_{all}]}(x, \pi)$
3 : return $(x \notin Q \wedge x \notin \mathcal{L}(R)$
4 :           $\wedge\, b = 1$
5 :           $\wedge\, \mathsf{tr}_v \cap \mathsf{pg}_a = \varnothing)$
```

```
Oracle $\mathcal{O}_{\mathsf{Sim}}(x, w)$
1 : if $(x, w) \notin R \vee |x| > n$ then
2 :     return $\perp$
3 : $Q := Q \cup \{x\}$
4 : $(\pi, \mathsf{pg}_s) \leftarrow \mathtt{Sim}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}_{all}]}(x)$
5 : $\mathsf{pg}_{all} := \mathsf{pg}_{all} \bowtie \mathsf{pg}_s$
6 : return $\pi$
```

Figure 4.3: SS game for a non-interactive argument NARG.

language of the relation R. We additionally note that the verification query trace tr_v made by $\mathtt{Vfy}$ should not intersect with the points programmed by $\mathcal{A}$. Otherwise, it yields a trivial win for $\mathcal{A}$.

Definition 37 (Simulation Soundness)**.** A non-interactive argument $\mathsf{NARG} = (\mathtt{Prv}, \mathtt{Vfy})$ for a relation R has *simulation soundness* error ε in *explicitly-programmable random oracle model* (EPROM) relative to a probabilistic polynomial-time simulator $\mathtt{Sim}$ if there exists a polynomial-time deterministic extractor $\mathtt{Ext}$ such that for any output size $\lambda \in \mathbb{N}$ of the random oracle $\mathcal{O}_{\mathsf{RO}}$, any instance size bound $n \in \mathbb{N}$, any admissible adversary $\mathcal{A}$ that makes $q_{ro} \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{RO}}$, $q_{pg} \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{Prog}}$, and $q_s \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{Sim}}$, it has $\mathbf{Adv}^{\mathrm{SS}}_{\mathsf{NARG}}(\mathcal{A}, \mathtt{Sim}) \leq \varepsilon(\lambda, q_{ro}, q_{pg}, q_s, n)$ where

$$\mathbf{Adv}^{\mathrm{SS}}_{\mathsf{NARG}}(\mathcal{A}, \mathtt{Sim}) := \Pr\left[\mathrm{G}^{\mathrm{SS}}_{\mathsf{NARG}}(\mathcal{A}, \mathtt{Sim}) \Rightarrow 1\right].$$

Simulation Extractability. Intuitively, one should be able to define a UC functionality that makes sense from simulation soundness. As discussed by Camenisch et al. [CKS11], such a functionality guarantees the *existence* of the witness. However, it does not necessarily mean that we can extract the witness from an instance-argument pair. In our case, we need to have the ideal-process adversary extract such a witness and use it to interact with the ideal functionality for the simulation. However, given that *extraction* of witness is not guaranteed, the environment $\mathcal{Z}$ could cause a simulation failure where the witness cannot be extracted, thus distinguishing the ideal process from the real execution through such a failure.

Thus to guarantee the extraction of the witness, we need to consider the notion of *Simulation-Extractability* [4] introduced by De Santis et al. [DDO+01]. The ideal functionality developed based on simulation-extractability has also been adopted by many prior works on UC-secure SNARK [GOS06, Gro06a, GKO+23, CF24].

[4]Chiesa and Fenzi [CF24] also termed this as UC-friendly Knowledge Soundness. Some other works also termed this notion as *simulation knowledge soundness*. Here we stick to straightline simulation-extractability since it is used in more works.

Game: $\mathrm{G}^{s\text{SIM-EXT}}_{\mathsf{NARG}}$

```
Procedure INIT
1 : $\mathcal{O}_{\mathsf{RO}} \leftarrow\$ \mathcal{F}_\lambda$
2 : $Q := \varnothing$
3 : $\mathsf{pg}_a \leftarrow [\,]$
4 : $\mathsf{pg}_{all} \leftarrow [\,]$

Oracle $\mathcal{O}_{\mathsf{Prog}}(x, v)$
1 : $b := \mathcal{O}_{\mathsf{RO}}[(x, v)](\cdot)$
2 : if $b = 1$ then
3 :     $\mathsf{pg}_a := \mathsf{pg}_a \bowtie (x, v)$
4 :     $\mathsf{pg}_{all} := \mathsf{pg}_{all} \bowtie \mathsf{pg}_a$
5 : return $b$

Procedure FIN
1 : $(x, \pi) \xleftarrow{\mathsf{tr}_a} \mathcal{A}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}_{all}], \mathcal{O}_{\mathsf{Sim}}}$
2 : $w \leftarrow \mathtt{Ext}(x, \pi, \mathsf{tr}_a \setminus \mathsf{pg}_a)$
3 : $b \xleftarrow{\mathsf{tr}_v} \mathtt{Vfy}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}_{all}]}(x, \pi)$
4 : return $(x \notin Q \land (x, w) \notin R$
5 :             $\land\, b = 1$
6 :             $\land\, \mathsf{tr}_v \cap \mathsf{pg}_a = \varnothing)$

Oracle $\mathcal{O}_{\mathsf{Sim}}(x, w)$
1 : if $(x, w) \notin R \lor |x| > n$ then
2 :     return $\perp$
3 : $Q := Q \cup \{x\}$
4 : $(\pi, \mathsf{pg}_s) \leftarrow \mathtt{Sim}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}_{all}]}(x)$
5 : $\mathsf{pg}_{all} := \mathsf{pg}_{all} \bowtie \mathsf{pg}_s$
6 : return $\pi$
```

Figure 4.4: sSIM-EXT game for a non-interactive argument NARG.

In Figure 4.4, we present the security game for *straightline* simulation-extractability. The adversary is given the same oracle $\mathcal{O}_{\mathsf{Sim}}$ and $\mathcal{O}_{\mathsf{Prog}}$ as in the definition for simulation soundness, and we add an additional call to an extractor `Ext`. The security is modeled by a "conflict" between the extractor `Ext` and the verifier algorithm `Vfy`. Eventually, the adversary $\mathcal{A}$ outputs a pair (x, π), which is provided to the extractor `Ext` alongside the adversary's query trace tr_a to $\mathcal{O}_{\mathsf{RO}}$ excluding the programmed points in pg_a. That is to ensure that the extractor does not depend on the programmed points.[5] The verification algorithm `Vfy` is also invoked with (x, π) to produce a bit b. The adversary's winning condition is defined as follows: (x, w) does not satisfy the relation R yet the verification algorithm `Vfy` accepts with $b = 1$. Also, the verification query trace tr_v intersects with the points programmed by $\mathcal{A}$.

Definition 38 (Straightline Simulation-Extractability (sSIM-EXT))**.** A non-interactive argument $\mathsf{NARG} = (\mathtt{Prv}, \mathtt{Vfy})$ for a relation R has *straightline simulation-extractability* error ε in *explicitly-programmable random oracle model* (EPROM) relative to a probabilistic polynomial-time simulator `Sim` if there exists a polynomial-time deterministic extractor `Ext` such that for any output size $\lambda \in \mathbb{N}$ of the random oracle $\mathcal{O}_{\mathsf{RO}}$, any instance size bound $n \in \mathbb{N}$, any admissible adversary $\mathcal{A}$ that makes $q_{ro} \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{RO}}$, $q_{pg} \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{Prog}}$, and $q_s \in \mathbb{N}$ queries to $\mathcal{O}_{\mathsf{Sim}}$, it has $\mathbf{Adv}^{\text{sSIM-EXT}}_{\mathsf{NARG}}(\mathcal{A}, \mathtt{Sim}) \le \varepsilon(\lambda, q_{ro}, q_{pg}, q_s, n)$ where

$$\mathbf{Adv}^{\text{sSIM-EXT}}_{\mathsf{NARG}}(\mathcal{A}, \mathtt{Sim}) := \Pr\left[\mathrm{G}^{s\text{SIM-EXT}}_{\mathsf{NARG}}(\mathcal{A}, \mathtt{Sim}) \Rightarrow 1\right].$$

Definition 39 (zkSNARK)**.** A succinct non-interactive argument $\mathsf{SNARG} = (\mathtt{Prv}, \mathtt{Vfy})$ is a *zero-knowledge succinct non-interactive argument of knowledge* (SNARK) if it also satisfies knowledge soundness and zero-knowledge.

Remark 17 (Impossibility of Extraction in plain CRS Model)**.** As discussed by Ganesh et al. [GKO+23], SNARKSs in the *plain* CRS model ([GGPR13, PHGR13, GWC19, MBKM19,

[5]We remark that it also works if we pass the programmed points to `Ext` but mark those as programmed by the adversary.

CHM$^+$20]) either fail to be black-box or straightline. The main insight is that if a witness holds enough entropy, the argument cannot encapsulate sufficient information about the witness, making extraction impossible unless the extractor is granted certain capabilities, such as access to the prover's internal random coins (non-black-box) or the ability to rewind the prover (non-straightline). This impossibility has also been formally analyzed by Campanelli et al. [CGKS23]. It can be observed that neither of these capabilities is possible in UC framework, implying that we can only consider zkSNARK in ROM.

Remark 18 (Non-Succinct zkNARKs in CRS Model)**.** Even though we focus on ROM in this this thesis, we stress that witness extraction is still achievable for *non-succinct* (the size of the argument string is at least the size of the witness for the proved nondeterministic computation) zkNARKs in CRS model. Specifically, Gorth [Gro06b] constructed a scheme based on prime order groups with a bilinear map whose proof size is linear in the circuit size. In addition to that, a common approach for UC-security is to include an encryption of the witness. Several constructions with this approach include [KZM$^+$15, ARS20, BS21]. Due to the inclusion of the encryption of the witness in the argument, these constructions are not succinct.

We remark that succinctness of a NARG is an important property. Note that we also need the multi-group HE to be *compact* as in Definition 11. This lets us obtain a message (ciphertext plus argument) that is compact. This ensure that, as the number of ciphertext increases, a significant communication overhead will not happen.

Remark 19 (UC-secure zkSNARK in CRS-ROM Model)**.** In Remark 16, we discuss the necessity of random oracle (RO) programmability. Notably, UC-secure zkSNARKs do exist in a model that provides both a common reference string and an *observable* RO. Ganesh et al. [GKO$^+$23] introduced a compiler that combines any (non-straightline) simulation-extractable zkSNARK with a polynomial commitment scheme secure under the *strong Diffie-Hellman assumption* (SDH). Their approach leverages the RO for straightline extraction, building on the technique introduced by Fischlin [Fis05].

However, Chiesa and Fenzi [CF24] demonstrated that *unconditional* UC-security is achievable with an RO that is both programmable and observable. Furthermore, the polynomial commitment scheme employed by Ganesh et al. is vulnerable to potential quantum attacks. This motivates us to focus directly on the plain ROM, where the RO is programmable, rather than relying on the CRS-ROM model.

4.3 UC View for Verifiability

In this section, we present a UC view for verifiability. We first show the ideal functionality $\mathcal{G}_{\mathsf{RO}}$ of a global random oracle that is programmable and observable as introduced by Camenisch et al. [CDG$^+$18], and the ideal functionality $\mathcal{F}_{\mathsf{zkSNARK}}$ of a zkSNARK introduced by Chiesa and Fenzi [CF24]. We then show the UC-realization of $\mathcal{F}_{\mathsf{zkSNARK}}$ in $\mathcal{G}_{\mathsf{RO}}$-hybrid model if a NARG satisfies the properties in Definitions 32, 34 and 38.

4.3.1 Global Random Oracle $\mathcal{G}_{\mathsf{RO}}$

In Figure 4.5, we illustrate a *restricted programmable and observable random oracle* (rpoRO) described by Camenisch et al. [CDG$^+$18], which is an assumption to realize the ideal functionality for zkSNARK. We note that global random oracle $\mathcal{G}_{\mathsf{RO}}$ operates with a party ITI $(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid})$, an adversary $\mathcal{S}$, and it is also directly accessible by the environment $\mathcal{Z}$.

Functionality: $\mathcal{G}_{\mathsf{RO}}$

Parameter: output length λ.
Participants: A party $\mathcal{P}_{\mathsf{pid}}$, an adversary $\mathcal{S}$, and the environment $\mathcal{Z}$.

Initalization

```
$\mathsf{Qry} := \varnothing$ // List of queries made.
$\mathsf{Illegal} := [\,]$ // List of illegitimate queries.
$\mathsf{Prog} := \varnothing$ // List of programmed entries.
```

Query

```
In $\langle \texttt{Query} : [x] \rangle \leftarrow \{(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid}), \mathcal{Z}\}$
  if $\nexists\, v : (x, v) \in \mathsf{Qry}$ then
    $v \leftarrow\!\!\$\ \{0,1\}^{\lambda}$
    $\mathsf{Qry} \leftarrow \mathsf{Qry} \cup \{(x, v)\}$
  $(\mathsf{sid}', x') := x$
  if $\mathsf{sid} \neq \mathsf{sid}' \vee \langle \texttt{Query} : [\cdot] \rangle \leftarrow \mathcal{Z}$ then
    $\mathsf{Illegal}[\mathsf{sid}'] := \mathsf{Illegal}[\mathsf{sid}'] \bowtie (x', v)$
  Out $\{\texttt{"val"} : [v]\} \rightarrow \{(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid}), \mathcal{S}\}$
```

Observation

```
In $\langle \texttt{Observe} : [\mathsf{sid}] \rangle \leftarrow \mathcal{S}$
  $\mathsf{tr} \leftarrow \mathsf{Illegal}[\mathsf{sid}]$
  Out $\{\texttt{"obv"} : [\mathsf{sid}, \mathsf{tr}]\} \rightarrow \mathcal{S}$
```

Program

```
In $\langle \texttt{ProgRO} : [x, v] \rangle \leftarrow \{\mathcal{Z}, \mathcal{S}\}$
  if $\exists\, v' : (x, v') \in \mathsf{Qry} \wedge v \neq v'$ then
    Out $\{\texttt{"cfm"} : [0]\} \rightarrow \{\mathcal{Z}, \mathcal{S}\}$
  else
    $\mathsf{Qry} := \mathsf{Qry} \cup \{(x, v)\}$
    $\mathsf{Prog} := \mathsf{Prog} \cup \{x\}$
    Out $\{\texttt{"cfm"} : [1]\} \rightarrow \{\mathcal{Z}, \mathcal{S}\}$
```

```
In $\langle \texttt{IsProg} : [x] \rangle \leftarrow \{(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid}), \mathcal{S}\}$
  $(\mathsf{sid}', x') := x$
  if $\mathsf{sid} \neq \mathsf{sid}'$
   $\wedge \langle \texttt{IsProg} : [\cdot] \rangle \leftarrow (\mathcal{P}_{\mathsf{pid}}, \mathsf{sid})$ then
    Abort
  $b := x \in_{?} \mathsf{Prog}$
  Out $\{\texttt{"chk"} : [b]\} \rightarrow \{(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid}), \mathcal{S}\}$
```

Figure 4.5: A restricted programmable and observable global random oracle (rpoRO) $\mathcal{G}_{\mathsf{RO}}$ described by Camenisch et al. [CDG+18].

Query. The input $\langle \texttt{Query} : [x] \rangle$ may come from either an ITI $(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid})$ or the environment $\mathcal{Z}$. Upon receiving the input, $\mathcal{G}_{\mathsf{RO}}$ checks whether there has already been an entry in the query history Qry. If not, then a value v is sampled uniformly at random from $\{0,1\}^{\lambda}$ where λ is the output length and the pair (x, v) is then store in Qry. Otherwise, the corresponding value is retrieved from Qry. Note that x in the query is parsed as (sid', x') for some session ID sid' and some value x'. If the query is from an ITI $(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid})$ in a different session i.e., $\mathsf{sid} \neq \mathsf{sid}'$, or the query is from the environment $\mathcal{Z}$, then such a query is deemed illegitimate and registered in the lookup table Illegal which contains the illegitimate queries.

Observation. We allow the ideal-process adversary $\mathcal{S}$ to observe illegitimate queries associated with a session sid by sending the input $\langle \texttt{Observe} : [\mathsf{sid}] \rangle$ to $\mathcal{G}_{\mathsf{RO}}$. Upon receiving this input, $\mathcal{G}_{\mathsf{RO}}$ retrieves the corresponding trace tr from $\mathsf{Illegal}$ and forwards it to $\mathcal{S}$.

This ability to observe the random oracle is necessary for the security proofs. Notably, while the environment $\mathcal{Z}$ has direct access to the RO, neither the real-world adversary $\mathcal{A}$ nor the ideal-process adversary $\mathcal{S}$ inherently knows the queries made by $\mathcal{Z}$. As pointed out by Canetti et al. [CJS14], a global setup functionality that provides only public information is ineffective for UC-security, since $\mathcal{S}$ gains no advantage over $\mathcal{A}$ through it. Intuitively, queries made by $\mathcal{Z}$ can act as *trapdoors*, enabling the simulator to extract information. For example, as illustrated in Figure 4.4, the extractor requires access to the adversary's query trace, which corresponds to the environment's queries in the UC setting.

By incorporating observability, we ensure that the simulator can access the environment's queries (made for the challenge session), while queries from honest parties remain hidden.

Programming. The environment $\mathcal{Z}$ and the ideal-process adversary $\mathcal{S}$ is additionally allowed to program the random oracle by sending the input $\langle \texttt{ProgRO} : [x, v] \rangle$ to $\mathcal{G}_{\mathsf{RO}}$. As indicated in Remark 16, it is necessary to allow $\mathcal{S}$ to *honestly* program the oracle. Upon such an input, we first check if there has already been an entry in Qry such that $(x, v') \neq (x, v)$. If yes, then it is not programmable, we then let $\mathcal{G}_{\mathsf{RO}}$ returns 0 to $\mathcal{S}$ indicating a failure in programming. Otherwise, this (x, v) is included in Qry. Then x is included in Prog to indicate that the output for x was programmed.

We allow a party $(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid})$ or the ideal-process adversary $\mathcal{S}$ to learn if an output is programmed by sending the input $\langle \texttt{IsProg} : [x] \rangle$. Note that x is similarly in the format (sid', x') and we let $\mathcal{G}_{\mathsf{RO}}$ abort if this request is from an out-of-session machine $(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid})$ with $\mathsf{sid} \neq \mathsf{sid}'$. Otherwise, b indicating whether $x \in \mathsf{Prog}$ is output to the requesting party. Note that the environment $\mathcal{Z}$ is not allowed to directly ask $\mathcal{G}_{\mathsf{RO}}$ if a point has been programmed. Instead, it must ask the adversary (or a corrupted party through the adversary) for that.

Remark 20 (RO at Global Level)**.** We note that $\mathcal{G}_{\mathsf{RO}}$ allows the analysis in the *explicitly*-programmable random oracle model (EPROM). Specifically, implementing the RO at a global level restricts the simulator $\mathcal{S}$ from having full control over the RO while still allowing it to program the RO by providing inputs to $\mathcal{G}_{\mathsf{RO}}$. In contrast, if $\mathcal{S}$ were to implement the RO "internally", the model would become a *fully*-programmable random oracle, which is too strong and may not be realistically achievable.

4.3.2 Ideal Functionality $\mathcal{F}_{\mathsf{zkSNARK}}$

In Figures 4.6 and 4.7, we present the ideal functionality $\mathcal{F}_{\mathsf{zkSNARK}}$[6], which is a simplified version of the one proposed by Chiesa and Fenzi [CF24]. Key differences include their explicit consideration of the query trace of $\texttt{Sim}$ to $\mathcal{G}_{\mathsf{RO}}$ and their alternative approach to describing verification steps.

Furthermore, their functionality is defined for a specific session, whereas in this thesis, we model the input to $\mathcal{F}_{\mathsf{zkSNARK}}$ as coming from a *party* ITM $\mathcal{P}_{\mathsf{pid}}$ rather than an ITI $(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid})$. This allows $\mathcal{P}_{\mathsf{pid}}$ to instantiate multiple sessions of the functionality within a protocol by spawning multiple ITIs, each interacting with a separate instance of the functionality for that session. This design better reflects real-world cases.

In their work [CF24], they also introduced a stronger model in which the random coins used for argument generation and verification is disclosed to the adversary when a party is

[6]We do not (nor can we) define succinctness in this functionality, as it pertains to efficiency rather than correctness, as stated in Definition 29.

corrupted. This is specifically for the case of *adaptive corruption*, which was also discussed by Lysyanskaya and Rosenbloom [LR22a]. Since we only consider static corruption, the definitions provided in Section 4.2 and the simplified functionality described in Figures 4.6 and 4.7 suffice for the security proofs in this work. For more details on UC-secure zkSNARKs against adaptive corruption, we refer readers to their original work [CF24].

Functionality: $\mathcal{F}^{R}_{\mathsf{zkSNARK}}$ (Part I)

Parameter: a binary relation R, an instance bound n.
Participants: a prover $\mathcal{P}_{\mathsf{pvr}}$, a verifier $\mathcal{P}_{\mathsf{vfr}}$, and an ideal-process adversary $\mathcal{S}$.

Initialization

```
1 : $\mathsf{Proved} := [\,]$ // list of proved statements.
2 : $\mathsf{ExtTr} := [\,]$ // list of queries of the adversary and the simulator to the GROM
3 : $\mathsf{Inst} := [\,]$ // list of proved instance
4 : $\mathsf{HProg} := [\,]$ // list of honestly programmed points.
5 : $\mathsf{Algo} := [\,]$ // A tuple of algorithms
6 : $\mathsf{Corrupt} := \varnothing$ // Set of corrupted parties
```

Corruption

```
1 : In $\langle \mathtt{Corrupt} : [\mathsf{pid}] \rangle \leftarrow \mathcal{S}$
2 :   $\mathsf{Corrupt} \leftarrow \mathsf{Corrupt} \cup \{\mathsf{pid}\}$
```

Figure 4.6: Ideal functionality $\mathcal{F}_{\mathsf{zkSNARK}}$ (Part I – Initialization and Corruption) for a zkSNARK by Chiesa and Fenzi [CF24].

Algorithm Setup. Upon receiving the input $\langle \mathtt{Setup} : [\mathsf{sid}] \rangle$ from the prover $\mathcal{P}_{\mathsf{pvr}}$, we let $\mathcal{F}_{\mathsf{zkSNARK}}$ forward it to $\mathcal{S}$ to setup a tuple of algorithm $(\mathtt{Vfy}, \mathtt{Sim}, \mathtt{Ext})$ where $\mathtt{Vfy}$ is the verification algorithm, $\mathtt{Sim}$ and $\mathtt{Ext}$ are the simulator and extractor defined in Definitions 33 and 38.

Here, we directly invoke $\mathtt{Sim}$ for simulation and $\mathtt{Ext}$ for extraction in $\mathcal{F}_{\mathsf{zkSNARK}}$, without forwarding the respective inputs to the ideal-process adversary $\mathcal{S}$ for these tasks. As also indicated by Chiesa and Fenzi [CF24], this means that only *non-interactive* argument can realize this functionality.

Argument Generation. We let the prover $\mathcal{P}_{\mathsf{pvr}}$ send the input $\langle \mathtt{Prove} : [\mathsf{sid}, x, w] \rangle$ to $\mathcal{F}_{\mathsf{zkSNARK}}$ in order to generate an argument for the instance-witness pair (x, w). Upon receiving this input, $\mathcal{F}_{\mathsf{zkSNARK}}$ first verifies whether (x, w) satisfies the relation R and $|x| < n$. If these conditions are not met, $\mathcal{F}_{\mathsf{zkSNARK}}$ aborts the request. Otherwise, $\mathcal{F}_{\mathsf{zkSNARK}}$ requests $\mathcal{S}$ to observe $\mathcal{G}_{\mathsf{RO}}$ and retrieve the list of queries it has made. Subsequently, $\mathcal{F}_{\mathsf{zkSNARK}}$ invokes $\mathtt{Sim}$ on x, which outputs an argument π along with a query-answer list pg. By definition, $\mathtt{Sim}$ has the ability to modify random oracle answers to produce a simulated proof. Thus $\mathcal{F}_{\mathsf{zkSNARK}}$ forwards pg to $\mathcal{S}$ to program $\mathcal{G}_{\mathsf{RO}}$ accordingly. Finally, (x, π) is added to the set Proved, and the pair (x, π) is returned to the prover $\mathcal{P}_{\mathsf{pvr}}$.

Verification. We let the verifier $\mathcal{P}_{\mathsf{vfr}}$ send the input $\langle \mathtt{Verify} : [\mathsf{sid}, x, \pi] \rangle$ to $\mathcal{F}_{\mathsf{zkSNARK}}$ to verify the argument π for an instance x. We first check if there is a record (x, π) in the lookup table Proved. If yes, then we just let $\mathcal{F}_{\mathsf{zkSNARK}}$ return 1 to $\mathcal{P}_{\mathsf{vfr}}$. This models the *completeness* where the every argument generated legitimately should be accepted.

Functionality: $\mathcal{F}^{R}_{\mathsf{zkSNARK}}$ (Part II)

Setup

```
In ⟨Setup : [sid]⟩ ↼ $\mathcal{P}_{\mathsf{pid}}$
    if $\mathsf{Algo}[\mathsf{sid}] \neq \bot \lor \mathsf{pid} \in \mathsf{Corrupt}$ then
        Abort
    else
        Out ⟨Setup : [sid]⟩ ⇀ $\mathcal{S}$
    In {"alg" : [sid, (Vfy, Ext, Sim)]} ↼ $\mathcal{S}$
    $\mathsf{Algo}[\mathsf{sid}] := (\mathtt{Vfy}, \mathtt{Ext}, \mathtt{Sim})$
```

Argument Generation

```
In ⟨Prove : [sid, $x$, $w$]⟩ ↼ $\mathcal{P}_{\mathsf{pvr}}$
    if $\mathsf{Algo}[\mathsf{sid}] = \bot \lor |x| > n$
     $\lor\ \mathsf{pvr} \in \mathsf{Corrupt}$ then
        Abort
    if $(x, w) \notin R$ then
        Abort
    Out ⟨Observe : [sid]⟩ ⇀ $\mathcal{S}$
    In {"obv" : [sid, $\mathsf{tr}_z$]} ↼ $\mathcal{S}$
    $\mathsf{ExtTr}[\mathsf{sid}] := \mathsf{ExtTr}[\mathsf{sid}] \bowtie \mathsf{tr}$
    $(\pi, \mathsf{pg}) \leftarrow \mathtt{Sim}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x)$
    for $(x, v) \in \mathsf{pg}$ do
        Out ⟨ProgRO : [(sid, $x$), $v$]⟩ ⇀ $\mathcal{S}$
        In {"cfm" : [$b$]} ↼ $\mathcal{S}$
        if $b = 0$ then
            Abort
    $\mathsf{Proved}[\mathsf{sid}] := \mathsf{Proved}[\mathsf{sid}] \bowtie (x, \pi)$
    $\mathsf{HProg}[\mathsf{sid}] := \mathsf{HProg}[\mathsf{sid}] \bowtie \mathsf{pg}$
    $\mathsf{Inst}[\mathsf{sid}] := \mathsf{Inst}[\mathsf{sid}] \bowtie x$
    Out {"pf" : [sid, $x$, $\pi$]} ⇀ $\mathcal{P}_{\mathsf{pvr}}$
```

Verification

```
In ⟨Verify : [sid, $x$, $\pi$]⟩ ↼ $\mathcal{P}_{\mathsf{vfr}}$
    if $\mathsf{Algo}[\mathsf{sid}] = \bot \lor |x| > n$
     $\lor\ \mathsf{vfr} \in \mathsf{Corrupt}$ then
        Abort
    if $(x, \pi) \in \mathsf{Proved}[\mathsf{sid}]$ then
        Out {"vf" : [sid, $x$, $\pi$, 1]} ⇀ $\mathcal{P}_{\mathsf{vfr}}$
    else
        $b_v \xleftarrow{\mathsf{tr}_v} \mathtt{Vfy}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x, \pi)$
        for $(x, v) \in \mathsf{tr}_v \setminus \mathsf{HProg}[\mathsf{sid}]$ do
            Out ⟨IsProg : [(sid, $x$)]⟩ ⇀ $\mathcal{G}_{\mathsf{RO}}$
            In {"chk" : [$b$]} ↼ $\mathcal{G}_{\mathsf{RO}}$
            if $b = 1$ then
                $b_{ap} := 1$
        Out ⟨Observe : [sid]⟩ ⇀ $\mathcal{S}$
        In {"obv" : [sid, $\mathsf{tr}_z$]} ↼ $\mathcal{S}$
        $\mathsf{ExtTr}[\mathsf{sid}] := \mathsf{ExtTr}[\mathsf{sid}] \bowtie \mathsf{tr}_z$
        for $(x, v) \in \mathsf{ExtTr}[\mathsf{sid}]$ do
            Out ⟨IsProg : [(sid, $x$)]⟩ ⇀ $\mathcal{S}$
            In {"chk" : [$b$]} ↼ $\mathcal{S}$
            if $b = 0 \lor (x, w) \in \mathsf{HProg}[\mathsf{sid}]$ then
                $\mathsf{tr}'_z := \mathsf{tr}'_z \bowtie (x, v)$
        $w \leftarrow \mathtt{Ext}(x, \pi, \mathsf{tr}'_z)$
        if $b_v = 0 \lor b_{ap} = 1$ then
            Out {"vf" : [sid, $x$, $\pi$, 0]} ⇀ $\mathcal{P}_{\mathsf{vfr}}$
        elseif $(x, w) \notin R \land x \notin \mathsf{Inst}[\mathsf{sid}]$ then
            Out {"vf" : [sid, $x$, $\pi$, $\bot$]} ⇀ $\mathcal{P}_{\mathsf{vfr}}$
        else
            Out {"vf" : [sid, $x$, $\pi$, 1]} ⇀ $\mathcal{P}_{\mathsf{vfr}}$
```

Figure 4.7: Ideal functionality $\mathcal{F}_{\mathsf{zkSNARK}}$ (Part II – Setup, Argument generation, and verification) for a zkSNARK by Chiesa and Fenzi [CF24].

Otherwise, we proceed as follows: $\mathcal{F}_{\mathsf{zkSNARK}}$ invokes the verification algorithm `Vfy`, which outputs a bit b_v indicating the conclusion from `Vfy`. Let tr_v denote the query trace generated by `Vfy`. Next, $\mathcal{F}_{\mathsf{zkSNARK}}$ queries $\mathcal{G}_{\mathsf{RO}}$ to check whether `Vfy` has queried any points that were programmed by the environment. Subsequently, $\mathcal{F}_{\mathsf{zkSNARK}}$ requests $\mathcal{S}$ to observe $\mathcal{G}_{\mathsf{RO}}$ and retrieve the query trace tr_z made by the environment to $\mathcal{G}_{\mathsf{RO}}$. From tr_z, we exclude all points that were previously programmed by the environment to produce a refined trace tr'_z. The tuple (x, π, tr'_z) is then passed to the extractor `Ext`, which outputs the witness w.

At this point, if $b_v = 0$ or tr_v contains any point programmed by the environment $\mathcal{Z}$, then $\mathcal{F}_{\mathsf{zkSNARK}}$ returns 0 to the verifier. If $(x, w) \notin R$ and the instance x has not been proved i.e., $x \notin \mathsf{Inst}[\mathsf{sid}]$, this signals an inconsistency between the verification outcome from `Vfy` and the extractor `Ext`. In such a case, $\mathcal{F}_{\mathsf{zkSNARK}}$ returns $\bot$ to the verifier, indicating a failure. Notably, if the adversary manages to convince `Vfy` to output 1 in the real world, the environment $\mathcal{Z}$ can then distinguish between the ideal and real worlds by observing the appearance of $\perp$, as discussed in Section 4.2.3.1. Finally, if none of these conditions are met, verification succeeds, and $\mathcal{F}_{\mathsf{zkSNARK}}$ outputs the bit 1 to the verifier.

Remark 21. In Figure 4.7 and later in Figure 4.8, we use the notation $\texttt{alg}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}$ to represent that the party $\mathcal{P}_{\mathsf{pid}}$ invokes the algorithm $\texttt{alg}$ and queries $\mathcal{G}_{\mathsf{RO}}$ in the session sid (which is also called a *domain separated oracle*). Specifically, this means that the ITI $(\mathcal{P}_{\mathsf{pid}}, \mathsf{sid})$ sends the input $\langle \texttt{Query} : [(\mathsf{sid}, x)] \rangle$ to $\mathcal{G}_{\mathsf{RO}}$, where x corresponds to the "actual" query made by $\texttt{alg}$.

4.3.3 Realizing $\mathcal{F}_{\mathsf{zkSNARK}}$ with $\mathcal{G}_{\mathsf{RO}}$

In this section, we show that we can realize $\mathcal{F}_{\mathsf{zkSNARK}}$ given a global random oracle $\mathcal{G}_{\mathsf{RO}}$ if the non-interactive argument satisfies completeness, zero-knowledge and straightline simulation-extractability.

Protocol: Π_{NARG}

Parameter: a binary relation R, an instance bound n.
Participants: a prover $\mathcal{P}_{\mathsf{pvr}}$ and a verifier $\mathcal{P}_{\mathsf{vfr}}$.

Argument Generation

```
$\mathcal{P}_{\mathsf{pvr}} \leftarrow \langle \texttt{Prove} : [\mathsf{sid}, x, m] \rangle$
    if $(x, m) \notin R \vee |x| > n$ then
        Abort
    $\pi \leftarrow \texttt{Prv}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x, w)$
    Out $\{\texttt{"pf"} : [\mathsf{sid}, x, \pi]\}$
```

Verification

```
$\mathcal{P}_{\mathsf{vfr}} \leftarrow \langle \texttt{Verify} : [\mathsf{sid}, x, \pi] \rangle$
    if $|x| > n$ then
        Abort
    $b_v \xleftarrow{\mathsf{tr}_v} \texttt{Vfy}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x, \pi)$
    for $(x, v) \in \mathsf{tr}_v$ do
        Out $\langle \texttt{IsProg} : [x] \rangle \rightarrow \mathcal{G}_{\mathsf{RO}}$
        In $\{\texttt{"chk"} : [b']\} \leftarrow \mathcal{G}_{\mathsf{RO}}$
        if $b' = 1$ then
            $b_{pg} := 0$
    Out $\{\texttt{"vf"} : [\mathsf{sid}, x, \pi, b_v \wedge b_{pg}]\}$
```

Figure 4.8: Protocol Π_{NARG} of a non-interactive argument.

4.3.3.1 Protocol Π_{NARG} and Simulator $\mathcal{S}_{\mathsf{NARG}}$

In Figure 4.8, we illustrate the protocol Π_{NARG} that implements a non-interactive argument. When the prover $\mathcal{P}_{\mathsf{pvr}}$ is activated with an instance-witness pair (x, w), it first verifies whether $(x, w) \in R$ and whether the instance size $|x|$ is bounded by n. If either condition fails, $\mathcal{P}_{\mathsf{pvr}}$ aborts. Otherwise, it invokes the prover algorithm $\texttt{Prv}$ to generate and output the argument π.

When the verifier $\mathcal{P}_{\mathsf{vfr}}$ is activated with an instance-argument pair (x, π), it begins by checking whether $|x| < n$. If this condition is not satisfied, $\mathcal{P}_{\mathsf{vfr}}$ aborts. Otherwise, it invokes the verifier algorithm $\texttt{Vfy}$ to evaluate the argument π. We let b_v denote the decision made by $\texttt{Vfy}$. Furthermore, $\mathcal{P}_{\mathsf{vfr}}$ queries $\mathcal{G}_{\mathsf{RO}}$ to determine whether the query trace tr_v made by $\texttt{Vfy}$ intersects with the points programmed by the environment. If an intersection exists, we set $b_{pg} = 0$, indicating that the verifier should reject the argument, since in this case an adversary can trivially break the security. Finally, $\mathcal{P}_{\mathsf{vfr}}$ outputs $b_v \wedge b_{pg}$ as the final decision to accept or reject the argument.

We note that the simulator $\mathcal{S}_{\mathsf{NARG}}$ is only responsible for forwarding input $\langle \texttt{Observe} : [\mathsf{sid}] \rangle$ from $\mathcal{F}_{\mathsf{zkSNARK}}$ to obtain the illegitimate query trace from $\mathcal{G}_{\mathsf{RO}}$ and the input $\langle \texttt{ProgRO} : [x, v] \rangle$ to *honestly* program $\mathcal{G}_{\mathsf{RO}}$. As also illustrated in Figure 4.7, all the simulation for argument generation and extraction are done with `Sim` and `Ext` respectively. Thus we omit the description for $\mathcal{S}_{\mathsf{NARG}}$.

4.3.3.2 Proof of Secure Realization

In Theorem 4, we show that the protocol Π_{NARG} realizes the functionality $\mathcal{F}_{\mathsf{zkSNARK}}$. Note that we can take the real-world adversary $\mathcal{A}$ as a *dummy adversary* $\mathcal{A}_D$ described by Canetti et al. [Can20], which means the adversary $\mathcal{A}_D$ simply delivers messages from the environment $\mathcal{Z}$ to the specified party, and delivers to the environment $\mathcal{Z}$ all messages generated by the protocol parties. Essentially, this means that the environment $\mathcal{Z}$ has the full control of the dummy adversary $\mathcal{A}_D$.

$$\begin{tikzcd}
\boxed{\mathsf{REAL}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}} \arrow[r, bend right, "\begin{array}{c}\text{Comp}\\ \text{Claim 4.1}\end{array}"'] & \boxed{\mathsf{HYB}^{1\,\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}} \arrow[r, bend right, "\begin{array}{c}\text{ZK}\\ \text{Claim 4.2}\end{array}"'] & \boxed{\mathsf{HYB}^{2\,\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}} \arrow[r, bend right, "\begin{array}{c}s\text{SIM-EXT}\\ \text{Claim 4.3}\end{array}"'] & \boxed{\mathsf{IDEAL}^{\mathcal{G}_{\mathsf{RO}}}_{\mathcal{F}_{\mathsf{zkSNARK}},\mathcal{S},\mathcal{Z}}}
\end{tikzcd}$$

Figure 4.9: An illustration of hybrid games and reduction in the security proofs for $\mathcal{F}_{\mathsf{zkSNARK}}$ realization.

Our proof largely builds on the work of Chiesa and Fenzi [CF24, Section 6], providing additional details on the perfectness of the simulated game/execution by an adversary $\mathcal{A}$ or the environment $\mathcal{Z}$ in a reduction. Furthermore, we modify their approach to handling queries from $\mathcal{Z}$ to $\mathcal{G}_{\mathsf{RO}}$ to take care of the appearance of multiple session IDs in queries. This adjustment, however, does not impact the established results.

Theorem 4. *Let* NARG *be a non-interactive argument. The protocol* Π_{NARG} *UC-realizes* $\mathcal{F}_{\mathsf{zkSNARK}}$ *in* $\mathcal{G}_{\mathsf{RO}}$*-hybrid model against a (dummy) adversary* $\mathcal{A}$ *if and only if* NARG *satisfies* Comp, ZK, sSIM-EXT, *and it is succinct.*

Proof. To prove the security of the protocol, we define the following sequence of hybrid games.

– $\mathsf{REAL}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$: This represents the execution of the protocol Π_{NARG} in the real world and an adversary $\mathcal{A}$ attacking the protocol.

– $\mathsf{HYB}^{1\,\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$: In this hybrid game, we modify Π_{NARG} such that every instance-argument pair (x, π) produced by an honest $\mathcal{P}_{\mathsf{pvr}}$ always yields $b_v = 1$. The value b_{gp} still depends on if there is an intersection between $\mathtt{tr}_v$ and points programmed by $\mathcal{Z}$.

– $\mathsf{HYB}^{2\,\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$: In this hybrid game, whenever an honest prover $\mathcal{P}_{\mathsf{pvr}}$ is activated with the input $\langle \texttt{Prove} : [\mathsf{sid}, x, w] \rangle$, the prover outputs the argument generated by the simulator. Specifically $\pi \leftarrow \texttt{Sim}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x, w)$. The query-answer list pg output by `Sim` is also programmed at $\mathcal{G}_{\mathsf{RO}}$.

  When an honest prover $\mathcal{P}_{\mathsf{vfr}}$ is activated with the input $\langle \texttt{Verify} : [\mathsf{sid}, x, w] \rangle$, we exclude the honestly programmed points made by `Sim` from the query trace $\mathtt{tr}_v$ made by `Vfy` and only check if it intersects with the points programmed by the adversary.

– $\mathsf{IDEAL}^{\mathcal{G}_{\mathsf{RO}}}_{\mathcal{F}_{\mathsf{zkSNARK}},\mathcal{S},\mathcal{Z}}$: This is the ideal world where the ideal functionality $\mathcal{F}_{\mathsf{zkSNARK}}$ is executed.

Claim 4.1. *$\mathsf{REAL}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{HYB}^{1\,\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ are indistinguishable if and only if* NARG *satisfies* Comp.

Proof. ($\Rightarrow$) We begin by proving the direction where if there exists an environment $\mathcal{Z}$ that distinguishes between these two worlds, we can then construct an adversary $\mathcal{A}_{cp}$ against Comp of NARG. We let $\mathcal{A}_{cp}$ handles its oracle queries based on the input provided by $\mathcal{Z}$ as follows.

– **On the input $\langle \texttt{Query} : [x] \rangle$ to $\mathcal{G}_{\mathsf{RO}}$**: We let $\mathcal{A}_{cp}$ parse the query as (sid, x'). If sid corresponds to the challenge session, $\mathcal{A}_{cp}$ queries $\mathcal{O}_{\mathsf{RO}}$ with x'. Otherwise, it queries the oracle with (sid, x'). Let v be the output of $\mathcal{O}_{\mathsf{RO}}$, which $\mathcal{A}_{cp}$ then uses to respond to $\mathcal{Z}$.

  We note that an honest prover $\mathcal{P}_{\mathsf{pvr}}$ and an honest verifier $\mathcal{P}_{\mathsf{vfr}}$ always query the oracle in the form (sid, x), where sid represents the challenge session. Given how oracle queries are handled, this effectively transforms the random oracle's behavior to depend only on x rather than (sid, x) when sid remains the same. As a result, $\mathcal{A}_{cp}$ can use its own random oracle $\mathcal{O}_{\mathsf{RO}}$ (which does *not* take sid as input) to simulate the domain-separated oracle $\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]$ (which does take sid as input).

– **On the input $\langle \texttt{ProgRO} : [x, v] \rangle$ to $\mathcal{G}_{\mathsf{RO}}$**: Similarly, we let $\mathcal{A}_{cp}$ parse the query as (sid, x'). If sid represents the challenge session, we let query $\mathcal{O}_{\mathsf{Prog}}$ with (x', v). Otherwise, we let $\mathcal{A}_{cp}$ program the RO with $((\mathsf{sid}, x'), v)$. Based on whether the programming is successful, we let $\mathcal{A}_{zk}$ return 0 and 1 to the environment $\mathcal{Z}$ respectively.

– **On the input $\langle \texttt{Prove} : [\mathsf{sid}, x, w] \rangle$ to an honest prover $\mathcal{P}_{\mathsf{pvr}}$.** We let $\mathcal{A}_{cp}$ query its oracle $\mathcal{O}_{\mathsf{Prv}}$ to generate an argument π. If $\perp$ is returned, we then let $\mathcal{A}_{cp}$ abort on this query. Otherwise, the argument is generated as $\pi \leftarrow \texttt{Prv}^{\mathcal{O}_{\mathsf{RO}}}(x, w)$.

  Note that in both two worlds, the arguments is generated as $\pi \leftarrow \texttt{Prv}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x, w)$. And as mentioned earlier, the oracle $\mathcal{O}_{\mathsf{RO}}$ perfectly simulates the behavior of $\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]$. Thus the behavior of $\mathcal{O}_{\mathsf{Prv}}$ corresponds the behavior of an honest prover when it is activated with the input. Therefore, we write π as the output of $\mathcal{P}_{\mathsf{pvr}}$.

– **On the input $\langle \texttt{Verify} : [\mathsf{sid}, x, w] \rangle$ to an honest verifier $\mathcal{P}_{\mathsf{vfr}}$.** We let $\mathcal{A}_{cp}$ query its oracle $\mathcal{O}_{\mathsf{Vfy}}$ for verification. If $\perp$ is returned, we then let $\mathcal{A}_{cp}$ abort on this query. Otherwise, we let $\mathcal{A}_{cp}$ answer $\mathcal{Z}$ with the bit b output by $\mathcal{O}_{\mathsf{Vfy}}$.

We first observe that in both worlds, if the query trace $\mathtt{tr}_v$ made by $\texttt{Vfy}$ intersects with the points programmed by the environment, b_{pg} is set to 0 in both worlds. Consequently, the two worlds differ only when $b_v = 0$ in $\mathsf{REAL}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$, while $b_v = 1$ in $\mathsf{HYB}^{1\,\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$, where $b_v \leftarrow \texttt{Vfy}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x, \pi)$ for an instance-argument pair (x, π) produced by an honest prover $\mathcal{P}_{\mathsf{pvr}}$. This directly corresponds to the winning condition of $\mathrm{G}^{\mathrm{Comp}}_{\mathsf{NARG}}$. Therefore, when $\mathcal{Z}$ distinguishes these two worlds using the instance-argument pair (x, π), the adversary $\mathcal{A}_{cp}$ wins the game $\mathrm{G}^{\mathrm{Comp}}_{\mathsf{NARG}}$ with the same pair. Thus, we have that

$$\Delta_{\mathcal{Z}}\left(\mathsf{REAL}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}, \mathsf{HYB}^{1\,\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}\right) \leq \mathbf{Adv}^{\mathrm{Comp}}_{\mathsf{NARG}}(\mathcal{A}_{cp}). \tag{4.1}$$

($\Leftarrow$) We now show the direction where if there exists an adversary $\mathcal{A}_{cp}$ that breaks the completeness of NARG, then we can use it to construct an environment $\mathcal{Z}$ that distinguishes between these two worlds. We then let $\mathcal{Z}$ handles the queries made by $\mathcal{A}_{cp}$ as follows.

– **On the query x to $\mathcal{O}_{\mathsf{RO}}$**: We let the environment $\mathcal{Z}$ send the input $\langle \texttt{Query} : [(\mathsf{sid}, x)] \rangle$ to $\mathcal{G}_{\mathsf{RO}}$ where sid denotes the challenge session. Upon receiving an output v, we let $\mathcal{Z}$ return this v to $\mathcal{A}_{zk}$ as response.

We note that as the converse direction, we now transform the random oracle's behavior to depend on (sid, x) instead of x. This then allows the environment $\mathcal{Z}$ to use the domain-separated oracle $\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]$ to simulate the behavior of $\mathcal{O}_{\mathsf{RO}}$.

- **On the query** (x, v) **to** $\mathcal{O}_{\mathsf{Prog}}$: We let the environment $\mathcal{Z}$ send the input $\langle \mathtt{ProgRO} : [(\mathsf{sid}, x), v]\rangle$ to $\mathcal{G}_{\mathsf{RO}}$. Upon receiving the bit indicating whether the programming was successful, we let $\mathcal{Z}$ return this bit to $\mathcal{A}_{zk}$ as response.
- **On the query** (x, w) **to** $\mathcal{O}_{\mathsf{Prv}}$: We let the environment $\mathcal{Z}$ activate the prover $\mathcal{P}_{\mathsf{pvr}}$ with the input $\langle \mathtt{Prove} : [\mathsf{sid}, x, w]\rangle$. Observe that in both of these worlds, an argument π is generated as $\pi \leftarrow \mathtt{Prv}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x, w)$.

  Note that at $\mathcal{O}_{\mathsf{Vfy}}$, the arguments is generated as $\pi \leftarrow \mathtt{Prv}^{\mathcal{O}_{\mathsf{RO}}}(x, w)$. And as mentioned earlier, the oracle $\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]$ perfectly simulates the behavior of $\mathcal{O}_{\mathsf{RO}}$. Thus we let $\mathcal{Z}$ answer $\mathcal{A}_{cp}$'s query with π output by $\mathcal{P}_{\mathsf{pvr}}$.
- **On the query** (x, π) **to** $\mathcal{O}_{\mathsf{Prv}}$: We let the environment $\mathcal{Z}$ activate the verifier $\mathcal{P}_{\mathsf{vfr}}$ with input $\langle \mathtt{Verify} : [\mathsf{sid}, x, \pi]\rangle$. When $\mathcal{P}_{\mathsf{vfr}}$ outputs a bit to indicate acceptance or rejection, we let $\mathcal{Z}$ return this bit to $\mathcal{A}_{cp}$ as answer.

We observe that in both $\mathbf{REAL}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{HYB}^{1\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$, if the query trace $\mathtt{tr}_v$ made by $\mathtt{Vfy}$ intersects with the points programmed by the environment, b_{pg} is set to 0 in both worlds. To win the game $\mathrm{G}^{\mathrm{Comp}}_{\mathsf{NARG}}$, the adversary $\mathcal{A}_{cp}$ must first pass an instance-witness pair (x, w) to $\mathcal{O}_{\mathsf{Prv}}$ to get an argument π, and this (x, π) when passed to $\mathcal{O}_{\mathsf{Vfy}}$ yield a rejection. Then if $\mathcal{P}_{\mathsf{vfr}}$ outputs 0, the environment $\mathcal{Z}$ distinguishes it as the real execution. Otherwise, $\mathcal{Z}$ distinguishes it the first hybrid game. Thus we have that

$$\mathbf{Adv}^{\mathrm{Comp}}_{\mathsf{NARG}}(\mathcal{A}_{cp}) \leq \Delta_{\mathcal{Z}}\left(\mathbf{REAL}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}, \mathbf{HYB}^{1\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}\right). \tag{4.2}$$

□

Claim 4.2. *$\mathbf{HYB}^{1\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{HYB}^{2\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ are indistinguishable if and only if* NARG *satisfies* ZK *with respect to a simulator* $\mathtt{Sim}$.

Proof. $(\Rightarrow)$ We first show the direction where if there exists an environment $\mathcal{Z}$ that distinguishes between $\mathbf{HYB}^{1\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{HYB}^{2\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$, then we can use it to construct an adversary $\mathcal{A}_{zk}$ that breaks zero-knowledge of NARG. We let $\mathcal{A}_{zk}$ handles its queries to its oracles as follows based on the input from the environment $\mathcal{Z}$.

- **On the input** $\langle \mathtt{Query} : [x]\rangle$ **to** $\mathcal{G}_{\mathsf{RO}}$: This is the same as in proof of Claim 4.1.
- **On the input** $\langle \mathtt{ProgRO} : [x, v]\rangle$ **to** $\mathcal{G}_{\mathsf{RO}}$: This is the same as in proof of Claim 4.1.
- **On the input** $\langle \mathtt{Prove} : [\mathsf{sid}, x, w]\rangle$ **to an honest prover** $\mathcal{P}_{\mathsf{pvr}}$: We let $\mathcal{A}_{zk}$ pass this instance-witness pair (x, w) to its oracle $\mathcal{O}_{\mathsf{Prv}}$. Observe that an argument is then either generated as $\pi \leftarrow \mathtt{Prv}^{\mathcal{O}_{\mathsf{RO}}}(x, w)$ or $\pi \leftarrow \mathtt{Sim}^{\mathcal{O}_{\mathsf{RO}}}(x)$, which exactly corresponds to the behavior of an honest prover $\mathcal{P}_{\mathsf{pvr}}$ in $\mathbf{HYB}^{1\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{HYB}^{2\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ respectively. Thus we write π as the output of the prover $\mathcal{P}_{\mathsf{pvr}}$.
- **On the input** $\langle \mathtt{Verify} : [\mathsf{sid}, x, \pi]\rangle$ **to an honest verifier** $\mathcal{P}_{\mathsf{vfr}}$: We let $\mathcal{A}_{zk}$ forward this instance-argument pair to its oracle $\mathcal{O}_{\mathsf{Vfy}}$. By definition, if (x, π) is in the list of proved instances, the oracle $\mathcal{O}_{\mathsf{Vfy}}$ returns 1. Otherwise, $\mathcal{O}_{\mathsf{Vfy}}$ invokes $\mathtt{Vfy}$ to produce b_v as the decision. Up to this point, the behavior of $\mathcal{O}_{\mathsf{Vfy}}$ corresponds to that of $\mathcal{P}_{\mathsf{vfr}}$ in both worlds.

Additionally, note that in both worlds, we only check the intersection between tr_v and the points programmed by the environment $\mathcal{Z}$ (excluding the honestly programmed points by $\mathtt{Sim}$ in $\mathbf{HYB2}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$). This behavior aligns with that of $\mathcal{O}_{\mathsf{Vfy}}$. Hence, we write the bit output by $\mathcal{O}_{\mathsf{Vfy}}$ as the output of $\mathcal{P}_{\mathsf{vfr}}$.

Thus we have $\mathcal{A}_{zk}$ simulate the interaction between $\mathcal{Z}$ and the two worlds. Then if the environment $\mathcal{Z}$ output a bit b, we then let $\mathcal{A}_{zk}$ output the same bit. Thus we have that:

$$\Delta_{\mathcal{Z}}\left(\mathbf{HYB1}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}, \mathbf{HYB2}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}\right) \leq \mathbf{Adv}^{\mathrm{ZK}}_{\mathsf{NARG}}(\mathcal{A}, \mathtt{Sim}). \tag{4.3}$$

($\Leftarrow$) We then prove for the other direction. Suppose there exists an adversary $\mathcal{A}_{zk}$ that breaks zero-knowledge of NARG. We can then construct an environment $\mathcal{Z}$ that distinguishes between $\mathbf{HYB1}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{HYB2}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ using $\mathcal{A}_{zk}$ as subroutine. We let the environment $\mathcal{Z}$ handle queries from $\mathcal{A}_{zk}$ as follows.

- **On the query** x **to** $\mathcal{O}_{\mathsf{RO}}$: This is the same as in proof of Claim 4.1.
- **On the query** (x, v) **to** $\mathcal{O}_{\mathsf{Prog}}$: This is the same as in proof of Claim 4.1.
- **On the query** (x, w) **to** $\mathcal{O}_{\mathsf{Prv}}$: We let the environment $\mathcal{Z}$ activates the prover $\mathcal{P}_{\mathsf{pvr}}$ with input $\langle\mathtt{Prove} : [\mathsf{sid}, x, w]\rangle$ to generate an argument π. In $\mathbf{HYB1}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ using $\mathcal{A}_{zk}$, the argument is generated as $\pi \leftarrow \mathtt{Prv}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x, w)$. In $\mathbf{HYB2}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$, it is generated as $\pi \leftarrow \mathtt{Sim}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x)$. This process matches the way arguments are generated in $\mathrm{G}^{\mathrm{ZK}\text{-}0}_{\mathsf{NARG}}$ and $\mathrm{G}^{\mathrm{ZK}\text{-}1}_{\mathsf{NARG}}$ respectively. The generated argument π is then forwarded to $\mathcal{A}_{zk}$.

  Additionally, note that in $\mathbf{HYB1}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$, $\mathcal{G}_{\mathsf{RO}}$ is programmed according to the query-answer list pg produced by $\mathtt{Sim}$. This behavior aligns with $\mathrm{G}^{\mathrm{ZK}\text{-}1}_{\mathsf{NARG}}$. Any subsequent queries to $\mathcal{O}_{\mathsf{RO}}$ are then answered based on these programmed values.
- **On the query** (x, π) **to** $\mathcal{O}_{\mathsf{Vfy}}$: We let the environment $\mathcal{Z}$ activates the verifier $\mathcal{P}_{\mathsf{vfr}}$ with input $\langle\mathtt{Verify} : [\mathsf{sid}, x, \pi]\rangle$. In both $\mathbf{REAL}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{HYB1}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$, the verifier algorithm evaluates $b_v \xleftarrow{\mathsf{tr}_v} \mathtt{Vfy}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x, \pi)$. Also, in both worlds, if tr_v intersects with the points programmed by $\mathcal{Z}$, then $b = 0$ is returned. This is consistent with the behavior of $\mathcal{O}_{\mathsf{Vfy}}$. Thus we return the bit b output by $\mathcal{P}_{\mathsf{vfr}}$ as answer to $\mathcal{A}_{zk}$'s query.

Finally, the environment $\mathcal{Z}$ outputs the same bit returned by $\mathcal{A}_{zk}$. Consequently, we obtain the following result.

$$\mathbf{Adv}^{\mathrm{ZK}}_{\mathsf{NARG}}(\mathcal{A}_{zk}, \mathtt{Sim}) \leq \Delta_{\mathcal{Z}}\left(\mathbf{HYB1}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}, \mathbf{HYB2}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}\right). \tag{4.4}$$

□

Claim 4.3. $\mathbf{HYB2}^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ *and* $\mathbf{IDEAL}^{\mathcal{G}_{\mathsf{RO}}}_{\mathcal{F}_{\mathsf{zkSNARK}},\mathcal{S}_{\mathsf{NARG}},\mathcal{Z}}$ *are indistinguishable if and only if* NARG *satisfies* s*SIM-EXT.*

Proof. ($\Rightarrow$) We then show the direction where, if there exists an environment $\mathcal{Z}$ that distinguishes between these two worlds, we can then use it to construct an adversary $\mathcal{A}_{se}$ that breaks sSIM-EXT security of NARG. We then let $\mathcal{A}_{se}$ simulate the interaction between $\mathcal{Z}$ and these two worlds based on the input $\mathcal{Z}$ gives to each party.

- **On the input** $\langle\mathtt{Query} : [x]\rangle$ **to** $\mathcal{G}_{\mathsf{RO}}$: This is the same as in proof of Claim 4.1.

- **On the input** $\langle \texttt{ProgRO} : [x, v]\rangle$ **to** $\mathcal{G}_{\mathsf{RO}}$: This is the same as in proof of Claim 4.1.
- **On the input** $\langle \texttt{Prove} : [\mathsf{sid}, x, w]\rangle$ **to an honest prover** $\mathcal{P}_{\mathsf{pvr}}$: We let $\mathcal{A}_{se}$ query its oracle $\mathcal{O}_{\mathsf{Sim}}$ with this instance-witness pair (x, w). Observe that an argument is generated as $\pi \leftarrow \texttt{Sim}^{\mathcal{O}_{\mathsf{RO}}[\mathsf{pg}_{all}]}(x)$ where pg_{all} includes the points programmed by both $\mathcal{A}_{se}$ and $\texttt{Sim}$. Also note that the query-answer list pg_s by $\texttt{Sim}$ has also been added pg_{all}, which then affects the next query to $\mathcal{O}_{\mathsf{RO}}$. This simulates the behavior in both worlds where the prover $\mathcal{P}_{\mathsf{pvr}}$ is activated to prove (x, w). Thus we then write π from $\mathcal{O}_{\mathsf{Sim}}$ as the output of the honest prover $\mathcal{P}_{\mathsf{pvr}}$.
- **On the input** $\langle \texttt{Verify} : [\mathsf{sid}, x, \pi]\rangle$ **to an honest prover** $\mathcal{P}_{\mathsf{pvr}}$: We then let $\mathcal{A}_{se}$ pass this (x, π) to game $\mathrm{G}^{s\text{SIM-EXT}}_{\mathsf{NARG}}$.

Observe that for the environment $\mathcal{Z}$ to distinguish between these two worlds. It needs to provide the verifier $\mathcal{P}_{\mathsf{vfr}}$ an instance-argument pair (x, π) such that the following condition holds. In the first case, it has $b = 1$ for $b \leftarrow \texttt{Vfy}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x, w)$. Second, it has that $(x, w) \notin R$ for $w \leftarrow \texttt{Ext}(x, \pi, \mathsf{tr}_a)$ where tr_a is the query made by the environment. Third, it has that the query trace tr_v made by $\texttt{Vfy}$ to $\mathcal{G}_{\mathsf{RO}}$ does not intersect with the points programmed by $\mathcal{Z}$. This is exactly the winning condition of $\mathrm{G}^{s\text{SIM-EXT}}_{\mathsf{NARG}}[(\mathcal{A}_{se}, \texttt{Sim})]$. Thus we have that

$$\Delta_{\mathcal{Z}}(\mathbf{HYB}1^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}, \mathbf{IDEAL}^{\mathcal{G}_{\mathsf{RO}}}_{\mathcal{F}_{\mathsf{zkSNARK}},\mathcal{S},\mathcal{Z}}) \leq \mathbf{Adv}^{\text{sSIM-EXT}}_{\mathsf{NARG}}(\mathcal{A}_{se}, \texttt{Sim}). \tag{4.5}$$

($\Leftarrow$) We begin by showing that if there exists an adversary $\mathcal{A}_{se}$ that breaks the sSIM-EXT security of NARG, then it is possible to construct an environment $\mathcal{Z}$ capable of distinguishing between $\mathbf{HYB}1^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{IDEAL}^{\mathcal{G}_{\mathsf{RO}}}_{\mathcal{F}_{\mathsf{zkSNARK}},\mathcal{S},\mathcal{Z}}$. Specifically, we let $\mathcal{Z}$ handle queries from $\mathcal{A}_{se}$ as follows.

- **On the query** x **to** $\mathcal{O}_{\mathsf{RO}}$: The same as in proof of Claim 4.1.
- **On the query** (x, v) **to** $\mathcal{O}_{\mathsf{Prog}}$: The same as in proof of Claim 4.1.
- **On the query** (x, w) **to** $\mathcal{O}_{\mathsf{Sim}}$: For each query (x, w) made by $\mathcal{A}_{ext}$ to its oracle $\mathcal{O}_{\mathsf{Sim}}$, the environment $\mathcal{Z}$ activates the prover $\mathcal{P}_{\mathsf{pvr}}$ with the input $\langle \texttt{Prove} : [\mathsf{sid}, x, w]\rangle$. In both worlds, the argument π is generated as $\pi \leftarrow \texttt{Sim}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x)$. Additionally, the query-answer list pg_s output by $\texttt{Sim}$ has also been programmed according to the definition of these two worlds. This behavior aligns with that of $\mathcal{O}_{\mathsf{Sim}}$, and hence $\mathcal{Z}$ answers $\mathcal{A}_{ext}$'s query with π.

At the conclusion of its interactions, $\mathcal{A}_{ext}$ outputs an instance-argument pair (x, π). The environment $\mathcal{Z}$ then activates the verifier $\mathcal{P}_{\mathsf{vfr}}$ with the input $\langle \texttt{Verify} : [\mathsf{sid}, x, \pi]\rangle$. In both $\mathbf{HYB}1^{\mathcal{G}_{\mathsf{RO}}}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}$ and $\mathbf{IDEAL}^{\mathcal{G}_{\mathsf{RO}}}_{\mathcal{F}_{\mathsf{zkSNARK}},\mathcal{S}_{\mathsf{NARG}},\mathcal{Z}}$, the verifier $\mathcal{P}_{\mathsf{vfr}}$ outputs a bit $b \leftarrow \texttt{Vfy}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}(x, w)$. By the winning condition of $\mathrm{G}^{s\text{SIM-EXT}}_{\mathsf{NARG}}$, the pair (x, w) yields that $b = 1$. Moreover, in both worlds, we verify that the query trace tr_v made by $\texttt{Vfy}$ does not intersect any point programmed by $\mathcal{Z}$, consistent with the winning condition of $\mathrm{G}^{s\text{SIM-EXT}}_{\mathsf{NARG}}$. Otherwise, in both worlds, we return 0, resulting in no distinguishing advantage.

Next, we examine the remaining winning conditions of $\mathrm{G}^{s\text{SIM-EXT}}_{\mathsf{NARG}}$. In $\mathbf{IDEAL}^{\mathcal{G}_{\mathsf{RO}}}_{\mathcal{F}_{\mathsf{zkSNARK}},\mathcal{S},\mathcal{Z}}$, the query trace tr made by $\mathcal{Z}$ (excluding points programmed by $\mathcal{Z}$) during the challenge session sid is extracted and passed to the extractor $\texttt{Ext}$ along with (x, w). According to the winning condition of $\mathrm{G}^{s\text{SIM-EXT}}_{\mathsf{NARG}}$, the extractor $\texttt{Ext}$ outputs a witness w such that $(x, w) \notin R$. Furthermore, it holds that $x \notin Q$, which corresponds to $x \notin \mathsf{Inst}[\mathsf{sid}]$ in $\mathcal{F}_{\mathsf{zkSNARK}}$.

Therefore, for this (x, w), the prover $\mathcal{P}_{\mathsf{vfr}}$ outputs $\perp$, indicating failure. In this case, the environment $\mathcal{Z}$ can distinguish between the two worlds based on the output $\perp$. Hence, we conclude that

$$\mathbf{Adv}_{\mathsf{NARG}}^{\text{sSIM-EXT}}(\mathcal{A}, \mathtt{Sim}) \leq \Delta_{\mathcal{Z}} \left(\textsf{\textbf{HYB}}^{1}{}_{\Pi_{\mathsf{NARG}},\mathcal{A},\mathcal{Z}}^{\mathcal{G}_{\mathsf{RO}}}, \textsf{\textbf{IDEAL}}_{\mathcal{F}_{\mathsf{zkSNARK}},\mathcal{S}_{\mathsf{NARG}},\mathcal{Z}}^{\mathcal{G}_{\mathsf{RO}}} \right) \tag{4.6}$$

□

By combining Claims 4.1 to 4.3, we can then conclude the proof for the theorem. □

We note that our approach to handling $\mathcal{Z}$'s queries to $\mathcal{G}_{\mathsf{RO}}$ in proof of Theorem 4 is necessary for constructing the reduction. Specifically, if we parse x as (sid, x') but simply query $\mathcal{O}_{\mathsf{RO}}$ with x', then for any environment queries of the form (sid, x') and (sid', x') with $\mathsf{sid} \neq \mathsf{sid}'$, the oracle $\mathcal{O}_{\mathsf{RO}}$ would return the same response. This contradicts the purpose of a domain-separated oracle, where queries of the form (sid, x) should be independent of those of the form (sid', x) whenever $\mathsf{sid} \neq \mathsf{sid}'$.

Chapter 5

Composition for On-the-Fly MPC

López-Alt et al. [LTV12] introduced *on-the-fly multi-party computation* (MPC), where a computationally powerful but untrusted cloud is outsourced to perform arbitrary computations on data from dynamically joining users, without requiring direct interaction among clients. Their proposed protocol is a *3-round* scheme: each client encrypts their data using a MKHE scheme, the server evaluates a circuit over the ciphertexts, and the involved clients collaboratively decrypt the result.

Subsequent works [MW16, AJJM20] sought to reduce the number of computation rounds by introducing *partial decryption*, where individual parties use their own keys, and the final output is reconstructed from these partial decryptions.

As Kwak et al. [KLSW24] highlighted, MKHE faces scalability issue as the number of clients increases. To address this limitation, in this chapter, we revisit the construction of on-the-fly MPC that is secure against malicious adversary through the composition of MGHE and zkSNARK.

5.1 System Overview

Model of Computation. In our system, we consider n clients $\{\mathcal{P}_{\mathsf{cli}_i}\}_{i\in[n]}$ and a server $\mathcal{P}_{\mathsf{srv}}$ that assists computation. Following the work by Smart [Sma23], we categorize the participants into three groups. For one execution of the protocol to evaluate a circuit f, we define the input set as $\mathbb{I} = \{\{\mathcal{P}_{\mathsf{cli}_i}\}_{i\in I}, \mathcal{P}_{\mathsf{srv}}\}$ with $I \subseteq [n]$, which means a subset of clients contributes inputs $(x_1, \dots, x_\ell)$ for the computation, and the server provides the circuit f. The computation party is the server $\mathbb{C} = \{\mathcal{P}_{\mathsf{srv}}\}$, who performs the computation $\hat{x} = f(x_1, \dots, x_\ell)$. The output set is defined as a set of clients $\mathbb{O} = \{\mathcal{P}_{\mathsf{cli}_i}\}_{i\in I'}$. We note that it may have $I \neq I'$ since a client may simply provide its input (e.g., by encrypting with the public key of another client) for the computation whose result is for another client.

Thus, we divide an execution of the protocol into three phases: *uploading data*, *evaluation*, and *retrieving results*. Specifically, a computation task may be proposed (either by a client or by the server). Subsequently, some clients provide inputs for this computation task. At a stage, the server evaluates the result based on the current inputs, after which the participating clients retrieve the result. We assume that these three phases can be executed concurrently. Specifically, a computation task can be proposed at any time; however, only inputs received before the evaluation phase will be considered. Inputs provided after this time may be used in the next computation task with another circuit.

It is assumed that all the n clients have been initialized before any computation; however, participation in a specific computation is optional. In practice, this model aligns with scenarios in which organizations subscribe to a cloud computing service and decide dynamically whether to engage in collaborative machine learning based on the relevance of the task at hand.

Remark 22 (Application of the Computation Model)**.** We observe that this mode of computation serves as a generalized framework for numerous real-world applications involving outsourced computations [JNO16, SVdV16, TC16]. In the special case of a single server and a single client, this model reduces to standard computation outsourcing, which has been analyzed with the UC framework by Beskorovajnov et al. [BEJMQ25]. Notable applications include collaborative machine learning [JKLS18, PTH21], private set intersection (PSI) [ATD16, ATMD18, ADMT22], and privacy-preserving database queries [YZW06, OG10].

Adversarial Model. Similarly as in Section 3.3.3.3, we assume the clients and the server are subject to *static corruption*. However, here we assume adversary to be *malicious* meaning that its behavior is not restricted and it actively attack the execution of the protocol (also known as an *active* adversary in the work by Cramer and Damgård [CD05]). We also assume the adversary to be *rushing* where it adaptively selects the message for the corrupted parties by observing the message from honest parties.

5.2 Message Buffer Functionality $\mathcal{F}_{\mathsf{MsgBuf}}$

Considering that evaluation is only possible after clients have uploaded sufficient data, and a client can only reconstruct the message after obtaining enough partial decryptions, a "buffer" is needed to allow the server and clients to wait for data and partial decryptions to be uploaded. Inspired previous works on UC-secure MPC [CCL15, LZ21], we introduce a functionality $\mathcal{F}_{\mathsf{MsgBuf}}$ to handle message transmission.

We let the functionality $\mathcal{F}_{\mathsf{MsgBuf}}$ internally maintain two lookup tables: $\mathsf{MsgPool}$ and ShPool. The table $\mathsf{MsgPool}$ is indexed by $(\mathsf{grp}, \mathsf{sid})$, corresponding to data uploaded by a client on behalf of the group grp and the ciphertext evaluated by the server for grp. Give that multiple message may be uploaded to the same index $(\mathsf{grp}, \mathsf{sid})$, we use a unique message identifier uid to specify which message is being referenced.

When a party $\mathcal{P}_{\mathsf{pid}}$ (with $\mathsf{pid} \in \{\mathsf{cli}, \mathsf{srv}\}$) or the adversary $\mathcal{S}$ (acting on behalf of a corrupted party) sends an input of the form $\langle \texttt{RegMsg} : [\mathsf{msg}] \rangle$, the functionality parses msg as

$$\{\texttt{"}\cdot\texttt{"} : [\mathsf{pid}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (\cdot)]\},$$

where pid denotes the party uploading the message. If pid corresponds to an uncorrupted party, or to a corrupted party with the adversary $\mathcal{S}$ providing the input, the message is added to $\mathsf{MsgPool}$ under the index $(\mathsf{grp}, \mathsf{sid})$. Similarly, when a client $\mathcal{P}_{\mathsf{cli}}$ submits an input $\langle \texttt{RegSh} : [\mathsf{msg}] \rangle$, the functionality parses msg as

$$\{\texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (\cdot)]\}$$

and records the corresponding partial decryption in $\mathsf{ShPool}[\mathsf{grp}, \mathsf{sid}; \mathsf{uid}]$.

To retrieve messages, a party $\mathcal{P}_{\mathsf{pid}}$ or the adversary $\mathcal{S}$ may submit $\langle \texttt{RetMsg} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle$. The functionality responds by extracting the message at $\mathsf{MsgPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}]$ and delivering it to the requester. For partial decryption retrieval, the input $\langle \texttt{RetSh} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle$ may be sent by $\mathcal{P}_{\mathsf{pid}}$ or $\mathcal{S}$, prompting the functionality to return all available partial decryptions stored at $\mathsf{ShPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}]$. We denote the set of these partial decryptions by shs and return shs to the requester.

The functionality $\mathcal{F}_{\mathsf{MsgBuf}}$ can be viewed as an authenticated channel plus a lookup table. Intuitively, this can be thought of as each party maintaining a local lookup table to track received

Functionality: $\mathcal{F}_{\mathsf{MsgBuf}}$

Participants: n clients $\mathcal{P}_{\mathsf{cli}_1}, \mathcal{P}_{\mathsf{cli}_2}, \dots, \mathcal{P}_{\mathsf{cli}_n}$. At a state, a subset of clients $\{\mathcal{P}_{\mathsf{cli}_i}\}_{i \in I}$ with $I \subseteq [n]$ form ℓ groups $\mathcal{P}_{\mathsf{grp}_j}$ with $\mathsf{grp}_j = \{\mathsf{cli}_i\}_{i \in I_j}$ for $j \in [\ell]$. A server $\mathcal{P}_{\mathsf{srv}}$, and an ideal-process adversary $\mathcal{S}$.

Initialization

```
$\mathsf{MsgPool} := [\,]$ // Message registry indexed by $(\mathsf{grp}, \mathsf{sid})$ with each entry as a list
$\mathsf{ShPool} := [\,]$
```

Corruption

```
**In** $\langle \texttt{Corrupt} : [\mathsf{pid}] \rangle \leftarrow \mathcal{S}$
    $\mathsf{Corrupt} := \mathsf{Corrupt} \cup \{\mathsf{pid}\}$
```

Message Registration

```
**In** $\langle \texttt{RegMsg} : [\mathsf{msg}] \rangle \leftarrow \{\mathcal{P}_{\mathsf{pid}}, \mathcal{S}\}$
    $\{\texttt{"}\cdot\texttt{"} : [\mathsf{pid}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (\cdot)]\} := \mathsf{msg}$
    **if** $\mathsf{pid} \notin \mathsf{Corrupt} \wedge \langle \texttt{RegMsg} : [\cdot] \rangle \leftarrow \mathcal{S}$ **then**
        $\texttt{Abort}$
    **if** $\mathsf{MsgPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}] \neq \bot$ **then**
        $\texttt{Abort}$
    **else**
        $\mathsf{MsgPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}] := \mathsf{msg}$
```

```
**In** $\langle \texttt{RegSh} : [\mathsf{msg}] \rangle \leftarrow \{\mathcal{P}_{\mathsf{cli}}, \mathcal{S}\}$
    $\{\texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (\cdot)]\} := \mathsf{msg}$
    **if** $\mathsf{cli} \notin \mathsf{Corrupt} \wedge \langle \texttt{RegMsg} : [\cdot] \rangle \leftarrow \mathcal{S}$ **then**
        $\texttt{Abort}$
    $\mathsf{ShPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}] := \mathsf{ShPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}] \bowtie \mathsf{msg}$
```

Message Retrieval

```
**In** $\langle \texttt{RetMsg} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle \leftarrow \{\mathcal{P}_{\mathsf{pid}}, \mathcal{S}\}$
    $\mathsf{msg} := \mathsf{MsgPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}]$
    **Out** $\mathsf{msg} \rightarrow \{\mathcal{P}_{\mathsf{pid}}, \mathcal{S}\}$
```

```
**In** $\langle \texttt{RetSh} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle \leftarrow \{\mathcal{P}_{\mathsf{pid}}, \mathcal{S}\}$
    $\mathsf{shs} := \mathsf{ShPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid})]$
    **Out** $\mathsf{shs} \rightarrow \{\mathcal{P}_{\mathsf{pid}}, \mathcal{S}\}$
```

Figure 5.1: Ideal functionality $\mathcal{F}_{\mathsf{MsgBuf}}$ of a message buffer.

messages and partial decryptions, with all communications occurring over an authenticated channel. If the sender is corrupted, the receiver obtains the message from the adversary instead. For a formal definition of an authenticated channel, we refer to $\mathcal{F}_{\mathsf{AUTH}}$ as defined by Canetti [Can03].

5.3 Ideal Functionality $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$

We present the ideal functionality $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ for an on-the-fly MPC protocol in a multi-group setting, as detailed in Figures 5.2 to 5.6. At a high level, this functionality operates as a "registry service", allowing each client and the server to upload data or evaluation under the name of a group grp. The security properties are modeled by controlling when and how $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ may release specific content to the associated parties and the adversary.

Initialization and Corruption. Before detailing the functionality of each phase in the system, in Figure 5.2, we similarly first describe the internal variables used by $\mathcal{F}_{\mathsf{MGHE}}$, and handling of corruption. The following variables are defined, and their visibility is restricted exclusively to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$.

– *Message Registry* $\mathsf{MsgPool}$*:* A lookup table indexed by $(\mathsf{grp}, \mathsf{sid})$, where each entry is a list (initially empty). We use $\mathsf{MsgPool}$ to track the messages registered for a group grp during a session sid.

– *Corrupted Party Set* $\mathsf{Corrupt}$: A set (initialized as $\varnothing$) that tracks parties covertly corrupted by the adversary.

As before, we consider pid-wise corruption. The adversary $\mathcal{S}$ can send a backdoor input $\langle\texttt{Corrupt} : [\mathsf{pid}]\rangle$ to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ to corrupt a party $\mathcal{P}_{\mathsf{pid}}$, where $\mathsf{pid} \in \{\mathsf{cli}, \mathsf{srv}\}$ specifies the target as either a client or the server. Then pid is added to $\mathsf{Corrupt}$ for further reference.

Note that we consider only *static corruption* of parties. In this case, after a party $\mathcal{P}_{\mathsf{pid}}$ is corrupted, all messages intended for it are instead delivered to $\mathcal{S}$. Since corruption of clients take place before any computation, no messages are output to $\mathcal{P}_{\mathsf{cli}}$ prior to its corruption. To simplify, this behavior is omitted from the corruption description in the code.

Functionality: $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ (Part I)

Participants: n clients $\mathcal{P}_{\mathsf{cli}_1}, \mathcal{P}_{\mathsf{cli}_2}, \ldots, \mathcal{P}_{\mathsf{cli}_n}$. At a state, a subset of clients $\{\mathcal{P}_{\mathsf{cli}_i}\}_{i\in I}$ with $I \subseteq [n]$ form k groups $\mathcal{P}_{\mathsf{grp}_j}$ with $\mathsf{grp}_j = \{\mathsf{cli}_i\}_{i\in I_k}$ for $j \in [k]$. A server $\mathcal{P}_{\mathsf{srv}}$, and an ideal-process adversary $\mathcal{S}$.

Initialization

```
1 : $\mathsf{MsgPool} := [\,]$ // Message registry indexed by $(\mathsf{grp}, \mathsf{sid})$ with each entry as a list
2 : $\mathsf{Corrupt} := \varnothing$ // Set of corrupted party.
```

Corruption

```
1 : In $\langle\texttt{Corrupt} : [\mathsf{pid}]\rangle \leftarrow \mathcal{S}$
2 :     $\mathsf{Corrupt} := \mathsf{Corrupt} \cup \{\mathsf{pid}\}$
```

Figure 5.2: Ideal functionality $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ (Part I – Initialization and Corruption) of an on-the-fly MPC protocol.

Data Uploading. In Figure 5.3, we describe the first phase where data are uploaded by clients. Based on whether $\mathcal{P}_{\mathsf{cli}}$ has been corrupted or not, we consider the following two cases.

– $\mathcal{P}_{\mathsf{cli}}$ *is not corrupted*: In this case, we let the client $\mathcal{P}_{\mathsf{cli}}$ send to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ an input

$$\langle\texttt{Upload} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m]\rangle$$

to upload data m to $\mathsf{MsgPool}$ under the name of a group $\mathcal{P}_{\mathsf{grp}}$ for a session sid for the later evaluation. Then $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ generates a message that is identified by uid of the format

$$\{\texttt{"up"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m]\}$$

Functionality: $\mathcal{F}_{\text{OtF-MPC}}$ (Part II)

Data Uploading

```
//The client is not corrupted.
In ⟨Upload : [cli, grp, sid, uid, $m$]⟩ ↞ $\mathcal{P}_{\text{cli}}$
    if cli $\in$ Corrupt then
        Abort
    if $|$Corrupt $\cap$ grp$| \geq |$grp$|/t$
        Out {"up" : [cli, grp, sid, uid, $m$]} ⇾ $\mathcal{S}$
    else
        Out {"up" : [cli, grp, sid, uid, $|m|$]} ⇾ $\mathcal{S}$
    msg := {"up" : [cli, grp, sid, uid, $m$]}
    if MsgPool[(grp, sid); uid] $\neq \perp$ then
        Abort
    MsgPool[(grp, sid); uid] := msg
//The client is corrupted.
In {"up" : [cli, grp$^*$, sid$^*$, uid$^*$, $m^*$]} ↞ $\mathcal{S}$
    if cli $\notin$ Corrupt then
        Abort
    msg := {"up" : [cli, grp$^*$, sid$^*$, uid$^*$, $m^*$]}
    if MsgPool[(grp$^*$, sid$^*$); uid$^*$] $\neq \perp$ then
        Abort
    MsgPool[(grp$^*$, sid$^*$); uid$^*$] := msg
```

Figure 5.3: Ideal functionality $\mathcal{F}_{\text{OtF-MPC}}$ (Part II – Data Uploading) of an on-the-fly MPC protocol.

where cli identifies the client who uploads the data m, and grp specifies the group name under which the data is uploaded. Then based on the corruption state of grp, there are two possible cases:

- *At least $\frac{|\text{grp}|}{t}$ clients are corrupted*: In this case, the adversary has corrupted enough clients in the group to decrypt for the secret. Thus we let $\mathcal{F}_{\text{OtF-MPC}}$ output the message {"up" : [cli, grp, sid, m]} containing the actual content m to the adversary $\mathcal{S}$.
- *Fewer than $\frac{|\text{grp}|}{t}$ clients are corrupted*: In this case, the adversary does not control enough clients to decrypt the message. Thus it should only learn the message length $|m|$. Thus we let $\mathcal{F}_{\text{OtF-MPC}}$ output the message length $|m|$ to $\mathcal{S}$.

This models the security that, when a client $\mathcal{P}_{\text{cli}}$ is not corrupted and the corruption threshold has not been met, then the adversary $\mathcal{S}$ does not learn the content of message uploaded by the client (*client-side confidentiality*).

In any case, the original message from $\mathcal{P}_{\text{cli}}$ is registered in the message registry MsgPool. This models the security that, when a client $\mathcal{P}_{\text{cli}}$ is not corrupted, regardless of corruption state of the group, the adversary can only relay the message without changing its content (*client-side integrity*).

- $\mathcal{P}_{\mathsf{cli}}$ *is corrupted*: In this case, $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ no longer takes input from the client $\mathcal{P}_{\mathsf{cli}}$ since input from it has been controlled by $\mathcal{S}$. Instead, $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ now directly takes a different message
$$\{\texttt{"up"} : [\mathsf{cli}, \mathsf{grp}^*, \mathsf{sid}^*, \mathsf{uid}^*, m^*]\}$$
from the adversary $\mathcal{S}$ as if this message were sent by $\mathcal{P}_{\mathsf{cli}}$. This message with m^* is instead registered in $\mathsf{MsgPool}$. The original data from $\mathcal{P}_{\mathsf{cli}}$ is then ignored by $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$.

Regardless of whether $\mathcal{P}_{\mathsf{cli}}$ is corrupted or not, the data is always included in $\mathsf{MsgPool}$. This is because $\mathsf{MsgPool}$ is an abstract construct rather than an entity capable of deciding whether to accept or reject a message. Instead, it represents the "communication layer" between parties, leaving the decision-making process to the parties themselves. One can think of $\mathsf{MsgPool}$ as a buffer that collects messages from various parties and messages can be retrieved from the buffer by parties.

Circuit Evaluation. In Figures 5.4 and 5.5, we describe the second phase where circuits are evaluated. Depending on the corruption state of a server $\mathcal{P}_{\mathsf{srv}}$, we consider the following two cases.

- $\mathcal{P}_{\mathsf{srv}}$ *is not corrupted* (Figure 5.4): In this case, we let the honest server $\mathcal{P}_{\mathsf{srv}}$ send an input
$$\left\langle \texttt{Evaluate} : \left[\mathsf{srv}, \mathsf{sid}, \mathsf{uid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j\in[\ell]}\right]\right\rangle$$
to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ to evaluate a circuit f over the data stored in $\mathsf{MsgPool}$. Note that the server $\mathcal{P}_{\mathsf{srv}}$ does not have direct access to the data provided by the clients. Instead, it submits tuples $(\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)$ for $j \in [\ell]$, which $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ uses to locate the data associated with each group grp_j in $\mathsf{MsgPool}$. If there is no such a message at the location, we let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ abort on this input. We assume that all messages to be evaluated on must be in the same session and the result must be registered for the same session i.e., $\forall i, j \in [\ell], \mathsf{sid}_i = \mathsf{sid}_j = \mathsf{sid}$. If there is any out of session messages, then we let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ abort on this input.

  For $j \in [\ell]$, we consider two types of messages that can be retrieved from the message registry $\mathsf{MsgPool}$.

  (1) The message originates from a client $\mathcal{P}_{\mathsf{cli}}$ and follows the format
$$\left\{\texttt{"up"} : \left[\mathsf{cli}, \mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j, m_j\right]\right\}.$$
  We let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ verify whether m is a valid message by checking that $m \neq \bot$. Notably, $\mathcal{S}$ may inject a message directly (on behalf of a corrupted client) instead of invoking $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$, which can lead to cases where $m = \bot$. To handle simulation failures in the security proof, we let $\mathcal{S}$ inject a message with $m = \bot$. A detailed discussion of this behavior will be provided when formally defining $\mathcal{S}$.

  (2) The message comes from a server $\mathcal{P}_{\mathsf{srv}}$ as the result of a previous computation [1] and is of the format
$$\left\{\texttt{"eval"} : \left[\mathsf{srv}, \mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j, f_j, m_j, (\mathsf{grp}_{j'}, \mathsf{sid}_{j'}, \mathsf{uid}_{j'})_{j'\in[\ell']}\right]\right\},$$

[1]We remark that this check enables us to consider a setting with multiple servers, where the output of one server serves as the input to another circuit evaluated by a different server. However, this check remains necessary even in the single-server case, as we can assume that the server "forgets" the result after evaluation and simply treats it as a received message.

Functionality: $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ (Part III - a)

Evaluation

```
// The server is not corrupted
In $\langle \texttt{Evaluate} : [\mathsf{srv}, \mathsf{sid}, \mathsf{uid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j\in[\ell]}] \rangle \leftarrow \mathcal{P}_{\mathsf{srv}}$
    if $\mathsf{srv} \in \mathsf{Corrupt}$ then
        Abort
    if $\exists i, j \in [\ell] : (\mathsf{sid}_i \neq \mathsf{sid}_j \vee \mathsf{sid}_j \neq \mathsf{sid})$ then
        Abort
    if $\exists\, j \in [\ell] : \mathsf{MsgPool}[(\mathsf{grp}_j, \mathsf{sid}_j); \mathsf{uid}_j] = \perp$ then
        Abort
    for $j \in [\ell]$ do
        $\mathsf{msg} := \mathsf{MsgPool}[(\mathsf{grp}_j, \mathsf{sid}_j); \mathsf{uid}_j]$
        // Abort if data is not well-formed.
        if $\mathsf{msg} = \{\texttt{"up"} : [\mathsf{cli}, \mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j, m_j]\}$ then
            if $m_j = \perp$ then
                Abort
        // Abort if result is incorrectly evaluated; for multi-hop
        elseif $\mathsf{msg} = \{\texttt{"eval"} : [\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j, f_j, m_j, (\mathsf{grp}_{j'}, \mathsf{sid}_{j'}, \mathsf{uid}_{j'})_{j'\in[\ell']}]\}$
            for $j' \in [\ell']$ do
                $\{\texttt{"}\cdot\texttt{"} : [\cdots, m_{j'}\} := \mathsf{MsgPool}[(\mathsf{grp}_{j'}, \mathsf{sid}_{j'}); \mathsf{uid}_{j'}]$
            if $m_j \neq f_j(\{m_{j'}\}_{j'\in[\ell']}) \vee \exists i', j' \in [\ell'] : \mathsf{sid}_{i'} \neq \mathsf{sid}_{j'} \vee \mathsf{sid}_{j'} \neq \mathsf{sid}_j$ then
                Abort
        else
            Abort
    $\hat{m} := f(m_1, \ldots, m_\ell)$
    $\hat{\mathsf{grp}} := \cup_{j\in[\ell]} \mathsf{grp}_j$
    Out $\{\texttt{"eval"} : [\mathsf{srv}, \hat{\mathsf{grp}}, \mathsf{sid}, \mathsf{uid}, \sqcup, \hat{m}, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j\in[\ell]}]\} \rightarrow \mathcal{S}$
    $\mathsf{msg} := \{\texttt{"eval"} : [\mathsf{srv}, \hat{\mathsf{grp}}, \mathsf{sid}, \mathsf{uid}, f, \hat{m}, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j\in[\ell]}]\}$
    if $\mathsf{MsgPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}] \neq \perp$ then
        Abort
    $\mathsf{MsgPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}] := \mathsf{msg}$
```

Figure 5.4: Ideal functionality $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ (Part III.a – Hoenst Evaluation) of an on-the-fly MPC protocol. The part highlighted in blue indicates the behavior of $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ when multi-hop evaluation is involved.

where $(\mathsf{grp}_{j'}, \mathsf{sid}_{j'}, \mathsf{uid}_{j'})_{j'\in[\ell']}$ indicates the data used in the previous computation. In this case, we extract each $m_{j'}$ from MsgPool and verifies that $m_j = f_j(\{m_{j'}\}_{j'\in[\ell']})$. We remark that if there is any $m_{j'} = \perp$, then the check fails automatically. We also let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ check that all the messages are in the same session. If not, then we let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ abort on this input.

If checks pass for all m_j's extracted from MsgPool, we then let $\hat{m} = f(m_1, \ldots, m_\ell)$ be the evaluation of f over these inputs m_j. and let $\hat{\mathsf{grp}} = \cup_{j\in[\ell]} \mathsf{grp}_j$ represent the group of clients whose data were used in this evaluation. Then $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ generates a message

$$\left\{\texttt{"eval"} : \left[\mathsf{srv}, \hat{\mathsf{grp}}, \mathsf{sid}, \mathsf{uid}, f, \hat{m}, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j\in[\ell]}\right]\right\}$$

We then let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ disclose the result $\hat{m}$ to $\mathcal{S}$. This is because we assume that the client always outputs its partial decryptions (through an authenticated channel) for any message that is the server's evaluation output, allowing $\mathcal{S}$ to observe these partial decryptions and reconstruct the evaluation. Indeed, the confidentiality of the result is *not* a requirement of an MPC protocol i.e., only the inputs provided by each party must remain private.

However, observe that regardless of the corruption state of $\hat{\mathsf{grp}}$, the circuit f remains hidden (we use $\sqcup$ as a placeholder for that). Additionally, the message containing the correct result $\hat{m} = f(m_1, \ldots, m_\ell)$ is always registered in MsgPool.

This models the security guarantees that, when the server $\mathcal{P}_{\mathsf{srv}}$ is not corrupted, the adversary $\mathcal{S}$ learns nothing about the circuit used for evaluation (*server-side confidentiality*), and can only relay the result honestly (*server-side integrity*). The confidentiality of data used for evaluation or the result depends on the corruption state of the aggregated group $\hat{\mathsf{grp}}$ (*client-side confidentiality*).

– $\mathcal{P}_{\mathsf{srv}}$ *is corrupted* (Figure 5.5) : In this case, $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ no longer takes an input from the server $\mathcal{P}_{\mathsf{srv}}$. Instead, it takes a message

$$\left\{\texttt{"eval"} : \left[\mathsf{srv}, \hat{\mathsf{grp}}^*, \mathsf{sid}^*, \mathsf{uid}^*, f^*, \hat{m}^*, (\mathsf{grp}^*_j, \mathsf{sid}^*_j, \mathsf{uid}^*_j)_{j\in[\ell^*]}\right]\right\}$$

directly from $\mathcal{S}$ to retrieve any messages $\{m^*_j\}_{j\in[\ell^*]}$ for evaluation, regardless of whether these messages are in a different session or it is invalid $m^*_j = \perp$. The adversary can also arbitrarily choose the circuit f^* and the resulting output $\hat{m}^*$. As before, we may have $\mathcal{S}$ directly inject $\hat{m}^* = \perp$ when there is a simulation failure in the security proof.

If the adversary does not explicitly inject a result (i.e., $\hat{m}^* = \sqcup$), we let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ perform the evaluation itself to get $\hat{m}^* = f^*(\{m^*_j\}_{j\in[\ell^*]})$. We remark that if $m^*_j = \perp$ for any $j \in [\ell^*]$, then we set $\hat{m}^* = \perp$. Then similarly depending on whether there is a client in $\hat{\mathsf{grp}}^*$ has been corrupted, we let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ output $\hat{m}^*$ to $\mathcal{S}$. Finally, this message is then registered in MsgPool.

Remark 23. For simplicity in syntax, we assume that $\mathcal{P}_{\mathsf{srv}}$ selects inputs from MsgPool for evaluating f and determines the position of each input within the circuit, rather than requiring an additional communication to let clients specify this. In practice, however, this process can involve posting a "high-level" description of the circuit or computation task. Clients can then decide whether to participate in the computation, which data to contribute for a specific evaluation, and how their data should be plugged into the circuit (via group consensus).

Furthermore, the message includes details about which data were used for the evaluation. This allows verification of both the correctness of the result and whether the intended inputs

Functionality: $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ (Part III - b)

Evaluation

```
//The server is corrupted
In $\{\texttt{"eval"} : [\mathsf{srv}, \hat{\mathsf{grp}}^*, \mathsf{sid}^*, \mathsf{uid}^*, f^*, \hat{m}^*, (\mathsf{grp}^*_j, \mathsf{sid}^*_j, \mathsf{uid}^*_j)_{j \in [\ell^*]}]\} \leftarrow \mathcal{S}$
    if $\mathsf{srv} \notin \mathsf{Corrupt}$ then
        Abort
    if $\hat{m}^* = \sqcup$ then
        if $\exists j \in [\ell^*] : \mathsf{MsgPool}[(\mathsf{grp}^*_j, \mathsf{sid}^*_j); \mathsf{uid}^*_j] = \perp$ then
            Abort
        for $j \in [\ell^*]$ do
            $\{\texttt{" \cdot "} : [\cdots, m^*_j\} := \mathsf{MsgPool}[(\mathsf{grp}^*_j, \mathsf{sid}^*_j); \mathsf{uid}^*_j]$
        if $\exists j \in [\ell^*] : m^*_j = \perp$
            $\hat{m}^* := \perp$
        else
            $\hat{m}^* := f^*(\{m^*_j\}_{j \in [\ell^*]})$
        Out $\{\texttt{"eval"} : [\mathsf{srv}, \hat{\mathsf{grp}}^*, \mathsf{sid}^*, \mathsf{uid}^*, f^*, |\hat{m}^*|, (\mathsf{grp}^*_j, \mathsf{sid}^*_j, \mathsf{uid}^*_j)_{j \in [\ell^*]}]\} \dashrightarrow \mathcal{S}$
    $\mathsf{msg} := \{\texttt{"eval"} : [\mathsf{srv}, \hat{\mathsf{grp}}^*, \mathsf{sid}^*, \mathsf{uid}^*, f^*, \hat{m}^*, (\mathsf{grp}^*_j, \mathsf{sid}^*_j, \mathsf{uid}^*_j)_{j \in [\ell^*]}]\}$
    if $\mathsf{MsgPool}[(\mathsf{grp}^*, \mathsf{sid}^*); \mathsf{uid}^*] \neq \perp$ then
        Abort
    $\mathsf{MsgPool}[(\mathsf{grp}^*, \mathsf{sid}^*); \mathsf{uid}^*] := \mathsf{msg}$
```

Figure 5.5: Ideal functionality $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ (Part III.b – Malicious Evaluation) of an on-the-fly MPC protocol.

were used. In practice, if clients detect discrepancies between the inputs used and those agreed upon for the evaluation, they can simply abort the execution. For simplicity, this thesis omits discussions about such abortions since they are trivial.

Data Retrieval. In Figure 5.6, we describe the final phase, where data registered in MsgPool can be retrieved. A client $\mathcal{P}_{\mathsf{cli}}$ (or $\mathcal{S}$ in name of a corrupted client) can send to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ the input

$$\langle \texttt{Retrieve} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle$$

to request message with identifier uid stored in the lookup table MsgPool for a group grp in a session sid. This message could either be inputs uploaded by other clients in the group or results evaluated by the server. The response from $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ depends on the corruption status of $\mathcal{P}_{\mathsf{cli}}$.

– $\mathcal{P}_{\mathsf{cli}}$ *is not corrupted*: We first let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ verify that the requesting client is in the group grp. Otherwise, the execution is aborted. Then $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ verifies that the message in the message registry position $\mathsf{MsgPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}]$ is not uploaded by a client i.e., not in the format $\{\texttt{"up"} : [\cdot]\}$. If yes, then $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ abort. That is to model that the honest client should not attempt to retrieve the private input of clients nor the result for a group that it is not in.

Otherwise, $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ checks that the message from the server with result contained in the

Functionality: $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ (Part IV)

Data Retrieval

```
In $\langle\texttt{Retrieve} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}]\rangle \leftarrow \mathcal{P}_{\mathsf{cli}}$
  if $\mathsf{cli} \notin \mathsf{grp} \vee \mathsf{cli} \in \mathsf{Corrupt}$ then
    Abort
  if $\mathsf{MsgPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}] = \bot$ then
    Abort
  $\mathsf{msg} := \mathsf{MsgPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}]$
  if $\mathsf{msg} = \{\texttt{"up"} : [\mathsf{cli}', \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m]\}$ then
    Abort
  elseif $\mathsf{msg} = \{\texttt{"eval"} : [\mathsf{srv}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, f, \hat{m}, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j\in[\ell]}]\}$
    for $j \in [\ell]$ do
      $\{\texttt{"}\cdot\texttt{"} : [\cdots, m_j\} := \mathsf{MsgPool}[(\mathsf{grp}_j, \mathsf{sid}_j); \mathsf{uid}_j]$
    // Abort if result is incorrectly evaluated, or not in the same session.
    if $\hat{m} \neq f(\{m_j\}_{j\in[\ell]}) \vee \exists i, j \in [\ell] : \mathsf{sid}_i \neq \mathsf{sid}_j \vee \mathsf{sid}_j \neq \mathsf{sid})$
      Abort
    // Number of covertly corrupted parties exceed threshold
    // $\mathcal{S}$ controls enough clients to reconstruct wrong retrieval.
    if $|\mathsf{Corrupt} \cap \mathsf{grp}| \geq |\mathsf{grp}|/t$ then
      In $\{\texttt{"ret"} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m^*]\} \leftarrow \mathcal{S}$
      Out $\{\texttt{"ret"} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m^*]\} \rightarrow \mathcal{P}_{\mathsf{cli}}$
    else
      Out $\{\texttt{"ret"} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m]\} \rightarrow \mathcal{P}_{\mathsf{cli}}$
  else
    Abort
In $\langle\texttt{Retrieve} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}]\rangle \leftarrow \mathcal{S}$
  // The requesting client is corrupted
  if $\mathsf{cli} \notin \mathsf{Corrupt}$ then
    Abort
  if $\mathsf{MsgPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}] = \bot$ then
    Abort
  $\mathsf{msg} := \mathsf{MsgPool}[(\mathsf{grp}, \mathsf{sid}); \mathsf{uid}]$
  if $\mathsf{msg} = \{\texttt{"up"} : [\mathsf{cli}', \mathsf{grp}, \mathsf{sid}, m]\}$ then
    if $(|\mathsf{Corrupt} \cap \mathsf{grp}| \geq |\mathsf{grp}|/t)$ then
      Out $\{\texttt{"ret"} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m]\} \rightarrow \mathcal{S}$
    else
      Out $\{\texttt{"ret"} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}, |m|]\} \rightarrow \mathcal{S}$
  elseif $\mathsf{msg} = \{\texttt{"eval"} : [\mathsf{srv}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, f, \hat{m}, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j\in[\ell]}]\}$
    Out $\{\texttt{"ret"} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}, \hat{m}]\} \rightarrow \mathcal{S}$
  else
    Abort
```

Figure 5.6: Ideal functionality $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ (Part IV – Data Retrieval) of an on-the-fly MPC protocol.

message is correct, i.e., $\hat{m} = f(\{m_j\}_{j\in[\ell]})$ [2], and they are in the same session. If any of these fails, $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ then aborts the execution. Then depending on the number of corrupted clients in the group grp, We consider the following two cases.

- *At least* $\frac{|\mathsf{grp}|}{t}$ *clients are corrupted*: The adversary, having enough influence to modify partial decryptions, can generate and return an incorrect value m^*, which is then output to the client $\mathcal{P}_{\mathsf{cli}}$.
- *Fewer than* $\frac{|\mathsf{grp}|}{t}$ *clients are corrupted*: The adversary does not have sufficient influence to modify the result or reconstruct the message. Then original data m is encoded into a message of the format

  $$\{\texttt{"ret"} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m]\}$$

  and securely delivered to the honest client $\mathcal{P}_{\mathsf{cli}}$.

This models the security that the adversary cannot alter the reconstructed result if it controls fewer clients than the corruption threshold.

- $\mathcal{P}_{\mathsf{cli}}$ *is corrupted*: In this case, we let $\mathcal{S}$ send the input to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ on the behalf of the corrupted client $\mathcal{P}_{\mathsf{cli}}$. If the adversary controls at least $\frac{|\mathsf{grp}|}{t}$ clients in the group, it can reconstruct the requested message regardless of the message is of the client or the server. Thus we let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ return the actual content m to $\mathcal{S}$. Otherwise, we let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ just return the length $|m|$ to it. If the message is from the server, then the adversary can eventually learn it through the corrupted party by combing the partial decryptions received from other honest clients (as discussed earlier, we assume honest clients always output partial decryption for an evaluation output by the server through an authenticated channel). Thus we let $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ output $\hat{m}$ to $\mathcal{S}$ in this case.

  This models the security that if the adversary corrupts fewer clients in a group than the corruption threshold, it cannot learn the secret inputs from uncorrupted clients.

Remark 24 (Approximate version of $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$)**.** In this thesis, we illustrate the functionality in terms of exact computations. However, this functionality can be adapted to accommodate approximate computations as well. Specifically, for liberal security, $\mathcal{F}_{\mathsf{OtF-MPC}}$ can be defined to output $\hat{m} + \sigma + \sum_{i\in I} \eta_i$, where σ is a random coin used by the server (either chosen by the adversary if the server is corrupted or sampled by $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ if the server is honest), and η_i's are smudging errors added by the client (similarly, either chosen by the adversary if the client is corrupted or sampled by $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ if the client is honest). For standard security, $\mathcal{F}_{\mathsf{OtF-MPC}}$ can instead sample an error from a distribution and output $\hat{m} + \delta$. We refer to [HHK+25] for further detailed discussion on security specifically for schemes using approximate evaluation. Additionally, we remark the following result in Section 5.5 remains applicable when calls are made to the respective approximate version of $\mathcal{F}_{\mathsf{MGHE}}$ (as discussed in Remark 10) in the protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$.

5.4 Protocol for On-the-Fly MPC

We introduce the protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$, which realizes the functionality $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ as illustrated in Figures 5.7 to 5.11. Following a standard assumption in prior MPC literature [CD05, Sma23], we assume that communication between parties occurs over an *authenticated channel*. In the following discussion, we use the functionality $\mathcal{F}_{\mathsf{MsgBuf}}$ for this purpose.

[2]It is implicitly assumed that if $m_j = \perp$ for any $j \in [\ell]$, then the verification fails.

Protocol: $\Pi_{\mathsf{OtF\text{-}MPC}}$ (Part I)

Participants: n clients $\mathcal{P}_{\mathsf{cli}_1}, \mathcal{P}_{\mathsf{cli}_2}, \ldots, \mathcal{P}_{\mathsf{cli}_n}$. At a state, a subset of clients $\{\mathcal{P}_{\mathsf{cli}_i}\}_{i \in I}$ with $I \subseteq [n]$ form ℓ groups $\mathcal{P}_{\mathsf{grp}_j}$ with $\mathsf{grp}_j = \{\mathsf{cli}_i\}_{i \in I_j}$ with $I_j \subseteq I$ for $j \in [\ell]$. A server $\mathcal{P}_{\mathsf{srv}}$.

Setup & Key Generation

```
1 : $\mathcal{P}_{\mathsf{cli}} \leftarrow \langle \texttt{RegParam} : [\mathsf{sid}] \rangle$
2 :     for $u \in \{1,2\}$ do
3 :         Out $\langle \texttt{RegParam} : [(\mathsf{sid}, u)] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
4 : $\mathcal{P}_{\mathsf{cli}} \leftarrow \langle \texttt{RegKey} : [\mathsf{cli}, \mathsf{sid}] \rangle$
5 :     for $u \in \{1,2\}$ do
6 :         Out $\langle \texttt{RegKey} : [\mathsf{cli}, (\mathsf{sid}, u), \mathsf{pp}_u] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
7 : $\mathcal{P}_{\mathsf{grp}} \leftarrow \langle \texttt{JoinKey} : [\mathsf{grp}, \mathsf{sid}] \rangle$
8 :     for $u \in \{1,2\}$ do
9 :         Out $\langle \texttt{JoinKey} : [\mathsf{grp}, (\mathsf{sid}, u)] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
```

Figure 5.7: Protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$ (Part I – Setup and Key Generation) of an on-the-fly MPC protocol.

Setup and Key Generation. In Figure 5.7, we describe the setup phase of the protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$. Upon receiving the input $\langle \texttt{RegParam} : [\mathsf{sid}] \rangle$ to initiate an on-the-fly MPC session sid, a client $\mathcal{P}_{\mathsf{cli}}$ creates two associated sub-sessions, denoted as $(\mathsf{sid}, 1)$ and $(\mathsf{sid}, 2)$. The client then sends parameter registration requests $\langle \texttt{RegParam} : [(\mathsf{sid}, u)] \rangle$ for $u \in \{1, 2\}$ to $\mathcal{G}_{\mathsf{KRK}}$, which initializes two public parameters, pp_1 and pp_2, for use within the respective sub-sessions.

The same procedure is followed when a client $\mathcal{P}_{\mathsf{cli}}$ is activated with $\langle \texttt{RegKey} : [\mathsf{sid}] \rangle$: it forwards registration requests to $\mathcal{G}_{\mathsf{KRK}}$ as before. Similarly, if multiple clients intend to form a group grp, the virtual entity for the group $\mathcal{P}_{\mathsf{grp}}$ is activated with the input $\langle \texttt{JoinKey} : [\mathsf{grp}, \mathsf{sid}] \rangle$. This client then forwards requests to $\mathcal{G}_{\mathsf{KRK}}$, including those for the two sub-sessions.

Data Uploading. In Figure 5.8, we illustrate data uploading phase in the protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$. When activated with the input

$$\langle \texttt{Upload} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m] \rangle,$$

a client party $\mathcal{P}_{\mathsf{cli}}$ runs the Naor-Yung double encryption paradigm [NY90]. We let the client first request the joint public keys jpk_1 and jpk_2 of grp in session $(\mathsf{sid}, 1)$ and $(\mathsf{sid}, 2)$ from $\mathcal{G}_{\mathsf{KRK}}$. The client then send two inputs for encryption to $\mathcal{F}_{\mathsf{MGHE}}$ to encrypt m in sessions $(\mathsf{sid}, 1)$ and $(\mathsf{sid}, 2)$ respectively. Upon receiving the ciphertexts c_1 and c_2 as responses from $\mathcal{F}_{\mathsf{MGHE}}$, the protocol sends an input to $\mathcal{F}_{\mathsf{zkSNARK}}^{R_{\mathsf{NY}}}$, where the relation R_{NY} is defined as:

$$R_{\mathsf{NY}} = \left\{ \left(\begin{pmatrix} \mathsf{jpk}_1, c_1, \\ \mathsf{jpk}_2, c_2 \end{pmatrix}, (m, \omega_1, \omega_2) \right) \middle| \begin{array}{l} c_1 = \mathsf{MGHE.Enc}(\mathsf{jpk}_1, m; \omega_1) \\ \wedge \\ c_2 = \mathsf{MGHE.Enc}(\mathsf{jpk}_2, m; \omega_2) \end{array} \right\}$$

to generate an argument π with $(\mathsf{jpk}_1, c_1, \mathsf{jpk}_2, c_2)$ as the instance and (m, ω_1, ω_2) as the witness where ω_1 and ω_2 are random coins used at these two encryption calls to $\mathcal{F}_{\mathsf{MGHE}}$. We let $\mathcal{P}_{\mathsf{cli}}$ generate a message of the format

$$\{ \texttt{"up"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (c_1, c_2, \pi)] \} .$$

We then let the client $\mathcal{P}_{\mathsf{cli}}$ register this message at $\mathcal{F}_{\mathsf{MsgBuf}}$ to be later retrieved by the server.

Protocol: $\Pi_{\mathsf{OtF\text{-}MPC}}$ (Part II)

Data Uploading

```
$\mathcal{P}_{\mathsf{cli}} \leftarrow \langle \texttt{Upload} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m] \rangle$
    **for** $u \in \{1, 2\}$ **do**
        **Out** $\langle \texttt{RetKey} : [\mathsf{grp}, (\mathsf{sid}, u)] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
        **In** $\{ \texttt{"key"} : [\mathsf{grp}, (\mathsf{sid}, u), \mathsf{jpk}_u] \} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
        **Out** $\langle \texttt{Encrypt} : [\mathsf{grp}, (\mathsf{sid}, u), \mathsf{jpk}_u, m] \rangle \rightarrow \mathcal{F}_{\mathsf{MGHE}}$
        **In** $\{ \texttt{"ct"} : [\mathsf{grp}, (\mathsf{sid}, u), c_u] \} \rightarrow \mathcal{F}_{\mathsf{MGHE}}$
    // $\omega_1, \omega_2$ are random coins used at calls to $\mathcal{F}_{\mathsf{MGHE}}$ for encryption.
    **Out** $\langle \texttt{Prove} : [\mathsf{sid}, (\mathsf{jpk}_1, c_1, \mathsf{jpk}_2, c_2), (m, \omega_1, \omega_2)] \rangle \rightarrow \mathcal{F}_{\mathsf{zkSNARK}}^{R_{\mathsf{NY}}}$
    **In** $\{ \texttt{"pf"} : [\mathsf{sid}, (\mathsf{jpk}_1, \mathsf{jpk}_2, c_1, c_2), \pi] \} \rightarrow \mathcal{F}_{\mathsf{zkSNARK}}^{R_{\mathsf{NY}}}$
    $\mathsf{msg} := \{ \texttt{"up"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (c_1, c_2, \pi)] \}$
    **Out** $\langle \texttt{RegMsg} : [\mathsf{msg}] \rangle \rightarrow \mathcal{F}_{\mathsf{MsgBuf}}$
```

Figure 5.8: Protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$ (Part II – Data Uploading) of an on-the-fly MPC protocol.

Remark 25 (Knowledge of random coins)**.** We assume that an honest client $\mathcal{P}_{\mathsf{cli}}$ has knowledge of the random coins ω_1 and ω_2 used during encryption, i.e., when invoking $\mathcal{F}_{\mathsf{MGHE}}$ to encrypt m in sessions $(\mathsf{sid}, 1)$ and $(\mathsf{sid}, 2)$, respectively, via a channel that is inaccessible to the environment $\mathcal{Z}$. Notably, in the definition of $\mathcal{F}_{\mathsf{MGHE}}$, only corrupted clients are allowed to *explicitly* specify the random coins. This ensures that the environment $\mathcal{Z}$, which provides inputs to (honest) clients, does not control the random coins used for their encryptions. Moreover, the random coins (honestly) sampled by $\mathcal{S}$ are not provided as output to the client by $\mathcal{F}_{\mathsf{MGHE}}$. This is because honest clients are treated as dummy parties that simply take the output from $\mathcal{F}_{\mathsf{MGHE}}$ as its output in the ideal world. Revealing the random coins to the client would indirectly expose them to $\mathcal{Z}$, potentially enabling a trivial distinction between the real and ideal worlds when we show the realization of $\mathcal{F}_{\mathsf{MGHE}}$. Thus in the protocol description, we assume that an honest client possesses knowledge of the random coins. Essentially, when we consider the composed protocol $\Phi^{\mathcal{F}_{\mathsf{MGHE}} \to \Pi_{\mathsf{MGHE}}}$ where the client directly invokes $\mathsf{MGHE}.\texttt{Enc}(\cdot)$, the client itself samples the random coins thus knows them.

Circuit Evaluation. In Figure 5.9, we illustrate the circuit evaluation phase of the protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$. When a server party $\mathcal{P}_{\mathsf{srv}}$ is activated with the input

$$\big\langle \texttt{Evaluate} : \big[\mathsf{srv}, \mathsf{sid}, \mathsf{uid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j \in [\ell]}\big] \big\rangle,$$

the server retrieves the corresponding messages by sending inputs to $\mathcal{F}_{\mathsf{MsgBuf}}$. We then check that all the ciphertexts to be evaluated are in the same session. If there is an out-of-session message i.e., $\exists i, j \in [\ell]$ s.t. $\mathsf{sid}_j \neq \mathsf{sid}_i \vee \mathsf{sid}_j \neq \mathsf{sid}$, we then let the server abort the protocol. Also, if there is a message missing at the position $(\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)$ in the message registry, the server aborts the protocol as well. Otherwise, for each $j \in [\ell]$, the server proceeds with the following verification steps depending the type of the message.

– *The message is from a client (with prefix* "up"*)*: In this case, the message is parsed as

$$\big\{ \texttt{"up"} : \big[\mathsf{cli}, \mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j, (c_{j,1}, c_{j,2}, \pi_j)\big] \big\}.$$

Protocol: $\Pi_{\mathsf{OtF\text{-}MPC}}$ (Part III)

Evaluation

```
$\mathcal{P}_{\mathsf{srv}} \leftarrow \langle \texttt{Evaluate} : [\mathsf{srv}, \mathsf{sid}, \mathsf{uid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j\in[\ell]}] \rangle$
for $j \in [\ell]$ do
    Out $\langle \texttt{RetMsg} : [\mathsf{grp}_{j'}, \mathsf{sid}_{j'}, \mathsf{uid}_{j'}] \rangle \rightarrow \mathcal{F}_{\mathsf{MsgBuf}}$
    In msg $\leftarrow \mathcal{F}_{\mathsf{MsgBuf}}$
    // Verification for message from clients.
    if msg $= \{\texttt{"up"} : [\mathsf{cli}, \mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j, (c_{j,1}, c_{j,2}, \pi_j)]\}$ then
        for $u \in \{1,2\}$ do
            Out $\langle \texttt{RetKey} : [\mathsf{grp}_j, (\mathsf{sid}, u)] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
            In $\{\texttt{"key"} : [\mathsf{grp}_j, (\mathsf{sid}, u), \mathsf{jpk}_{j,u}]\} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
        Out $\langle \texttt{Verify} : [\mathsf{sid}, (\mathsf{jpk}_{j,1}, c_{j,1}, \mathsf{jpk}_{j,2}, c_{j,2}), \pi_j] \rangle \rightarrow \mathcal{F}^{R_{\mathsf{NY}}}_{\mathsf{zkSNARK}}$
        In $\{\texttt{"vf"} : [\mathsf{sid}, (\mathsf{jpk}_{j,1}, c_{j,1}, \mathsf{jpk}_{j,2}, c_{j,2}), \pi_j, b_j]\} \leftarrow \mathcal{F}^{R_{\mathsf{NY}}}_{\mathsf{zkSNARK}}$
        if $b_j = 0 \lor \mathsf{sid} \neq \mathsf{sid}_j$ then
            Abort
    // Verification for message from server.
    elseif msg $= \{\texttt{"eval"} : [\mathsf{srv}, \mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j, (c_{j,1}, c_{j,2}, \pi_j, (\mathsf{grp}_{j'}, \mathsf{sid}_{j'}, \mathsf{uid}_{j'})_{j'\in[\ell']})]\}$
        for $j' \in [\ell']$ do
            Out $\langle \texttt{RetMsg} : [\mathsf{grp}_{j'}, \mathsf{sid}_{j'}, \mathsf{uid}_{j'}] \rangle \rightarrow \mathcal{F}_{\mathsf{MsgBuf}}$
            In $\{\texttt{"}\cdot\texttt{"} : \mathsf{pid}_{j'}, \mathsf{grp}_{j'}, \mathsf{sid}_{j'}, \mathsf{uid}_{j'}, (c_{j',1}, c_{j',2}, \pi_{j'})\} \leftarrow \mathcal{F}_{\mathsf{MsgBuf}}$
            for $u \in \{1,2\}$ do
                Out $\langle \texttt{RetKey} : [\hat{\mathsf{grp}}_{j'}, (\mathsf{sid}_{j'}, u)] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
                In $\{\texttt{"key"} : [\hat{\mathsf{grp}}_{j'}, (\mathsf{sid}_{j'}, u), \mathsf{jpk}_{j',u}]\} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
        Out $\langle \texttt{Verify} : [\mathsf{sid}, (c_{j,u}, (\mathsf{jpk}_{j',u}, c_{j',u})_{j'\in[\ell]})_{u\in\{1,2\}}, \pi_j] \rangle \rightarrow \mathcal{F}^{R_{\mathsf{Eval}}}_{\mathsf{zkSNARK}}$
        In $\{\texttt{"vf"} : [\mathsf{sid}, (c_{j,u}, (\mathsf{jpk}_{j',u}, c_{j',u})_{j'\in[\ell]})_{u\in\{1,2\}}, \pi_j, b_j]\} \leftarrow \mathcal{F}^{R_{\mathsf{Eval}}}_{\mathsf{zkSNARK}}$
        if $b_j \neq 1 \lor (\exists j' \in [\ell']\ \mathsf{sid}_{j'} \neq \mathsf{sid}_j) \lor \mathsf{grp}_j \neq \cup_{j'\in[\ell']}\mathsf{grp}_{j'} \lor \mathsf{sid} \neq \mathsf{sid}_j$
        $\lor\ (\ \exists j' \in [\ell'] : (c_{j',1}, c_{j',2}) \in \mathsf{Invalid})$ then
            Abort
    else
        Abort
$\hat{\mathsf{grp}} := \cup_{j\in[\ell]}\ \mathsf{grp}_j$
for $j \in [\ell], u \in \{1,2\}$ do
    Out $\langle \texttt{RetKey} : [\mathsf{grp}_j, (\mathsf{sid}, u)] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
    In $\{\texttt{"key"} : [\mathsf{grp}_j, (\mathsf{sid}, u), \mathsf{jpk}_{j,u}]\} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
for $u \in \{1,2\}$ do
    Out $\langle \texttt{Evaluate} : [(\mathsf{sid}, u), f, (\mathsf{grp}_j, (\mathsf{sid}_j, u), \mathsf{jpk}_{j,u}, c_{j,u})_{j\in[\ell]}] \rangle \rightarrow \mathcal{F}_{\mathsf{MGHE}}$
    In $\{\texttt{"ct"} : [(\mathsf{sid}, u), \hat{c}_u, (\mathsf{grp}_j, (\mathsf{sid}_j, u), \mathsf{jpk}_{j,u}, c_{j,u})_{j\in[\ell]}]\} \leftarrow \mathcal{F}_{\mathsf{MGHE}}$
// $\sigma_1, \sigma_2$ are random coins used at the two calls to $\mathcal{F}_{\mathsf{MGHE}}$ for evaluation.
Out $\langle \texttt{Prove} : [\mathsf{sid}, (\hat{c}_1, (\mathsf{jpk}_{j,1}, c_{j,1})_{j\in[\ell]}, \hat{c}_2, (\mathsf{jpk}_{j,2}, c_{j,2})_{j\in[\ell]}), (f, \sigma_1, \sigma_2)] \rangle \rightarrow \mathcal{F}^{R_{\mathsf{Eval}}}_{\mathsf{zkSNARK}}$
In $\{\texttt{"pf"} : [\mathsf{sid}, (\hat{c}_1, (\mathsf{jpk}_{j,1}, c_{j,1})_{j\in[\ell]}, \hat{c}_2, (\mathsf{jpk}_{j,2}, c_{j,2})_{j\in[\ell]}), \pi)]\} \leftarrow \mathcal{F}^{R_{\mathsf{Eval}}}_{\mathsf{zkSNARK}}$
msg $:= \{\texttt{"eval"} : [\mathsf{srv}, \hat{\mathsf{grp}}, \mathsf{sid}, \mathsf{uid}, (\hat{c}_1, \hat{c}_2, \pi, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j\in[\ell]})]\}$
Out $\langle \texttt{RegMsg} : [\mathsf{msg}] \rangle \rightarrow \mathcal{F}_{\mathsf{MsgBuf}}$
```

Figure 5.9: Protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$ (Part III – Circuit Evaluation) of an on-the-fly MPC protocol. The part highlighted in blue indicates the behavior of $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ when multi-hop evaluation is involved.

We first let the server query $\mathcal{G}_{\mathsf{KRK}}$ to retrieve the joint public $\mathsf{jpk}_{j,1}$ and $\mathsf{jpk}_{j,2}$ for the group grp_j in the two sessions. The server then sends a verification request to $\mathcal{F}^{R_{\mathsf{NY}}}_{\mathsf{zkSNARK}}$ to verify if π_j is a valid argument for the instance $(\mathsf{jpk}_{j,1}, c_{j,1}, \mathsf{jpk}_{j,2}, c_{j,2})$. If $\mathcal{F}^{R_{\mathsf{NY}}}_{\mathsf{zkSNARK}}$ rejects the proof, the server aborts the protocol.

– *The message originates from the server (prefixed with* `"eval "`*)*: The message is then parsed as follows:

$$\left\{ \texttt{"eval"} : \left[\mathsf{srv}, \mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j, (\hat{c}_{j,1}, \hat{c}_{j,2}, \pi_j, (\mathsf{grp}_{j'}, \mathsf{sid}_{j'}, \mathsf{uid}_{j'})_{j' \in [\ell']}\right]\right\} .$$

For each $j' \in [\ell']$, we let the server retrieve the message at position $(\mathsf{grp}_{j'}, \mathsf{sid}_{j'}, \mathsf{uid}_{j'})$ from $\mathcal{F}_{\mathsf{MsgBuf}}$ and parse it as $(c_{j',1}, c_{j',2}, \pi_{j'})$. The server then retrieves the joint public keys $\mathsf{jpk}_{j',1}$ and $\mathsf{jpk}_{j',2}$ corresponding to group $\mathsf{grp}_{j'}$ from $\mathcal{G}_{\mathsf{KRK}}$. Next, it submits an input for verification to $\mathcal{F}^{R_{\mathsf{Eval}}}_{\mathsf{zkSNARK}}$ to check that $\hat{c}_{j,1}$ and $\hat{c}_{j,2}$ are valid evaluations of a circuit f over the ciphertexts $c_{j',1}$ and $c_{j',2}$, using the joint public keys $\mathsf{jpk}_{j',1}$ and $\mathsf{jpk}_{j',2}$. The relation R_{Eval} is defined as:

$$R_{\mathsf{Eval}} = \left\{ \begin{array}{c|} \left(\begin{pmatrix} \hat{c}_1, (\mathsf{jpk}_{j,1}, c_{j,1})_{j \in [\ell]}, \\ \hat{c}_2, (\mathsf{jpk}_{j,2}, c_{j,2})_{j \in [\ell]} \end{pmatrix}, (f, \sigma_1, \sigma_2) \right) \\ \\ \hat{c}_1 = \mathsf{MGHE}.\mathtt{Eval}(f, (\mathsf{jpk}_{j,1}, c_{j,1})_{j \in [\ell]}; \sigma_1) \\ \wedge \\ \hat{c}_2 = \mathsf{MGHE}.\mathtt{Eval}(f, (\mathsf{jpk}_{j,2}, c_{j,2})_{j \in [\ell]}; \sigma_2) \end{array} \right\} .$$

If the argument π_j is rejected by $\mathcal{F}^{R_{\mathsf{Eval}}}_{\mathsf{zkSNARK}}$, the server aborts the protocol. Additionally, the server validates the following conditions. If any of these checks fail, the server aborts the protocol.

(1) All evaluated ciphertexts must originate from the same session, matching the current session. Specifically, for all $i', j' \in [\ell']$, it must hold that $\mathsf{sid}_{i'} = \mathsf{sid}_{j'}$ and $\mathsf{sid}_{j'} = \mathsf{sid}_j$.

(2) The group grp_j must equal the union of groups $\cup_{j' \in [\ell']} \mathsf{grp}_{j'}$.

If all verifications succeed, the server proceeds to retrieves the joint public keys $\mathsf{jpk}_{j,1}$ and $\mathsf{jpk}_{j,2}$ for group grp_j for $j \in [\ell]$ and send two inputs for to $\mathcal{F}_{\mathsf{MGHE}}$ using the circuit f and ciphertext sets $(c_{1,1}, \ldots, c_{\ell,1})$ and $(c_{1,2}, \ldots, c_{\ell,2})$ for sessions $(\mathsf{sid}, 1)$ and $(\mathsf{sid}, 2)$. The server then receives $\hat{c}_1$ and $\hat{c}_2$ from $\mathcal{F}_{\mathsf{MGHE}}$ as responses. Subsequently, the server sends an input to $\mathcal{F}^{R_{\mathsf{Eval}}}_{\mathsf{zkSNARK}}$ to generate the arguments π. This argument is constructed using $(\hat{c}_1, (\mathsf{jpk}_{j,1}, c_{j,1})_{j \in [\ell]})$ and $(\hat{c}_2, (\mathsf{jpk}_{j,2}, c_{j,2})_{j \in [\ell]})$ as the instance, along with f as the witness. Finally, the server sets a new group to include all the clients whose inputs are involved in this computation as $\hat{\mathsf{grp}} = \cup_{j \in [\ell]} \mathsf{grp}_j$. The server then registers the message of the format

$$\left\{ \texttt{"eval"} : \left[\mathsf{srv}, \hat{\mathsf{grp}}, \mathsf{sid}, \mathsf{uid}, ((\hat{c}_1, \hat{c}_2, \pi), (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j \in [\ell]})\right]\right\}$$

to the functionality $\mathcal{F}_{\mathsf{MsgBuf}}$.

Remark 26 (Parameterization of the circuit in the relation)**.** It may be equivalent to define R_{Eval} by parameterizing the function f, i.e., using R^f_{Eval} and treating only (σ_1, σ_2) as the witness. In this case, the server would need to define a new relation each time it evaluates a new circuit. This approach eliminates the need to extract f as part of the witness, which can be more efficient. Intuitively, the security should still hold under the extraction of only (σ_1, σ_2), possibly even if we relax the assumption to a simulation-sound (zero-knowledge) SNARG. We leave a formal analysis of this question to future work.

Remark 27 (Verification for input ciphertexts)**.** We note that when verifying the validity of an evaluated ciphertext, in addition to checking correctness, we let the client and server verify that each input ciphertext contributing to the evaluation is valid. Specifically, it must either satisfy the well-formedness requirements of the Naor-Yung paradigm or be the result of a correct evaluation in a prior computation (as in Figure 5.9). In $\Pi_{\mathsf{OtF\text{-}MPC}}$, we adopt the simplest approach, where both the client and the server maintain a record of invalid ciphertexts and verify that these ciphertexts are not used in subsequent evaluations. For description simplicity, we assume that this check is implicitly enforced by the client and server, omitting it from the explicit protocol description. In the protocol, we denote this condition using $(c_{j,1}, c_{j,2}) \in \mathsf{Invalid}$.

We note that another approach for verification for multi-hop computation is to build a *chain of argument* by adding the condition that each input ciphertext is verified to be valid in the relation, as also described by Boneh et al. [BSW12]. We can then define the relation R^*_{Eval} as follows instead.

$$R^*_{\mathsf{Eval}} := \left\{ \begin{array}{c} \left(\begin{array}{l} \left(\hat{c}_u, (\mathsf{jpk}_{j,1}, c_{j,1})_{j\in[\ell]}\right)_{u\in\{1,2\}}, \\ \left(f, (\mathsf{jpk}_{(j,j'),u}, c_{(j,j'),u})_{u\in\{1,2\},j'\in[\ell'],j\in J\subseteq[\ell]}, \{\pi_j\}_{j\in[\ell]}\right) \end{array} \right) \Bigg| \\ \\ \forall u \in \{1,2\} : \hat{c}_u = \mathsf{MGHE}.\mathtt{Eval}(f, (\mathsf{jpk}_{j,u}, c_{j,u})_{j\in[\ell]}) \wedge \\ \left(\begin{array}{l} \forall j \in [\ell] : \mathtt{Vfy}^{\mathcal{O}_{\mathsf{RO}}}((\mathsf{jpk}_{j,u}, c_{j,u})_{u\in\{1,2\}}, \pi_j) = 1 \vee \\ \qquad\qquad \mathtt{Vfy}^{\mathcal{O}_{\mathsf{RO}}}((c_{j,u}, (\mathsf{jpk}_{j',u}, c_{j',u})_{j'\in[\ell']})_{u\in\{1,2\}}, \pi_j) = 1 \end{array} \right) \end{array} \right\}.$$

We note that the construction by Boneh et al. [BSW12] is based on the CRS model and assumes a simulation-sound NARG. However, simulation soundness is weaker than the security notion we require, as we need to ensure witness extraction. As discussed in Remark 17, achieving both extraction and succinctness at the same time in the plain CRS model is impossible.

Since the recursive structure of this relation and the computation it defines involves queries to the random oracle, we must consider a *relativized* SNARG in the random oracle model(s). Notably, Barbara et al. [BCG24] has shown that relativized SNARGs are not possible in ROM. In contrast, Chen et al. [CCG+23] showed that one can construct a relativized SNARG in the *arithmetized* random oracle model (AROM), which incorporates the arithmetization of a hash function. However, these works establish only knowledge soundness, which is weaker than the simulation security required for UC-security. Moreover, it remains unclear how programmability and observability influence the random oracle in these settings. We leave it as future work to investigate the existence of UC-secure relativized zkSNARKs.

Data Retrieval. In Figures 5.9 and 5.10, we illustrate the final phase for data retrieval. For the this phase, we consider two inputs on which the a client $\mathcal{P}_{\mathsf{cli}}$ is activated which corresponds to two sub-phases in this phase.

– *Obtaining Shares*: In Figure 5.10, we illustrate the process of retrieving partial decryptions for the client. On the input $\langle \mathtt{Share} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle$, the client $\mathcal{P}_{\mathsf{cli}}$ extracted the message by sending inputs to $\mathcal{F}_{\mathsf{MsgBuf}}$. If the message is uploaded by a client, we then let the client abort the execution.

  Otherwise, the message is uploaded by a server. We then let the client follow the same verification as what the server does during circuit evaluation. If any of the failures conditions is met, then we let the client abort the protocol. Otherwise, the client sends two inputs to $\mathcal{F}_{\mathsf{MGHE}}$ in sessions $(\mathsf{sid}, 1)$ and $(\mathsf{sid}, 2)$ to retrieve partial decryptions d_1 and d_2 with respect to the ciphertext c_1 and c_2 embedded in the message. Then the client sends an input to

Protocol: $\Pi_{\mathsf{OtF\text{-}MPC}}$ (Part IV)

Obtain Shares

```
$\mathcal{P}_{\mathsf{cli}} \leftarrow \langle \mathtt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle$
    **Out** $\langle \mathtt{RetMsg} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle \rightarrow \mathcal{F}_{\mathsf{MsgBuf}}$
    **In** $\mathsf{msg} \leftarrow \mathcal{F}_{\mathsf{MsgBuf}}$
    // Inputs from clients should not be retrieved.
    **if** $\mathsf{msg} = \{ \texttt{"up"} : [\mathsf{cli}', \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (c_1, c_2, \pi)] \}$ **then**
        Abort
    // Verification for message from server.
    **elseif** $\mathsf{msg} = \{ \texttt{"eval"} : [\mathsf{srv}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (\hat{c}_1, \hat{c}_2, \pi, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j \in [\ell]})] \}$
        **for** $j \in [\ell]$ **do**
            **Out** $\langle \mathtt{RetMsg} : [\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j] \rangle \rightarrow \mathcal{F}_{\mathsf{MsgBuf}}$
            **In** $\{ \texttt{"}\cdot\texttt{"} : \mathsf{pid}_j, \mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j, (c_{j,1}, c_{j,2}, \pi_j) \} \leftarrow \mathcal{F}_{\mathsf{MsgBuf}}$
            **for** $u \in \{1, 2\}$ **do**
                **Out** $\langle \mathtt{RetKey} : [\hat{\mathsf{grp}}_j, (\mathsf{sid}_j, u)] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
                **In** $\{ \texttt{"key"} : [\hat{\mathsf{grp}}_j, (\mathsf{sid}_j, u), \mathsf{jpk}_{j,u}] \} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
        **Out** $\langle \mathtt{Verify} : [\mathsf{sid}, (\hat{c}_u, (\mathsf{jpk}_{j,u}, c_{j,u})_{j \in [\ell]})_{u \in \{1,2\}}, \pi] \rangle \rightarrow \mathcal{F}^{R_{\mathsf{Eval}}}_{\mathsf{zkSNARK}}$
        **In** $\{ \texttt{"vf"} : [\mathsf{sid}, (\hat{c}_u, (\mathsf{jpk}_{j,u}, c_{j,u})_{j \in [\ell]})_{u \in \{1,2\}}, \pi, b] \} \leftarrow \mathcal{F}^{R_{\mathsf{Eval}}}_{\mathsf{zkSNARK}}$
        **if** $(b \neq 1) \vee (\exists j \in [\ell] : \mathsf{sid}_j \neq \mathsf{sid}) \vee \mathsf{grp} \neq \cup_{j \in [\ell]} \mathsf{grp}_j$
         $\vee (\exists j \in [\ell] : (c_{j,1}, c_{j,2}) \in \mathsf{Invalid})$ **then**
            Abort
    **else**
        Abort
    **for** $u \in \{1, 2\}$ **do**
        **Out** $\langle \mathtt{RetParam} : [(\mathsf{sid}, u)] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
        **In** $\{ \texttt{"param"} : [(\mathsf{sid}, u), \mathsf{pp}_u] \} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
        **Out** $\langle \mathtt{RetKey} : [\mathsf{cli}, (\mathsf{sid}, u)] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
        **In** $\{ \texttt{"key"} : [\mathsf{cli}, (\mathsf{sid}, u), (\mathsf{pk}_u, \mathsf{sk}_u)] \} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
        **Out** $\langle \mathtt{Share} : [\mathsf{cli}, \mathsf{grp}, (\mathsf{sid}, u), c_u] \rangle \rightarrow \mathcal{F}_{\mathsf{MGHE}}$
        **In** $\{ \texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, (\mathsf{sid}, u), c_u, d_u] \} \leftarrow \mathcal{F}_{\mathsf{MGHE}}$
    // $\gamma_1, \gamma_2$ are random coins used at calls for key registry to $\mathcal{G}_{\mathsf{KRK}}$.
    // $\eta_1, \eta_2$ are noises used at calls to $\mathcal{F}_{\mathsf{MGHE}}$ for partial decryption.
    **Out** $\langle \mathtt{Prove} : [\mathsf{sid}, (\mathsf{pp}_u, \mathsf{pk}_u, c_u, d_u)_{u \in \{1,2\}}, (\mathsf{sk}_u, \gamma_u, \eta_u)_{u \in \{1,2\}}] \rangle \rightarrow \mathcal{F}^{R_{\mathsf{Dec}}}_{\mathsf{zkSNARK}}$
    **In** $\{ \texttt{"pf"} : [\mathsf{sid}, (\mathsf{pp}_u, \mathsf{pk}_u, c_u, d_u)_{u \in \{1,2\}}, \pi] \} \leftarrow \mathcal{F}^{R_{\mathsf{Dec}}}_{\mathsf{zkSNARK}}$
    $\mathsf{msg} := \{ \texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, (c_1, d_1, c_2, d_2, \pi)] \}$
    **Out** $\langle \mathtt{RegSh} : [\mathsf{msg}] \rangle \rightarrow \mathcal{F}_{\mathsf{MsgBuf}}$
```

Figure 5.10: Protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$ (Part IV – Obtain partial decryptions) of an on-the-fly MPC protocol.

Protocol: $\Pi_{\mathsf{OtF\text{-}MPC}}$ (Part V)

Combine Shares

```
$\mathcal{P}_{\mathsf{cli}} \leftarrow \langle \texttt{Combine} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle$
  **Out** $\langle \texttt{RetMsg} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle \rightarrow \mathcal{F}_{\mathsf{MsgBuf}}$
  **In** $\mathsf{msg} \leftarrow \mathcal{F}_{\mathsf{MsgBuf}}$
  // Inputs from clients should not be retrieved.
  **if** $\mathsf{msg} = \{\texttt{"up"} : [\mathsf{cli}', \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (c_1, c_2, \pi)]\}$ **then**
    Abort
  // Verification for message from server.
  **elseif** $\mathsf{msg} = \{\texttt{"eval"} : [\mathsf{srv}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (\hat{c}_1, \hat{c}_2, \pi, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j \in [\ell]})]\}$
    $\mathsf{grp}' := \varnothing$
    **Out** $\langle \texttt{RetSh} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle \rightarrow \mathcal{F}_{\mathsf{MsgBuf}}$
    **In** $\mathsf{shs} \leftarrow \mathcal{F}_{\mathsf{MsgBuf}}$
    **for** $\mathsf{cli}_i \in \mathsf{grp}$ **do**
      **if** $\{\texttt{"sh"} : [(\mathsf{cli}_i, \mathsf{grp}, \mathsf{sid}, (c_{i,1}, d_{i,1}, c_{i,2}, d_{i,2}, \pi_i)]\} \in \mathsf{shs}$ **then**
        **for** $u \in \{1, 2\}$ **do**
          **Out** $\langle \texttt{RetParam} : [(\mathsf{sid}, u)] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
          **In** $\{\texttt{"param"} : [(\mathsf{sid}, u), \mathsf{pp}_u]\} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
          **Out** $\langle \texttt{RetKey} : [\mathsf{cli}_i, (\mathsf{sid}, u)] \rangle \rightarrow \mathcal{G}_{\mathsf{KRK}}$
          **In** $\{\texttt{"key"} : [\mathsf{cli}_i, (\mathsf{sid}, u), \mathsf{pk}_{i,u}]\} \leftarrow \mathcal{G}_{\mathsf{KRK}}$
        **Out** $\langle \texttt{Verify} : [\mathsf{sid}, (\mathsf{pp}_u, \mathsf{pk}_{i,u}, c_{i,u}, d_{i,u})_{u \in \{1,2\}}, \pi_i] \rangle \rightarrow \mathcal{F}^{R_{\mathsf{Dec}}}_{\mathsf{zkSNARK}}$
        **In** $\{\texttt{"vf"} : [\mathsf{sid}, (\mathsf{pp}_u, \mathsf{pk}_{i,u}, c_{i,u}, d_{i,u})_{u \in \{1,2\}}, \pi_i, b_i]\} \leftarrow \mathcal{F}^{R_{\mathsf{Dec}}}_{\mathsf{zkSNARK}}$
      **if** $b_i = 1 \wedge c_{i,1} = \hat{c}_1 \wedge c_{i,2} = \hat{c}_2$ **then**
        $\mathsf{grp}' := \mathsf{grp}' \cup \{\mathsf{cli}\}$
    **for** $u \in \{1, 2\}$ **do**
      **Out** $\langle \texttt{Combine} : [\mathsf{grp}, \mathsf{sid}, c_u, \{d_{i,u}\}_{\mathsf{cli}_i \in \mathsf{grp}'}] \rangle \rightarrow \mathcal{F}_{\mathsf{MGHE}}$
      **In** $\{\texttt{"pt"} : [\mathsf{grp}, \mathsf{sid}, c_u, m_u]\} \leftarrow \mathcal{F}_{\mathsf{MGHE}}$
    **if** $\boxed{m_1 \neq m_2}$ $||m_1 - m_2|| \leq \mathsf{Estimate}(\cdot)$ **then**
      Abort
    **Out** $\{\texttt{"ret"} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m_1]\} \rightarrow \mathcal{P}_{\mathsf{cli}}$
  **else**
    Abort
```

Figure 5.11: Protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$ (Part V – Combine partial decryptions) of an on-the-fly MPC protocol. The dot-boxed part is exclusive for MGHE on exact numbers and the highlighted part is exclusive for MGHE on approximate numbers.

$\mathcal{F}^{R_{\mathsf{Dec}}}_{\mathsf{zkSNARK}}$, where the relation R_{Dec} is defined as:

$$R_{\mathsf{Dec}} = \left\{ \left(\begin{matrix} \mathsf{pp}_1, \mathsf{pk}_1, c_1, d_1, \\ \mathsf{pp}_2, \mathsf{pk}_2, c_2, d_2 \end{matrix} \right), \left(\begin{matrix} \mathsf{sk}_1, \gamma_1, \eta_1, \\ \mathsf{sk}_2, \gamma_2, \eta_2 \end{matrix} \right) \middle| \begin{array}{l} \forall u \in \{1,2\} : \\ \quad d_u = \mathsf{MGHE.PDec}(\mathsf{sk}_u, c_u; \eta_u) \\ \quad \wedge\ \mathsf{pk}_u = \mathsf{PKGen}(\mathsf{pp}_u, \mathsf{sk}_u; \gamma_u) \end{array} \right\},$$

to generate an argument π with $(\mathsf{pp}_1, \mathsf{pk}_1, c_1, d_1, \mathsf{pp}_2, \mathsf{pk}_2, c_2, d_2)$ as instance, and the secret keys and the respective random coins used for key generation $(\mathsf{sk}_1, \gamma_1, \mathsf{sk}_2, \gamma_2)$ as witness (we similarly assume that the client knows the random coins used during key generation as in Remark 25). We assume that the public key pk can be derived from the secret key sk and the random coin γ used during key generation via a key derivation function PKGen. This ensures that the correctness of partial decryption can be proven when the correct secret key is used. Then the client generates a message of the following format

$$\{\texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, (c_1, d_1, c_2, d_2, \pi)]\}\,.$$

We then let the client register this message by sending input to $\mathcal{F}_{\mathsf{MsgBuf}}$.

– *Result Reconstruction*: In Figure 5.11, we illustrate the final step where the clients reconstruct the final result from partial decryptions uploaded by other clients. On the input $\langle\texttt{Combine} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}]\rangle$, we let a client $\mathcal{P}_{\mathsf{cli}}$ retrieve a set shs of all the partial decryptions that are available now by quering $\mathcal{F}_{\mathsf{MsgBuf}}$. For each client $\mathsf{cli}_i \in \mathsf{grp}$, we check if a message

$$\{\texttt{"sh"} : [\mathsf{cli}_i, \mathsf{grp}, \mathsf{sid}, (c_{i,1}, d_{i,1}, c_{i,2}, d_{i,2}, \pi_i]\}$$

is in the set shs. If not, we then continue with the next client in the group.

For a message in shs, the client sends an input to $\mathcal{F}^{R_{\mathsf{Dec}}}_{\mathsf{zkSNARK}}$ to verify that π proves that d_1 and d_2 are a valid partial decryptions for the ciphertext c_1 and c_2. The client maintains a “local group” grp' to collect the IDs of clients that produce valid shares. If the verification succeeds, then the client cli_i is added to the group grp'. Once all valid shares are collected, the client reconstructs these two messages in session $(\mathsf{sid}, 1)$ and $(\mathsf{sid}, 2)$ by sending two inputs to $\mathcal{F}_{\mathsf{MGHE}}$. Upon receiving m_1 and m_2 as responses from $\mathcal{F}_{\mathsf{MGHE}}$, if the reconstructed messages m_1 and m_2 are identical, the protocol outputs the message to the client. Otherwise, the protocol aborts.

We note that $\langle\texttt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}]\rangle$ and $\langle\texttt{Combine} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}]\rangle$ represent internal inputs triggered when the environment sends $\langle\texttt{Retrieve} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}]\rangle$ to a client. Specifically, upon receiving the input $\langle\texttt{Retrieve} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}]\rangle$, the client is sequentially activated with the input $\langle\texttt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}]\rangle$ followed by $\langle\texttt{Combine} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}]\rangle$. This structure accounts for the potential *time gap* between the reception of partial decryptions and the reconstruction of the final result, as the client must collect enough partial decryptions before combining them. Importantly, these internal inputs and outputs are not visible to the environment $\mathcal{Z}$, that is the environment $\mathcal{Z}$ only sees $\langle\texttt{Retrieve} : [\cdot]\rangle$ as inputs and $\{\texttt{"ret"} : [\cdot]\}$ as outputs, since exposing them would allow for a trivial distinguishing between the ideal and real worlds. We adopt this notation to simplify the description.

Remark 28 (Verification for Decryption Consistency)**.** We remark that if MGHE satisfies decryption consistency (Figure 3.12), then proving the correctness of partial decryption using R_{Dec} may not be necessary, provided that the adversary does not corrupt more than the threshold.

There are various approaches to achieving decryption consistency. Some works [CSS+22, BS23] leverage the error correction properties of linear secret sharing to ensure decryption consistency, while others [BGG+18] employ zero-knowledge proofs to verify the correctness of partial decryption, as in Figure 5.10. From a general perspective, we require MGHE to satisfy decryption consistency and incorporate zero-knowledge proofs for correctness in the protocol.

5.5 Realizing $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$

Theorem 5. *The protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$ UC-realizes $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ in $[\mathcal{F}_{\mathsf{MGHE}}, \mathcal{F}_{\mathsf{zkSNARK}}, \mathcal{F}_{\mathsf{MsgBuf}}]$-hybrid model in presence of a global key registry $\mathcal{G}_{\mathsf{KRK}}$ and a global random oracle $\mathcal{G}_{\mathsf{RO}}$ against a malicious adversary that non-adaptively corrupts the server and fewer $\frac{|\mathsf{grp}|}{t}$ clients in any group* grp *formed during the execution of the protocol for a threshold factor t.*

Proof. We now show the security of the protocol. Let $\mathcal{A}$ be an adversary that interacts with the parties running the protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$ in $[\mathcal{F}_{\mathsf{MGHE}}, \mathcal{F}_{\mathsf{zkSNARK}}, \mathcal{F}_{\mathsf{MsgBuf}}]$-hybrid model. We now construct an ideal-process adversary $\mathcal{S}$ such that any environment $\mathcal{Z}$ cannot distinguish between an interaction with $\mathcal{A}$ and $\Pi_{\mathsf{OtF\text{-}MPC}}$, and the interaction with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$. We let $\mathcal{S}$ run a internal copy of $\mathcal{A}$ and forwards message from $\mathcal{Z}$ to $\mathcal{A}$ and back. Then we let $\mathcal{S}$ corrupt the same clients (and the server) that $\mathcal{A}$ corrupts by sending a backdoor message to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$. Now we let $\mathcal{S}$ simulate each phase in the protocol as follows.

– **Corrupted Client in Data Uploading**: We let $\mathcal{S}$ simulate the behavior of $\mathcal{A}$ when the client is corrupted. Upon receiving from $\mathcal{A}$, on behalf of the corrupted client, a (backdoor) input

$$\langle \texttt{Encrypt} : [\mathsf{grp}, \mathsf{sid}, \mathsf{jpk}, m; \omega] \rangle$$

to $\mathcal{F}_{\mathsf{MGHE}}$, we then let $\mathcal{S}$ send back this input to $\mathcal{A}$ in the name of $\mathcal{F}_{\mathsf{MGHE}}$. Then $\mathcal{A}$ will return c as the ciphertext of m.

When $\mathcal{A}$ sends an input to $\mathcal{F}_{\mathsf{MsgBuf}}$ to register a message msg (on behalf of the corrupted client), we let $\mathcal{S}$ parses msg as

$$\{\texttt{"up"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (c_1, c_2, \pi)]\}\,.$$

If cli corresponds to a corrupted client, then $\mathcal{S}$ stores this message in internally for later use. Otherwise, the message is ignored.

Next, $\mathcal{S}$ sends the input $\langle \texttt{Observe} : [\mathsf{sid}] \rangle$ to $\mathcal{G}_{\mathsf{RO}}$ to obtain the query trace tr_z made by the environment $\mathcal{Z}$. Then for each (x, v) in tr_z, we let $\mathcal{S}$ send the input $\langle \texttt{IsProg} : [x] \rangle$ to obtain a list of programmed points pg_z made by $\mathcal{Z}$. Using this information, $\mathcal{S}$ invokes the extractor to compute

$$w \leftarrow \texttt{Ext}((\mathsf{jpk}_1, c_1, \mathsf{jpk}_2, c_2), \pi, \mathsf{tr}_z \setminus \mathsf{pg}_z).$$

We let $\mathcal{S}$ parse the witness w as (m, ω_1, ω_2) and verify that w is a valid witness for R_{NY}. Then $\mathcal{S}$ sends the following input to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ on behalf of the corrupted client:

$$\{\texttt{"up"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m]\}\,.$$

In all other cases, including when the message format cannot be parsed correctly or extraction cannot be made, $\mathcal{S}$ sets $m = \bot$ when sending the input to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$.

– **Honest Client in Data Uploading**: Observe that in $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$, when an honest client $\mathcal{P}_{\mathsf{cli}}$ is activated with the input $\langle \texttt{Upload} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m] \rangle$, the simulator $\mathcal{S}$ receives either m or $|m|$ from $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$, depending on the corruption status of grp. We let $\mathcal{S}$ simulate the expected interaction with $\mathcal{F}_{\mathsf{MGHE}}$ for $\mathcal{A}$. Specifically, we first let $\mathcal{S}$ retrieve jpk_u for $u \in \{1, 2\}$ for these two sessions from $\mathcal{G}_{\mathsf{KRK}}$. Then we let $\mathcal{S}$ send to $\mathcal{A}$ in the name of $\mathcal{F}_{\mathsf{MGHE}}$ the input

$$\langle \texttt{Encrypt} : [\mathsf{grp}, (\mathsf{sid}, u), \mathsf{jpk}_u, m/|m|] \rangle$$

for $u \in \{1,2\}$, based on whether m or $|m|$ was received by $\mathcal{S}$. Upon receiving c_1 and c_2 from $\mathcal{A}$, since $\mathcal{A}$ is not involved in $\mathcal{F}_{\mathsf{zkSNARK}}$, we let $\mathcal{S}$ directly invoke $(\pi, \mathsf{pg}) \leftarrow \mathtt{Sim}_{zk}{}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}$ to generate an argument π for the relation R_{NY} and program $\mathcal{G}_{\mathsf{RO}}$ with pg respectively. We then let $\mathcal{S}$ generate a message

$$\{\texttt{"up"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (c_1, c_2, \pi)]\}$$

When $\mathcal{A}$ sends an input to $\mathcal{F}_{\mathsf{MsgBuf}}$ to retrieve this message, we let $\mathcal{S}$ deliver the message to the adversary $\mathcal{A}$ (in the name of $\mathcal{F}_{\mathsf{MsgBuf}}$).

– **Corrupted Server in Circuit Evaluation**: We let $\mathcal{S}$ simulate the behavior of $\mathcal{A}$ in the case of a corrupted server. When $\mathcal{A}$ sends a (backdoor) input

$$\left\langle \texttt{Evaluate} : \left[\mathsf{sid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_j, c_j)_{j \in [\ell]}; \sigma\right] \right\rangle$$

to $\mathcal{F}_{\mathsf{MGHE}}$ on behalf of the corrupted client, we then let $\mathcal{S}$ send back this input to $\mathcal{A}$ in the name of $\mathcal{F}_{\mathsf{MGHE}}$. Then $\mathcal{A}$ will return $\hat{c}$ as the evaluated ciphertext.

When $\mathcal{A}$ sends an input to $\mathcal{F}_{\mathsf{MsgBuf}}$ to register a message msg, we let $\mathcal{S}$ parse msg as

$$\left\{\texttt{"eval"} : \left[\mathsf{srv}, \hat{\mathsf{grp}}, \mathsf{sid}, \mathsf{uid}, (\hat{c}_1, \hat{c}_2, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j \in [\ell]}, \pi)\right]\right\}.$$

If srv represents the corrupted server, $\mathcal{S}$ stores the message internally for later use; otherwise, the message is ignored.

Next, $\mathcal{S}$ queries $\mathcal{G}_{\mathsf{RO}}$ to obtain the query trace tr_z made by the environment $\mathcal{Z}$, and the list of points pg_z programmed by $\mathcal{Z}$. Then we let $\mathcal{S}$ then runs the extractor:

$$w \leftarrow \mathtt{Ext}(\hat{c}_1, (\mathsf{jpk}_{j,1}, c_{j,1})_{j \in [\ell]}, \hat{c}_2, (\mathsf{jpk}_{j,2}, c_{j,2})_{j \in [\ell]}, \pi, \mathsf{tr}_z \setminus \mathsf{pg}_z),$$

and parses w as a circuit f. Then $\mathcal{S}$ sends the following input to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$:

$$\left\{\texttt{"eval"} : \left[\mathsf{srv}, \hat{\mathsf{grp}}, \mathsf{sid}, \mathsf{uid}, f, \sqcup, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j \in [\ell]}\right]\right\},$$

allowing $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ to evaluate $\hat{m} = f(\{m_j\}_{j \in [\ell]})$ and record the result $\hat{m}$.

In all other cases including extraction failure or message cannot be parsed, $\mathcal{S}$ sends the following message to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ to use $\perp$ to indicate an invalid evaluation:

$$\left\{\texttt{"eval"} : \left[\mathsf{srv}, \hat{\mathsf{grp}}, \mathsf{sid}, \mathsf{uid}, f, \perp, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j \in [\ell]}\right]\right\}.$$

– **Honest Server in Circuit Evaluation**: When an honest server $\mathcal{P}_{\mathsf{srv}}$ is activated with the input

$$\left\langle \texttt{Evaluate} : \left[\mathsf{srv}, \mathsf{sid}, \mathsf{uid}, f, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j \in [\ell]}\right] \right\rangle,$$

we observe that in $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$, if the message at position $(\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)$ is uploaded by a client, the message must satisfy $m_j \neq \perp$; if uploaded by the server, the message m_j must represent a correct evaluation.

In $\Pi_{\mathsf{OtF\text{-}MPC}}$, when the message at position $(\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)$ originates from a client, the honest server parses it as

$$\left\{\texttt{"up"} : \left[\mathsf{cli}, \mathsf{grp}_j, \mathsf{sid}_j, (c_{j,1}, c_{j,2}, \pi)\right]\right\}$$

and verifies that π is a valid argument for the instance $(\mathsf{jpk}_{j,1}, c_{j,1}, \mathsf{jpk}_{j,2}, c_{j,2})$ using $\mathcal{F}_{\mathsf{zkSNARK}}^{R_{\mathsf{NY}}}$ by first retrieving the joint public keys $\mathsf{jpk}_{j,1}$ and $\mathsf{jpk}_{j,2}$ for grp_j from $\mathcal{G}_{\mathsf{KRK}}$.

By the properties of $\mathcal{F}_{\mathsf{zkSNARK}}^{R_{\mathsf{NY}}}$, we know that π will always be accepted if it is honestly generated by a client. Otherwise, the behavior of $\mathcal{F}_{\mathsf{zkSNARK}}^{R_{\mathsf{NY}}}$ is exactly the same as what we have the simulator $\mathcal{S}$ do in case of a corrupted client during the data uploading phase. Recall that in case of verification is not successful, then $\mathcal{S}$ writes $m = \bot$ to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$. which causes $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ to abort during evaluation. Thus we have that the condition to abort an execution is the same in both of these two worlds.

Other if the message is from the server, in $\Pi_{\mathsf{OtF\text{-}MPC}}$, the honest server parses it as

$$\left\{\texttt{"eval"} : \left[\mathsf{srv}, \mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j, (c_{j,1}, c_{j,2}, (\mathsf{grp}_{j^{\prime}}, \mathsf{sid}_{j'}, \mathsf{uid}_{j'})_{j' \in [\ell']}, \pi)\right]\right\}.$$

Similarly, if π are generated honestly by the server, they will always be accepted. Otherwise, $\mathcal{F}_{\mathsf{zkSNARK}}$ invokes $\mathtt{Ext}$ to extract a circuit f. By the definition of $\mathcal{F}_{\mathsf{MGHE}}$, if f satisfies the relation R_{Eval}, then it has

$$m_{j,1} = f(\{m_{j',1}\}_{j' \in [\ell']}) = m_{j,2} = f(\{m_{j',2}\}_{j' \in [\ell']}),$$

where $m_{j,1}$ and $m_{j,2}$ is the underlying message of $c_{j,1}$ and $c_{j,2}$ respectively. We recall that when the verification fails, $\mathcal{S}$ is defined to write $\hat{m} = \bot$ to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$, which prevent $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ from further execution. Thus, the condition for proceeding to the next step is equivalent in both the real and ideal worlds.

If all these checks pass, we let $\mathcal{S}$ simulate the expected interaction with $\mathcal{F}_{\mathsf{MGHE}}$ for $\mathcal{A}$. Note that the simulator receives $\hat{m} = f(\{m_j\}_{j \in [\ell]})$ from $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$. We then let $\mathcal{S}$ retrieve $\mathsf{jpk}_{j,1}$ and $\mathsf{jpk}_{j,2}$ for grp_j's from $\mathcal{G}_{\mathsf{KRK}}$ and send $\mathcal{A}$ in the name of $\mathcal{F}_{\mathsf{MGHE}}$ the input

$$\left\langle \texttt{Evaluate} : \left[(\mathsf{sid}, u), \sqcup, \hat{m}, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{jpk}_{j,u}, c_{j,u})_{j \in [\ell]}\right]\right\rangle$$

for $u \in \{1, 2\}$. Upon receiving $\hat{c}_1$ and $\hat{c}_2$ from $\mathcal{A}$, we let $\mathcal{S}$ invoke $(\pi, \mathsf{pg}) \leftarrow \mathtt{Sim}_{zk}{}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}$ to generate an argument π for the relation R_{Eval} and program $\mathcal{G}_{\mathsf{RO}}$ with pg. We then let $\mathcal{S}$ store the message

$$\left\{\texttt{"eval"} : \left[\mathsf{srv}, \hat{\mathsf{grp}}, \mathsf{sid}, \mathsf{uid}, (\hat{c}_1, \hat{c}_2, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j \in [\ell]}, \pi)\right]\right\}.$$

When $\mathcal{A}$ sends input to $\mathcal{F}_{\mathsf{MsgBuf}}$ to retrieve this message, we let $\mathcal{S}$ output it to $\mathcal{A}$.

– **Corrupted Client in Data Retrieval**: We now examine the case where a corrupted client $\mathcal{P}_{\mathsf{cli}}$ is activated with the input

$$\langle \texttt{Retrieve} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle.$$

We let $\mathcal{S}$ simulate the behavior of $\mathcal{A}$ in the case of a corrupted server. When $\mathcal{A}$ sends a (backdoor) input

$$\langle \texttt{Share} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, c; \eta] \rangle$$

to $\mathcal{F}_{\mathsf{MGHE}}$ on behalf of the corrupted client. We then let $\mathcal{S}$ send back this input to $\mathcal{A}$ in the name of $\mathcal{F}_{\mathsf{MGHE}}$. Then $\mathcal{A}$ will return d as the partial decryption.

When $\mathcal{A}$ uploads a partial decryption to $\mathcal{F}_{\mathsf{MsgBuf}}$ in the name of the corrupted client, we let $\mathcal{S}$ parses the message as

$$\{\texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (c_1, d_1, c_2, d_2, \pi)]\}.$$

If cli is not a corrupted client, we let $\mathcal{S}$ ignore this message. Then we similarly let $\mathcal{S}$ query $\mathcal{G}_{\mathsf{RO}}$ to obtain the query trace tr_z made by the environment $\mathcal{Z}$, and the list of points pg_z programmed by $\mathcal{Z}$. Then $\mathcal{S}$ calls the extractor to obtain the witness

$$w \leftarrow \mathtt{Ext}((c_1, d_1, c_2, d_2), \pi, \mathsf{tr}_z \setminus \mathsf{pg}_z),$$

and parse $w = (\mathsf{sk}_1, \gamma_1, \eta_1, \mathsf{sk}_2, \gamma_2, \eta_2)$. Note that if $w = \perp$, we then let $\mathcal{S}$ ignore this message. We then let $\mathcal{S}$ record this partial decryption but mark it as valid.

In the next sub-phase, when $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{MGHE}}$ on behalf of the corrupted client a (backdoor) input

$$\langle \mathtt{Combine} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle,$$

to combine partial decryption. We let $\mathcal{S}$ checks if $\mathcal{A}$ has corrupted the group grp and has obtained enough partial decryptions d (through $\mathcal{S}$). If yes, we let $\mathcal{S}$ send to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ on behalf of the corrupted client the message $\langle \mathtt{Retrieve} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle$ to get the message m. outputs to $\mathcal{A}$ the message

$$\{ \mathtt{"ret"} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}, m] \} .$$

– **Honest Client in Data Retrieval**: We now examine the case where an honest client $\mathcal{P}_{\mathsf{cli}}$ is activated with the input

$$\langle \mathtt{Retrieve} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle.$$

Note that we assume the group grp is not corrupted. In $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$, if the message originates from a client (i.e., it begins with the identifier $\{ \mathtt{"up"} : [\cdot] \}$), the execution is aborted. The same condition applies in $\Pi_{\mathsf{OtF\text{-}MPC}}$. Otherwise, if the message originates from the server (beginning with $\{ \mathtt{"eval"} : [\cdot] \}$), the embedded message $\hat{m}$ is considered valid only if $\hat{m} = f(\{m_j\}_{j \in [\ell]})$, where $\{m_j\}_{j \in [\ell]}$ are the input messages referenced in the message.

In $\Pi_{\mathsf{OtF\text{-}MPC}}$, this process is divided into two sub-phases. When $\mathcal{P}_{\mathsf{cli}}$ is activated with the sub-input $\langle \mathtt{Share} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle$, it first retrieves the corresponding message from $\mathcal{F}_{\mathsf{MsgBuf}}$ and parses it as

$$\left\{ \mathtt{"eval"} : \left[\mathsf{srv}, \hat{\mathsf{grp}}, \mathsf{sid}, \mathsf{uid}, (\hat{c}_1, \hat{c}_2, (\mathsf{grp}_j, \mathsf{sid}_j, \mathsf{uid}_j)_{j \in [\ell]}, \pi \right] \right\} .$$

If π are generated honestly by the server, they will be accepted. Otherwise, by the definition of $\mathcal{F}_{\mathsf{zkSNARK}}$, we can extract a circuits f such that

$$\hat{c}_u = \mathsf{MGHE}.\mathtt{Eval}(f, (\mathsf{jpk}_{j,u}, c_{j,u})_{j \in [\ell]})$$

for $u \in \{1, 2\}$. If verification of π fails, the client aborts the execution. Also recall from the earlier definition of $\mathcal{S}$ for a corrupted server that, when either $\hat{c}_1$ or $\hat{c}_2$ is an incorrect evaluation, we let $\mathcal{S}$ write $\hat{m} = \perp$ to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$, causing the protocol to abort during data retrieval. This results in consistent abortion conditions in both the ideal and real worlds.

We now let $\mathcal{S}$ simulate the interaction between $\mathcal{F}_{\mathsf{MGHE}}$ and $\mathcal{A}$. We let $\mathcal{S}$ send, on behalf of the corrupted client, a (backdoor) message

$$\langle \mathtt{Retrieve} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle$$

to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$. When the tuple $(\mathsf{grp}, \mathsf{sid}, \mathsf{uid})$ corresponds to a message from the server (identified by $\{ \mathtt{"eval"} : [\cdot] \}$), the simulator $\mathcal{S}$ obtains the evaluated plaintext $\hat{m}$. Then $\mathcal{S}$ sends the following input to $\mathcal{A}$ in the name of $\mathcal{F}_{\mathsf{MGHE}}$ for each $u \in \{1, 2\}$:

$$\langle \mathtt{Share} : [\mathsf{cli}, \mathsf{grp}, (\mathsf{sid}, u), c_u, \hat{m}] \rangle.$$

Upon receiving the partial decryptions d_1 and d_2, then $\mathcal{S}$ invokes $(\pi, \mathsf{pg}) \leftarrow \mathtt{Sim}_{zk}{}^{\mathcal{G}_{\mathsf{RO}}[\mathsf{sid}]}$ to generate an argument π for the relation R_{Dec} and programs $\mathcal{G}_{\mathsf{RO}}$ with pg. Then $\mathcal{S}$ constructs the message

$$\{\texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (c_1, d_1, c_2, d_2, \pi)]\}$$

and delivers it to $\mathcal{A}$ in the name of $\mathcal{F}_{\mathsf{MsgBuf}}$ when $\mathcal{A}$ queries to retrieve it.

In the next sub-phase, when the client in $\Pi_{\mathsf{OtF\text{-}MPC}}$ is activated with the sub-input

$$\langle \texttt{Combine} : [\mathsf{grp}, \mathsf{sid}, \mathsf{uid}] \rangle,$$

the client retrieves all available partial decryptions from $\mathcal{F}_{\mathsf{MsgBuf}}$ and parses each as

$$\{\texttt{"sh"} : [\mathsf{cli}, \mathsf{grp}, \mathsf{sid}, \mathsf{uid}, (c_1, d_1, c_2, d_2, \pi)]\}.$$

By the definition of $\mathcal{F}_{\mathsf{zkSNARK}}$, if the arguments π are generated honestly by clients, they will always be accepted. Since we assume that the adversary does not corrupt more clients than allowed by the threshold, an honest client will possess enough partial decryptions d_1 and d_2, which are honestly generated using the correct secret keys. Consequently, by the definition of $\mathcal{F}_{\mathsf{MGHE}}$, the pair (c_1, d_1) will reconstruct to an evaluated message $\hat{m}_1$, and (c_2, d_2) will reconstruct to $\hat{m}_2$, such that

$$\hat{m}_1 = \hat{m}_2 = f(\{m_{j,1}\}_{j\in[\ell]}) = f(\{m_{j,2}\}_{j\in[\ell]}).$$

If, however, $\hat{m}_1 \neq \hat{m}_2$, the protocol aborts. Notably, in the earlier definition of $\mathcal{S}$ for a corrupted server, $\mathcal{S}$ also sets $\hat{m} = \perp$ if a circuit f cannot be extracted such that both $\hat{c}_1$ and $\hat{c}_2$ are correct evaluations, leading to the same abort condition. Therefore, the outputs in both the ideal and real worlds remain consistent.

It can be observed that the combined view of $\mathcal{Z}$ and $\mathcal{A}$ in an execution of $\Pi_{\mathsf{OtF\text{-}MPC}}$ is distributed identically to the combined view of $\mathcal{Z}$ and the simulated copy of $\mathcal{A}$ within $\mathcal{S}$ in the ideal functionality $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$. Thus we have that the protocol $\Pi_{\mathsf{OtF\text{-}MPC}}$ securely realizes the functionality $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$ in $[\mathcal{F}_{\mathsf{MGHE}}, \mathcal{F}_{\mathsf{zkSNARK}}, \mathcal{F}_{\mathsf{MsgBuf}}]$-hybrid model in presence of a global key registry $\mathcal{G}_{\mathsf{KRK}}$ and a global random oracle $\mathcal{G}_{\mathsf{RO}}$. □

Remark 29 (Sufficiency of Single-Pass Structure)**.** We observe that a single encryption, as opposed to the double encryption required by the Naor-Yung transform, suffices for security when employing a UC-secure zkSNARK. Specifically, the simulation-extractability property of the zkSNARK ensures that a client generating an argument π, which proves that a ciphertext c encrypts a message m, must possess knowledge of the underlying m. When π is accepted by `Vfy`, the extractor `Ext` is guaranteed to recover m from π, allowing $\mathcal{S}$ to write m to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$. Conversely, if extraction fails, extractability ensures that π is rejected.

In contrast, if we relax the security requirements to a (zero-knowledge) SNARG that satisfies only *simulation soundness*, as defined in Definition 37, the Naor-Yung double encryption becomes necessary. In this case, when `Vfy` accepts an argument π, we can only deduce that $x \in \mathcal{L}(R)$ (i.e., c_1 and c_2 encrypt the same m), without any guarantee of extracting m from π. Consequently, $\mathcal{S}$ cannot write m to $\mathcal{F}_{\mathsf{OtF\text{-}MPC}}$, resulting in a simulation failure. The Naor-Yung paradigm mitigates this by leveraging double encryption to ensure that $x \in \mathcal{L}(R)$ implies a unique m extractable through decryption.

At a high level, security, specifically *non-malleability*, is achieved either by harnessing the extractability of a simulation-extractable NARG or by adopting the Naor-Yung paradigm with simulation-soundness. Both approaches effectively enforce that the client knows the underlying message to generate a valid ciphertext tuple, thereby ensuring the ciphertext's *non-malleability*.

Chapter 6

Future Work and Conclusion

6.1 Future Work

In this section, we outline several directions for future work building upon this thesis, including security consideration and possible improvement for efficiency.

Adaptive Security. Our current model assumes *non-adaptive* corruption of players. A natural extension is to explore stronger security guarantees that allow the adversary to *adaptively* corrupt players. Specifically, for the integrity layer, Chiesa and Fenzi [CF24] has proposed a stronger UC-secure zkSNARK that accounts for adaptive corruption by revealing the random coins used by both the prover and verifier to the adversary.

For the confidentiality layer, the adversary should be restricted from corrupting any group whose joint public key has been used in challenge encryption queries or the respective evaluations when formalizing the game-based security of MGHE. Without such a restriction, a trivial distinguishing becomes possible. For the UC-security of MGHE, the ideal functionality should deliver all messages sent and received by the clients upon corruption and the adversary's corruption capabilities should be restricted similarly. It is also worth investigating whether adaptive corruption will make one have to consider CCA2 security.

Attack on Key Generation. We observe that the introduction of the global key registry $\mathcal{G}_{\mathsf{KRK}}$ restrict the adversary from manipulating the key generation process except for choosing the random coins used for (individual) key generation. Thus during the key generation process, we actually always consider a semi-malicious adversary. As also indicated in [OPP14, DD22], the malformed evaluation keys may grant an adversary additional advantage. Intuitively, incorporating a SNARK to additionally prove the well-formedness of keys should mitigate this issue.

Also, we assume that the key aggregation is *honestly* done by actually integrating MPC functionality within $\mathcal{G}_{\mathsf{KRK}}$ to simplify the security proofs. A more rigorous analysis could be conducted by treating this functionality independently and show the realization of $\mathcal{F}_{\mathsf{MGHE}}$ in $\mathcal{F}_{\mathsf{MPC}}$-hybrid model.

Relaxed Assumption for zkSNARK. In this thesis, we adopt the standard assumption of a straightline simulation-extractable zkSNARK for UC-security. However, this assumption can be expensive to realize in practice, particularly for lattice-based zkSNARKs. As noted by Camenisch et al. [CKS11], simulation soundness can also suffice for UC-security, especially when the (zero-knowledge) SNARG is used only to prove correct execution. As discussed in Remark 29, we may relax our assumption to require only simulation soundness, while still ensuring protocol security. A direction is to formally analyze our construction under this relaxed assumption and also examine whether one-pass evaluation and decryption continue to provide security guarantees.

Implementation of Construction. In this thesis, we focus solely on the theoretical aspects of the construction. Implementing the proposed construction and evaluating its practical performance would be valuable future steps. Notably, several libraries for homomorphic encryption in multi-party settings, such as Lattigo [lat24], are available, along with zkSNARK libraries like Spartan [Set20]. We leave the implementation and performance benchmarking of our construction as future work.

6.2 Conclusion

In this thesis, we present a composable framework for homomorphic encryption (HE) in a multi-party setting, integrating it with zkSNARKs to ensure integrity through verifiability. We introduce novel game-based security notions that address both client-side and server-side confidentiality across various HE variants, including threshold, multi-key, and multi-group homomorphic encryption. Focusing on multi-group homomorphic encryption, we formalize its ideal functionality within the Universal Composability (UC) framework and establish the conditions required for its realization by reducing them to the security notions we define.

We propose a construction for on-the-fly multi-party computation (MPC), allowing parties to dynamically join computations while outsourcing them to an untrusted yet powerful server. This primitive is achieved by composing multi-group homomorphic encryption with zkSNARKs, leveraging a double-pass encryption and evaluation approach. We rigorously prove that our construction attains UC-security against a malicious adversary capable of non-adaptively corrupting a subset of clients and potentially the server.

We identify several future directions from this thesis, including refining the theoretical underpinnings of our security model, optimizing the efficiency of our construction, and exploring real-world implementations to assess performance and practicality.

In conclusion, we provide a concrete composable security analysis of verifiable homomorphic encryption in a multi-party setting, offering a security guarantee for the real-world application built through such compositions.

Bibliography

[AB09] Shweta Agrawal and Dan Boneh. Homomorphic MACs: MAC-based integrity for network coding. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09International Conference on Applied Cryptography and Network Security*, volume 5536 of *LNCS*, pages 292–305. Springer, Berlin, Heidelberg, June 2009. doi:10.1007/978-3-642-01957-9_18.

[ADMT22] Aydin Abadi, Changyu Dong, Steven J. Murdoch, and Sotirios Terzis. Multi-party updatable delegated private set intersection. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *LNCS*, pages 100–119. Springer, Cham, May 2022. doi:10.1007/978-3-031-18283-9_6.

[AH19] Asma Aloufi and Peizhao Hu. Collaborative homomorphic computation on data encrypted under multiple keys. *arXiv preprint arXiv:1911.04101*, 2019.

[AJJM20] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 28–57. Springer, Cham, November 2020. doi:10.1007/978-3-030-64375-1_2.

[AJL$^+$12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Berlin, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_29.

[AL11] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 17–34. Springer, Berlin, Heidelberg, March 2011. doi:10.1007/978-3-642-19379-8_2.

[ARS20] Behzad Abdolmaleki, Sebastian Ramacher, and Daniel Slamanig. Lift-and-shift: Obtaining simulation extractable subversion and updatable SNARKs generically. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1987–2005. ACM Press, November 2020. doi:10.1145/3372297.3417228.

[ATD16] Aydin Abadi, Sotirios Terzis, and Changyu Dong. VD-PSI: Verifiable delegated private set intersection on outsourced private datasets. In Jens Grossklags and Bart

Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 149–168. Springer, Berlin, Heidelberg, February 2016. doi:10.1007/978-3-662-54970-4_9.

[ATMD18] Aydin Abadi, Sotirios Terzis, Roberto Metere, and Changyu Dong. Efficient delegated private set intersection on outsourced private datasets. Cryptology ePrint Archive, Report 2018/496, 2018. URL: https://eprint.iacr.org/2018/496.

[AV21] Adi Akavia and Margarita Vald. On the privacy of protocols based on CPA-secure homomorphic encryption. Cryptology ePrint Archive, Report 2021/803, 2021. URL: https://eprint.iacr.org/2021/803.

[BBB$^+$22] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R. V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Report 2022/915, 2022. URL: https://eprint.iacr.org/2022/915.

[BBD24] Christian Badertscher, Fabio Banfi, and Jesus Diaz. What did come out of it? analysis and improvements of DIDComm messaging. Cryptology ePrint Archive, Paper 2024/1361, 2024. URL: https://eprint.iacr.org/2024/1361.

[BBH06] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 226–243. Springer, Berlin, Heidelberg, February 2006. doi:10.1007/11605805_15.

[BBM18] Christian Badertscher, Fabio Banfi, and Ueli Maurer. A constructive perspective on signcryption security. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 102–120. Springer, Cham, September 2018. doi:10.1007/978-3-319-98113-0_6.

[BCD$^+$24] Flavio Bergamaschi, Anamaria Costache, Dana Dachman-Soled, Hunter Kippen, Lucas LaBuff, and Rui Tang. Revisiting the security of approximate FHE with noise-flooding countermeasures. Cryptology ePrint Archive, Report 2024/424, 2024. URL: https://eprint.iacr.org/2024/424.

[BCFK21] Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 528–558. Springer, Cham, May 2021. doi:10.1007/978-3-030-75248-4_19.

[BCG24] Annalisa Barbara, Alessandro Chiesa, and Ziyi Guan. Relativized succinct arguments in the ROM do not exist. Cryptology ePrint Archive, Report 2024/728, 2024. URL: https://eprint.iacr.org/2024/728.

[BCH$^+$20] Christian Badertscher, Ran Canetti, Julia Hesse, Björn Tackmann, and Vassilis Zikas. Universal composition with global subroutines: Capturing global setup within plain UC. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 1–30. Springer, Cham, November 2020. doi:10.1007/978-3-030-64381-2_1.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th FOCS*, pages 186–195. IEEE Computer Society Press, October 2004. doi:10.1109/FOCS.2004.71.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Berlin, Heidelberg, October / November 2016. doi:10.1007/978-3-662-53644-5_2.

[BCTV13] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive arguments for a von neumann architecture. Cryptology ePrint Archive, Report 2013/879, 2013. URL: https://eprint.iacr.org/2013/879.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Berlin, Heidelberg, May 2011. doi:10.1007/978-3-642-20465-4_11.

[BdPMW16] Florian Bourse, Rafaël del Pino, Michele Minelli, and Hoeteck Wee. FHE circuit privacy almost for free. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 62–89. Springer, Berlin, Heidelberg, August 2016. doi:10.1007/978-3-662-53008-5_3.

[BEJMQ25] Wasilij Beskorovajnov, Sarai Eilebrecht, Yufan Jiang, and Jörn Mueller-Quade. A formal treatment of homomorphic encryption based outsourced computation in the universal composability framework. Cryptology ePrint Archive, Paper 2025/109, 2025. URL: https://eprint.iacr.org/2025/109.

[BF11] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 149–168. Springer, Berlin, Heidelberg, May 2011. doi:10.1007/978-3-642-20465-4_10.

[BFKW09] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 68–87. Springer, Berlin, Heidelberg, March 2009. doi:10.1007/978-3-642-00468-1_5.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. doi:10.1145/62212.62222.

[BGG$^+$18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Cham, August 2018. doi:10.1007/978-3-319-96884-1_19.

[BGV11] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 111–131. Springer, Berlin, Heidelberg, August 2011. doi:10.1007/978-3-642-22792-9_7.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012. doi:10.1145/2090236.2090262.

[BJSW24] Olivier Bernard, Marc Joye, Nigel P. Smart, and Michael Walter. Drifting towards better error probabilities in fully homomorphic encryption schemes. Cryptology ePrint Archive, Paper 2024/1718, 2024. URL: https://eprint.iacr.org/2024/1718.

[BM18] Christian Badertscher and Ueli Maurer. Composable and robust outsourced storage. In Nigel P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 354–373. Springer, Cham, April 2018. doi:10.1007/978-3-319-76953-0_19.

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 190–213. Springer, Berlin, Heidelberg, August 2016. doi:10.1007/978-3-662-53018-4_8.

[BPR24] Dan Boneh, Aditi Partap, and Lior Rotem. Accountability for misbehavior in threshold decryption via threshold traitor tracing. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 317–351. Springer, Cham, August 2024. doi:10.1007/978-3-031-68394-7_11.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. doi:10.1145/168588.168596.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Berlin, Heidelberg, May / June 2006. doi:10.1007/11761679_25.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Berlin, Heidelberg, August 2012. doi:10.1007/978-3-642-32009-5_50.

[BS21] Karim Baghery and Mahdi Sedaghat. Tiramisu: Black-box simulation extractable NIZKs in the updatable CRS model. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *CANS 21*, volume 13099 of *LNCS*, pages 531–551. Springer, Cham, December 2021. doi:10.1007/978-3-030-92548-2_28.

[BS23] Katharina Boudgoust and Peter Scholl. Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part I*, volume 14438 of *LNCS*, pages 371–404. Springer, Singapore, December 2023. doi:10.1007/978-981-99-8721-4_12.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Berlin, Heidelberg, March 2011. doi:10.1007/978-3-642-19571-6_16.

[BSW12] Dan Boneh, Gil Segev, and Brent Waters. Targeted malleability: homomorphic encryption for restricted computations. In Shafi Goldwasser, editor, *ITCS 2012*, pages 350–366. ACM, January 2012. doi:10.1145/2090236.2090264.

[Cac95] Christian Cachin. On-line secret sharing. In Colin Boyd, editor, *5th IMA International Conference on Cryptography and Coding*, volume 1025 of *LNCS*, pages 190–198. Springer, Berlin, Heidelberg, December 1995. doi:10.1007/3-540-60693-9_22.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. URL: https://eprint.iacr.org/2000/067.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. doi:10.1109/SFCS.2001.959888.

[Can03] Ran Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. URL: https://eprint.iacr.org/2003/239.

[Can20] Ran Canetti. Universally composable security. *J. ACM*, 67(5), September 2020. doi:10.1145/3402457.

[CCG$^+$23] Megan Chen, Alessandro Chiesa, Tom Gur, Jack O'Connor, and Nicholas Spooner. Proof-carrying data from arithmetized random oracles. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 379–404. Springer, Cham, April 2023. doi:10.1007/978-3-031-30617-4_13.

[CCKY25] Jung Hee Cheon, Hyeongmin Choe, Seunghong Kim, and Yongdong Yeo. Reusable dynamic multi-party homomorphic encryption. Cryptology ePrint Archive, Paper 2025/581, 2025. URL: https://eprint.iacr.org/2025/581.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 3–22. Springer, Berlin, Heidelberg, August 2015. doi:10.1007/978-3-662-48000-7_1.

[CCM$^+$25] Bhuvnesh Chaturvedi, Anirban Chakraborty, Nimish Mishra, Ayantika Chatterjee, and Debdeep Mukhopadhyay. IND-CPA$^\mathrm{c}$: A new security notion for conditional decryption in fully homomorphic encryption. Cryptology ePrint Archive, Paper 2025/045, 2025. URL: https://eprint.iacr.org/2025/045.

[CCP$^+$24] Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. Attacks against the IND-CPA$^\mathrm{D}$ security of exact FHE schemes. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2505–2519. ACM Press, October 2024. doi:10.1145/3658644.3690341.

[CD05] Ronald Cramer and Ivan Damgård. *Multiparty Computation, an Introduction*, pages 41–87. Birkhäuser Basel, Basel, 2005. doi:10.1007/3-7643-7394-6_2.

[CDG$^+$18] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78381-9_11.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Berlin, Heidelberg, February 2007. doi:10.1007/978-3-540-70936-7_4.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 336–352. Springer, Berlin, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_21.

[CF24] Alessandro Chiesa and Giacomo Fenzi. zkSNARKs in the ROM with unconditional UC-security. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part I*, volume 15364 of *LNCS*, pages 67–89. Springer, Cham, December 2024. doi:10.1007/978-3-031-78011-0_3.

[CFP$^+$24] Sébastien Canard, Caroline Fontaine, Duong Hieu Phan, David Pointcheval, Marc Renard, and Renaud Sirdey. Relations among new CCA security notions for approximate FHE. Cryptology ePrint Archive, Report 2024/812, 2024. URL: https://eprint.iacr.org/2024/812.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020. doi:10.1007/s00145-019-09319-x.

[CGKS23] Matteo Campanelli, Chaya Ganesh, Hamidreza Khoshakhlagh, and Janno Siim. Impossibilities in succinct arguments: Black-box extraction and more. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 465–489. Springer, Cham, July 2023. doi:10.1007/978-3-031-37679-5_20.

[CHM$^+$20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020. doi:10.1007/978-3-030-45721-1_26.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, November 2014. doi:10.1145/2660267.2660374.

[CKKR19] Jan Camenisch, Stephan Krenn, Ralf Küsters, and Daniel Rausch. iUC: Flexible universal composability made simple. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 191–221. Springer, Cham, December 2019. doi:10.1007/978-3-030-34618-8_7.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Cham, December 2017. doi:10.1007/978-3-319-70694-8_15.

[CKP$^+$22] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. Verifiable encodings for secure homomorphic analytics. *arXiv preprint arXiv:2207.14071*, 2022.

[CKS11] Jan Camenisch, Stephan Krenn, and Victor Shoup. A framework for practical universally composable zero-knowledge protocols. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 449–467. Springer, Berlin, Heidelberg, December 2011. doi:10.1007/978-3-642-25385-0_24.

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 630–656. Springer, Berlin, Heidelberg, August 2015. doi:10.1007/978-3-662-48000-7_31.

[CMS$^+$23] Sylvain Chatel, Christian Mouchet, Ali Utkan Sahin, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. PELTA - shielding multiparty-FHE against malicious adversaries. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 711–725. ACM Press, November 2023. doi:10.1145/3576915.3623139.

[CRRV17] Ran Canetti, Srinivasan Raghuraman, Silas Richelson, and Vinod Vaikuntanathan. Chosen-ciphertext secure fully homomorphic encryption. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 213–240. Springer, Berlin, Heidelberg, March 2017. doi:10.1007/978-3-662-54388-7_8.

[CSBB24] Marina Checri, Renaud Sirdey, Aymen Boudguiga, and Jean-Paul Bultel. On the practical CPAD security of “exact” and threshold FHE schemes and libraries. Cryptology ePrint Archive, Report 2024/116, 2024. URL: https://eprint.iacr.org/2024/116.

[CSS$^+$22] Siddhartha Chowdhury, Sayani Sinha, Animesh Singh, Shubham Mishra, Chandan Chaudhary, Sikhar Patranabis, Pratyay Mukherjee, Ayantika Chatterjee, and Debdeep Mukhopadhyay. Efficient threshold FHE with application to real-time systems. Cryptology ePrint Archive, Report 2022/1625, 2022. URL: https://eprint.iacr.org/2022/1625.

[CY24] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024. URL: https://github.com/hash-based-snargs-book.

[CZW17] Long Chen, Zhenfeng Zhang, and Xueqing Wang. Batched multi-hop multi-key FHE from ring-LWE with compact ciphertext extension. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 597–627. Springer, Cham, November 2017. doi:10.1007/978-3-319-70503-3_20.

[DD22] Nico Döttling and Jesko Dujmovic. Maliciously circuit-private FHE from information-theoretic principles. In Dana Dachman-Soled, editor, *ITC 2022*, volume 230 of *LIPIcs*, pages 4:1–4:21. Schloss Dagstuhl, July 2022. doi:10.4230/LIPIcs.ITC.2022.4.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *23rd ACM STOC*, pages 542–552. ACM Press, May 1991. doi:10.1145/103418.103474.

[DDO$^+$01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Berlin, Heidelberg, August 2001. doi:10.1007/3-540-44647-8_33.

[DS16] Léo Ducas and Damien Stehlé. Sanitization of FHE ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 294–310. Springer, Berlin, Heidelberg, May 2016. doi:10.1007/978-3-662-49890-3_12.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985. doi:10.1109/TIT.1985.1057074.

[FGP14] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 844–855. ACM Press, November 2014. doi:10.1145/2660267.2660366.

[FIM$^+$01] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *ICALP 2001*, volume 2076 of *LNCS*, pages 927–938. Springer, Berlin, Heidelberg, July 2001. doi:10.1007/3-540-48224-5_75.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Berlin, Heidelberg, August 2005. doi:10.1007/11535218_10.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Berlin, Heidelberg, December 2012. doi:10.1007/978-3-642-34931-7_5.

[FNP20] Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 124–154. Springer, Cham, May 2020. doi:10.1007/978-3-030-45388-6_5.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. URL: https://eprint.iacr.org/2012/144.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009. doi:10.1145/1536414.1536440.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Berlin, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_37.

[GKKR10] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 142–160. Springer, Berlin, Heidelberg, May 2010. doi:10.1007/978-3-642-13013-7_9.

[GKO$^+$23] Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Witness-succinct universally-composable SNARKs. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 315–346. Springer, Cham, April 2023. doi:10.1007/978-3-031-30617-4_11.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013. doi:10.1145/2488608.2488678.

[GNSJ24] Qian Guo, Denis Nabokov, Elias Suvanto, and Thomas Johansson. Key recovery attacks on approximate homomorphic encryption with non-worst-case noise flooding countermeasures. In *Usenix Security*, 2024.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Berlin, Heidelberg, May / June 2006. doi:10.1007/11761679_21.

[Gro06a] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. 2006.

[Gro06b] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, Berlin, Heidelberg, December 2006. doi:10.1007/11935230_29.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 301–320. Springer, Berlin, Heidelberg, December 2013. `doi:10.1007/978-3-642-42045-0_16`.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. URL: `https://eprint.iacr.org/2019/953`.

[HHK$^+$25] Intak Hwang, Yisol Hwang, Miran Kim, Dongwon Lee, and Yongsoo Song. Provably secure approximate computation protocols from CKKS. Cryptology ePrint Archive, Paper 2025/395, 2025. URL: `https://eprint.iacr.org/2025/395`.

[HMS25] Intak Hwang, Seonhong Min, and Yongsoo Song. Ciphertext-simulatable HE from BFV with randomized evaluation. Cryptology ePrint Archive, Paper 2025/203, 2025. URL: `https://eprint.iacr.org/2025/203`.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 575–594. Springer, Berlin, Heidelberg, February 2007. `doi:10.1007/978-3-540-70936-7_31`.

[JKLS18] Xiaoqian Jiang, Miran Kim, Kristin E. Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1209–1222. ACM Press, October 2018. `doi:10.1145/3243734.3243837`.

[JMSW02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In Bart Preneel, editor, *CT-RSA 2002*, volume 2271 of *LNCS*, pages 244–262. Springer, Berlin, Heidelberg, February 2002. `doi:10.1007/3-540-45760-7_17`.

[JNO16] Thomas P. Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A framework for outsourcing of secure computation. Cryptology ePrint Archive, Report 2016/037, 2016. URL: `https://eprint.iacr.org/2016/037`.

[JP14] Abhishek Jain and Omkant Pandey. Non-malleable zero knowledge: Black-box constructions and definitional relationships. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 435–454. Springer, Cham, September 2014. `doi:10.1007/978-3-319-10879-7_25`.

[JY14] Chihong Joo and Aaram Yun. Homomorphic authenticated encryption secure against chosen-ciphertext attack. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 173–192. Springer, Berlin, Heidelberg, December 2014. `doi:10.1007/978-3-662-45608-8_10`.

[KLSW24] Hyesun Kwak, Dongwon Lee, Yongsoo Song, and Sameer Wagh. A general framework of homomorphic encryption for multiple parties with non-interactive key-aggregation. In Christina Pöpper and Lejla Batina, editors, *ACNS 24International Conference on Applied Cryptography and Network Security, Part II*, volume 14584 of *LNCS*, pages 403–430. Springer, Cham, March 2024. doi:10.1007/978-3-031-54773-7_16.

[KNY16] Ilan Komargodski, Moni Naor, and Eylon Yogev. How to share a secret, infinitely. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 485–514. Springer, Berlin, Heidelberg, October / November 2016. doi:10.1007/978-3-662-53644-5_19.

[KP17] Ilan Komargodski and Anat Paskin-Cherniavsky. Evolving secret sharing: Dynamic thresholds and robustness. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 379–393. Springer, Cham, November 2017. doi:10.1007/978-3-319-70503-3_12.

[KS23] Kamil Kluczniak and Giacomo Santato. On circuit private, multikey and threshold approximate homomorphic encryption. Cryptology ePrint Archive, Report 2023/301, 2023. URL: https://eprint.iacr.org/2023/301.

[KVMGH23] Christian Knabenhans, Alexander Viand, Antonio Merino-Gallardo, and Anwar Hithnawi. Vfhe: Verifiable fully homomorphic encryption. In *12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC)*, 2023.

[KZM$^+$15] Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093, 2015. URL: https://eprint.iacr.org/2015/1093.

[lat24] Lattigo v6. Online: https://github.com/tuneinsight/lattigo, August 2024. EPFL-LDS, Tune Insight SA.

[LKL$^+$22] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *iEEE Access*, 10:30039–30054, 2022.

[LM21] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 648–677. Springer, Cham, October 2021. doi:10.1007/978-3-030-77870-5_23.

[LMSS22] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. Securing approximate homomorphic encryption using differential privacy. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 560–589. Springer, Cham, August 2022. doi:10.1007/978-3-031-15802-5_20.

[LMSV12] Jake Loftus, Alexander May, Nigel P. Smart, and Frederik Vercauteren. On CCA-secure somewhat homomorphic encryption. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 55–72. Springer, Berlin, Heidelberg, August 2012. doi:10.1007/978-3-642-28496-0_4.

[LR22a] Anna Lysyanskaya and Leah Namisa Rosenbloom. Efficient and universally composable non-interactive zero-knowledge proofs of knowledge with security against adaptive corruptions. Cryptology ePrint Archive, Report 2022/1484, 2022. URL: https://eprint.iacr.org/2022/1484.

[LR22b] Anna Lysyanskaya and Leah Namisa Rosenbloom. Universally composable Σ-protocols in the global random-oracle model. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 203–233. Springer, Cham, November 2022. doi:10.1007/978-3-031-22318-1_8.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012. doi:10.1145/2213977.2214086.

[LWZ18] Shimin Li, Xin Wang, and Rui Zhang. Privacy-preserving homomorphic macs with efficient verification. In *Web Services–ICWS 2018: 25th International Conference, Held as Part of the Services Conference Federation, SCF 2018, Seattle, WA, USA, June 25-30, 2018, Proceedings 16*, pages 100–115. Springer, 2018.

[LZ21] Chen-Da Liu Zhang. *Multi-Party Computation: Definitions, Enhanced Security Guarantees and Efficiency*. PhD thesis, ETH Zurich, 2021.

[Mau11] Ueli Maurer. Constructive cryptography – a new paradigm for security definitions and proofs. In S. Moedersheim and C. Palamidessi, editors, *Theory of Security and Applications (TOSCA 2011)*, volume 6993 of *Lecture Notes in Computer Science*, pages 33–56. Springer-Verlag, 4 2011.

[MBH23] Christian Mouchet, Elliott Bertrand, and Jean-Pierre Hubaux. An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption. *Journal of Cryptology*, 36(2):10, 2023.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019. doi:10.1145/3319535.3339817.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[MN24] Mark Manulis and Jérôme Nguyen. Fully homomorphic encryption beyond IND-CCA1 security: Integrity through verifiability. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 63–93. Springer, Cham, May 2024. doi:10.1007/978-3-031-58723-8_3.

[MTBH21] Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *PoPETs*, 2021(4):291–311, October 2021. doi:10.2478/popets-2021-0071.

[MTPBH21] Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *Proceedings on Privacy Enhancing Technologies*, 2021(4):291–311, 2021.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Berlin, Heidelberg, May 2016. `doi:10.1007/978-3-662-49896-5_26`.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990. `doi:10.1145/100216.100273`.

[OG10] Femi G. Olumofin and Ian Goldberg. Privacy-preserving queries over relational databases. In Mikhail J. Atallah and Nicholas J. Hopper, editors, *PETS 2010*, volume 6205 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, July 2010. `doi:10.1007/978-3-642-14527-8_5`.

[OPP14] Rafail Ostrovsky, Anat Paskin-Cherniavsky, and Beni Paskin-Cherniavsky. Maliciously circuit-private FHE. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 536–553. Springer, Berlin, Heidelberg, August 2014. `doi:10.1007/978-3-662-44371-2_30`.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Berlin, Heidelberg, May 1999. `doi:10.1007/3-540-48910-X_16`.

[Par21] Jeongeun Park. Homomorphic encryption for multiple users with less communications. Cryptology ePrint Archive, Report 2021/1085, 2021. URL: `https://eprint.iacr.org/2021/1085`.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013. `doi:10.1109/SP.2013.47`.

[PR05] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 533–542. ACM Press, May 2005. `doi:10.1145/1060590.1060670`.

[PS16] Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 217–238. Springer, Berlin, Heidelberg, October / November 2016. `doi:10.1007/978-3-662-53644-5_9`.

[PS24] Alain Passelègue and Damien Stehlé. Low communication threshold fully homomorphic encryption. Cryptology ePrint Archive, Paper 2024/1984, 2024. URL: `https://eprint.iacr.org/2024/1984`.

[PTH21] Robert Podschwadt, Daniel Takabi, and Peizhao Hu. Sok: Privacy-preserving deep learning with homomorphic encryption. *arXiv preprint arXiv:2112.12855*, 2021.

[RAD$^+$78] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, February 1978. doi:10.1145/359340.359342.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999. doi:10.1109/SFFCS.1999.814628.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Cham, August 2020. doi:10.1007/978-3-030-56877-1_25.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. doi:10.1145/359168.359176.

[Sma23] Nigel P. Smart. Practical and efficient FHE-based MPC. In Elizabeth A. Quaglia, editor, *19th IMA International Conference on Cryptography and Coding*, volume 14421 of *LNCS*, pages 263–283. Springer, Cham, December 2023. doi:10.1007/978-3-031-47818-5_14.

[SVdV16] Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16International Conference on Applied Cryptography and Network Security*, volume 9696 of *LNCS*, pages 346–366. Springer, Cham, June 2016. doi:10.1007/978-3-319-39555-5_19.

[TC16] V Thangam and K Chandrasekaran. Elliptic curve based secure outsourced computation in multi-party cloud environment. In *Security in Computing and Communications: 4th International Symposium, SSCC 2016, Jaipur, India, September 21-24, 2016, Proceedings 4*, pages 199–212. Springer, 2016.

[TDB16] Giulia Traverso, Denise Demirel, and Johannes A. Buchmann. Dynamic and verifiable hierarchical secret sharing. In Anderson C. A. Nascimento and Paulo Barreto, editors, *ICITS 16*, volume 10015 of *LNCS*, pages 24–43. Springer, Cham, August 2016. doi:10.1007/978-3-319-49175-2_2.

[TW24] Louis Tremblay Thibault and Michael Walter. Towards verifiable FHE in practice: Proving correct execution of TFHE's bootstrapping using plonky2. Cryptology ePrint Archive, Report 2024/451, 2024. URL: https://eprint.iacr.org/2024/451.

[Via23] Alexander Viand. *Useable Fully Homomorphic Encryption*. PhD thesis, ETH Zurich, 2023.

[YZW06] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Privacy-preserving queries on encrypted data. In Dieter Gollmann, Jan Meier, and Andrei Sabelfeld, editors, *ESORICS 2006*, volume 4189 of *LNCS*, pages 479–495. Springer, Berlin, Heidelberg, September 2006. doi:10.1007/11863908_29.

[ZLL14] Feng Zhao, Chao Li, and Chun Feng Liu. A cloud computing security solution based on fully homomorphic encryption. In *16th International Conference on Advanced Communication Technology*, pages 485–488, 2014. doi:10.1109/ICACT.2014.6779008.