Cloud-Assisted Multiparty Computation from Fully Homomorphic Encryption

Adriana López-Alt
New York University

Eran Tromer*
Tel-Aviv University

Vinod Vaikuntanathan†
University of Toronto

December 7, 2011

Abstract

We construct protocols for secure multiparty computation with the help of a computationally powerful party, namely the "cloud". Our protocols are simultaneously efficient in a number of metrics:

- *Rounds:* our protocols run in 4 rounds in the semi-honest setting, and 5 rounds in the malicious setting.
- *Communication:* the number of bits exchanged in an execution of the protocol is *independent of the complexity of function f* being computed, and depends only on the length of the inputs and outputs.
- *Computation:* the computational complexity of all parties is *independent of the complexity of the function f*, whereas that of the cloud is linear in the size of the circuit computing f.

In the semi-honest case, our protocol relies on the "ring learning with errors" (RLWE) assumption, whereas in the malicious case, security is shown under the Ring LWE assumption as well as the existence of simulation-extractable NIZK proof systems and succinct non-interactive arguments. In the malicious setting, we also relax the communication and computation requirements above, and only require that they be "small" – polylogarithmic in the computation size and linear in the size of the joint size of the inputs.

Our constructions leverage the *key homomorphic* property of the recent fully homomorphic encryption scheme of Brakerski and Vaikuntanathan (CRYPTO 2011, FOCS 2011). Namely, these schemes allow combining encryptions of messages under different keys to produce an encryption (of the sum of the messages) under the sum of the keys. We also design an efficient, non-interactive *threshold decryption* protocol for these fully homomorphic encryption schemes.

*This work was partially supported by the Check Point Institute for Information Security and by the Israeli Centers of Research Excellence (I-CORE) program (center No. 4/11).

†This work was partially supported by an NSERC Discovery Grant, by DARPA under Agreement number FA8750-11-2-0225. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

Contents

1 Introduction

Cloud computing is usually portrayed as a user outsourcing its computation to a provider. However, in reality each cloud provider has numerous users, who may wish to compute with each other. How can multiple cloud users conduct an expensive joint computation, when they wish to keep private all information except the final output? They could send all their inputs to the cloud and ask it to compute the output, but this would violate their confidentiality. Alternatively, they could conduct a secure multiparty computation protocol (MPC) [GMW87, BGW88, CCD88] among themselves, and bear the full burden of computation. In particular, standard MPC protocols incur a large blow-up in the computational complexity of all the players involved, as well as a commensurate increase in the communication complexity of the protocol, neither of which are acceptable in the client-server setting. Can the presence of the cloud make the multiparty computation more efficient, without compromising its security?

More precisely, what we would like is an MPC protocol where n parties contribute inputs $x_1, \ldots, x_n$ respectively, and they jointly compute $f(x_1, \ldots, x_n)$ securely satisfying the following two properties:

- **Computational efficiency:** the running time of each party is polynomial in the length of its input and output. In particular, it should be independent of the complexity of the function f.
- **Communication efficiency:** the communication complexity of the protocol should be polynomial in the length of the inputs and outputs, but otherwise independent of the complexity of f.

Clearly, achieving computational efficiency alone requires the parties to seek the help of a (powerful) cloud capable of computing f. The protocol should be secure against a collusion of an arbitrary subset of parties together with the cloud.

The first possibility that comes to mind is to use a fully homomorphic encryption (FHE) scheme, which provides a mechanism to design communication-efficient protocols in the two-party setting. Starting from the groundbreaking work of Gentry [Gen09b], a number of progressively more efficient and simpler FHE constructions have surfaced [vDGHV10, SV10], most recently the works of Brakerski and Vaikuntanathan [BV11b, BV11a] and the work of Brakerski, Gentry and Vaikuntanathan [BGV11]. In the context of a single user outsourcing her computation to the cloud, the user can encrypt her input using her key under the fully homomorphic encryption scheme, and send the ciphertext to the server. The server homomorphically evaluates the desired function and returns the evaluated ciphertext back to the user who decrypts and gets the result of the computation.[1] Can we use fully homomorphic encryption in the *multiparty* setting, where the computation is performed jointly on the inputs of many players? A naïve way to approach this problem runs into the difficulty that we know of no way to operate on ciphertexts encrypted under different keys.

Gentry suggested an approach to do communication- and computation-efficient cloud-assisted MPC in the following way. The parties first run a multiparty coin-tossing protocol to generate a joint public key for (an arbitrary) FHE scheme, together with a secret sharing of the corresponding secret key (the i^{th} party gets the i^{th} share of the common secret key). Once this is done, the parties encrypt their inputs using the *common public key* and have the cloud compute an encryption of the result. They then run yet another MPC protocol to perform threshold decryption and recover the result. Clearly, since both the coin-tossing and the threshold decryption functionality are independent of the function f, this protocol is communication and computation-efficient (in the sense above). However, due to the use of generic MPC techniques, the protocol has a fairly large round-complexity and concrete computational complexity.

[1] For this discussion, we assume that the adversaries are semi-honest.

Can this state of affairs can be improved, by taking advantage of the algebraic structure that underlies many of the recent FHE schemes? We show this is indeed the case, by demonstrating concrete constructions.

1.1 Our Results

We show MPC protocols that are efficient in communication and computation as well as round complexity, using a special property of recent FHE constructions, namely *key homomorphism*. Informally, a key-homomorphic (public key) encryption scheme allows us to *deterministically* combine many public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$ into a combined public key pk, and simultaneously combine secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$ into a combined secret key sk. Informally, this property allows the parties to come up with a joint public key under which they can all encrypt their respective inputs, without resorting to an expensive coin-tossing protocol as in Gentry's solution [Gen09b, Gen09a]. Of course, for security, we require that semantic security under the joint public key pk holds even given $t < N$ secret keys sk_i. In a sense, sk_i form the secret shares of a common secret key sk whose corresponding public key is pk. The novelty is that a key homomorphic encryption scheme allows us to come up with this structure in a rather natural and efficient way.

Key-homomorphic encryption schemes abound. For example, consider Regev's public key encryption scheme [Reg09] where the public key $\mathsf{pk} = \mathbf{A}^\top \mathbf{s} + x$, where $\mathbf{A}$ is a matrix that forms a global system parameter, and the secret key vector $\mathsf{sk} = \mathbf{s}$. Adding together two such public keys $\mathsf{pk}_1 = \mathbf{A}^\top \mathbf{s}_1 + x_1$ and $\mathsf{pk}_2 = \mathbf{A}^\top \mathbf{s}_2 + x_2$ gives us $\mathsf{pk} = \mathsf{pk}_1 + \mathsf{pk}_2 = \mathbf{A}^\top(\mathbf{s}_1 + \mathbf{s}_2) + (x_1 + x_2)$, a joint public key whose corresponding secret key sk is the sum of sk_1 and sk_2. This structure is by no means unique to lattice based schemes, but it appears in numerous number-theoretic systems as well, e.g., the El Gamal encryption scheme. In lattice schemes, they are especially useful since they give us fully homomorphic encryption schemes that are key-homomorphic as well.

As we argued before, key-homomorphic schemes provide us with a natural avenue to remove the coin-tossing protocol from Gentry's schema for multiparty computation from FHE. In the context of FHE schemes, we also need a method of combining evaluation keys to produce a joint evaluation key which turns out to be trickier. In particular, we do not know of a nice non-interactive way of combining evaluation keys, but instead, we come up with a two-round protocol to do this.

Finally, the linear structure of these schemes give us a very efficient way to do *non-interactive threshold decryption*, similar to the work of Bendlin and Dåmgard [BD10].

These ideas are fairly general and can be instantiated with a number of different combinations of the existing schemes. We choose to present the instantiation that combines the ring LWE based somewhat homomorphic encryption scheme from [BV11b] with the re-linearization technique of [BV11a] and the noise management technique of [BGV11] (which, in turn, relies on the modulus reduction technique of [BV11a]). We chose this instantiation since it is quite efficient and simplest to present. The reader is referred to Section 6 for details on the instantiations. The resulting protocol is secure against semi-honest adversaries that corrupt any $t < N$ parties, and runs in 4 rounds.

We then compile this protocol to be secure against malicious adversaries, using simulation-extractable NIZK proof systems [SP92, Sah99, CLOS02] as well as succinct non-interactive argument systems [Mic94, GKR08, BCCT11, GLR11]. The resulting protocol runs in 5 rounds.

Our main theorem is the following:

Theorem 1.1. *There are communication- and computation-efficient MPC protocols that are:*

- *secure against semi-honest corruptions of an arbitrary subset of players, under the Ring LWE assumption and runs in* 4 *rounds. The communication and computation of the parties is inde-*

pendent of the complexity of f, and the computation time of the server is linear in the size of the circuit computing f.

- *secure against malicious adversaries that corrupt an arbitrary number of players, under the Ring LWE assumption, the existence of simulation-extractable NIZK proof systems and succinct non-interactive argument systems. The protocol runs in 5 rounds. The communication of the parties is polylogarithmic in the size of the circuit computing f, their computation time is nearly linear in the total size of the inputs, and the computation time of the server is polynomial in the size of the circuit computing f.*

1.2 Other Related Work

The basic idea of using homomorphic encryption schemes in conjunction with threshold decryption to boost the efficiency of MPC protocols was first noticed by Cramer, Damgård and Nielsen [CDN01]. The idea of using a cloud to alleviate the computational efforts of parties was recently explored in the work on "server-aided MPC" by Kamara, Mohassel and Raykova [KMR11]. Their protocols, however, require some of the parties to do a large amount of computation, essentially proportional to the size of the function f being computed. Thus, their protocols are not computation-efficient in our sense. The works of Myers, Sergi and shelat [MSas11], Bendlin et.al [BDOZ11], and Damgård et. al. [DPSZ11] construct MPC protocols (in the classical sense, without the help of a cloud) using FHE schemes (in the case of [MSas11]) and somewhat homomorphic schemes (in [BDOZ11, DPSZ11]). Their protocols are communication-efficient, but not computation-efficient in our sense, namely the computational complexity of the parties is proportional to the size of f. Halevi, Lindell and Pinkas [HLP11] recently considered the model of "secure computation on the web" wherein the goal is to minimize interaction between the parties. Unfortunately, as they show, their notion can only be achieved for a small class of functions. In contrast, we focus here on designing MPC protocols for arbitrary functions, of course at the cost of interaction.

2 Preliminaries

Notation. For an integer n, we use the notation $[n]$ to denote the set $[n] = \{1, \ldots, n\}$. For a randomized function f, we write $f(x\ ; r)$ to denote the unique output of f on input x with random coins r. We write $f(x)$ to denote a random variable for the output of $f(x; r)$ over the random coins r. For a distribution or random variable X, we write $x \leftarrow X$ to denote the operation of sampling a random x according to X. For a set S, we overload notation and use $s \leftarrow S$ to denote sampling s from the uniform distribution over S. For two distributions, X and Y, we use $X \overset{c}{\approx} Y$ to mean that X and Y are computationally indistinguishable, and $X \overset{s}{\approx} Y$ to mean that they are statistically close. Finally, we use $y := f(x)$ to denote the *deterministic* evaluation of f on input x with output y.

Fully Homomorphic Encryption. We review the definition of (leveled) fully homomorphic encryption. We state the definition of [BGV11], which is a relaxation of the original definition in [Gen09b]. The main difference is that the definition in [Gen09b] requires that schemes $\mathcal{E}^{(D)}$ that evaluate circuits of depth at most D, all share the same decryption circuit. However, as in [BGV11], we relax this definition and let all algorithms (including decryption) depend on the depth D of the circuit that is evaluated.

Definition 2.1 ($\mathcal{C}$-Homomorphic Encryption [Gen09b])**.** *For a class of circuits $\mathcal{C}$, we say that an encryption scheme $\mathcal{E} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ is* $\mathcal{C}$-homomorphic *if:*

- $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa)$, *for security parameter* κ, *outputs public parameters* params. *All other algorithms,* $\mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}$ *implicitly take* params *as input, even when not explicitly stated.*
- $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{Keygen}(\mathsf{params})$, *for a security parameter* κ, *outpus a public key* pk, *a secret key* sk, *and a (public) evaluation key* ek.
- $m' \leftarrow \mathsf{Dec}_{\mathsf{sk}}(c)$, *given a secret key* sk *and a ciphertext* c, *outputs a message* m'.
- $\gamma := \mathsf{Eval}(C, c_1, \ldots, c_\ell)$, *given a (description of) a circuit* C *along with an evaluation key* pk *and* ℓ *ciphertexts* $c_1, \ldots, c_\ell$, *outputs a ciphertext* γ.

We require that for all $c \in \mathcal{C}$, *all* $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{Keygen}(1^\kappa)$ *and all plaintexts* $(m_1, \ldots, m_\ell)$ *and ciphertexts* $(c_1, \ldots, c_\ell)$ *such that* c_i *is in the support of* $\mathsf{Enc}_{\mathsf{pk}}(m_i)$, *if* $\gamma := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_\ell)$, *then* $\mathsf{Dec}_{\mathsf{sk}}(\gamma) = C(m_1, \ldots, m_\ell)$.

Definition 2.2 (Leveled Fully Homomorphic Encryption [Gen09b])**.** *Let* $\mathcal{C}^{(D)}$ *be the class of all circuits of depth at most* D *(that use some specified complete set of gates). We say that a family of homomorphic encryption schemes* $\{\mathcal{E}^{(D)} : D \in \mathbb{Z}^+\}$ *is leveled fully homomorphic if, for all* $D \in \mathbb{Z}^+$, *it satisfies the following properties:*

Correctness: $\mathcal{E}^{(D)}$ *is* $\mathcal{C}^{(D)}$*-homomorphic.*

Compactness: *The computational complexity of* $\mathcal{E}^{(D)}$*'s algorithms is polynomial (the same polynomial for all* D*) in the security parameter,* D, *and (in the case of the evaluation algorithm) the size of the circuit.*

From now on, when we say *fully homomorphic*, we refer to *leveled fully homomorphic encryption.*

Functionalities. We review the following basic MPC functionalities, which we will use when describing our cloud-assisted MPC protocol.

- $\mathcal{F}_f$, the functionality computing a function f. See Figure 1.
- $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$, the CRS functionality that outputs a common reference string distributed according to a predetermined distribution $\mathcal{D}$. See Figure 2.
- $\mathcal{F}_{\text{BC}}$, the Broadcast functionality that upon receiving a message from a party P_i, forwards the message to *all* parties. See Figure 3.
- $\mathcal{F}_{\text{ZK.BC}}^{R}$, the "Prove-and-Broadcast" functionality $\mathcal{F}_{\text{ZK.BC}}^{R}$ that upon receiving a statement and witness pair (x, w) from a party P_i, verifies that $R(x, w) = 1$ (for some predetermined NP-relation R) and if so, sends x to all parties. See Figure 4.

Functionality $\mathcal{F}_f$

$\mathcal{F}_f$ interacts with parties $P_0, P_1, \ldots, P_N$ and adversary $\mathcal{S}$, and is parametrized by an N-input function f.

Upon receiving (sid, i, x_i) from P_i for all $i \in [N]$ (and no input from P_0), compute $y = f(x_1, \ldots, x_N)$, send (sid, y) to all clients, the server, and the adversary $\mathcal{S}$, and halt.

Figure 1: Functionality $\mathcal{F}_f$

3 Cloud-Assisted MPC

We consider the problem of N parties who wish to securely compute a joint function of their inputs, but want to do so in a way such that the amount of data they interchange, as well as their computation

Functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$

$\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ interacts with parties $P_0, P_1, \ldots, P_N$ and adversary $\mathcal{S}$, and is parametrized by an algorithm $\mathcal{D}$.

Upon receiving (sid, i) from P_i, compute $\mathsf{crs} \leftarrow \mathcal{D}(1^\kappa)$, and send $(\mathsf{sid}, i, \mathsf{crs})$ to P_i and the adversary. Upon receiving subsequent messages (sid, j) from P_j, send $(\mathsf{sid}, j, \mathsf{crs})$ to P_j and the adversary $\mathcal{S}$, and halt.

Figure 2: CRS Functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$

Functionality $\mathcal{F}_{\text{BC}}$

$\mathcal{F}_{\text{BC}}$ interacts with parties $P_0, P_1, \ldots, P_N$ and adversary $\mathcal{S}$.

Upon receiving (sid, i, m) from P_i, send (sid, i, m) to all parties in $P_1, \ldots, P_N$ and the adversary $\mathcal{S}$, and halt.

Figure 3: Broadcast Functionality $\mathcal{F}_{\text{BC}}$

Functionality $\mathcal{F}_{\text{ZK.BC}}^{R}$

$\mathcal{F}_{\text{ZK.BC}}^{R}$ interacts with parties $P_0, P_1, \ldots, P_N$ and adversary $\mathcal{S}$, and is parametrized by an NP-relation R.

Upon receiving (sid, i, x, w) from P_i: If $R(x, w) = 1$, send (sid, i, x) to all parties and the adversary $\mathcal{S}$, and halt. Otherwise, halt.

Figure 4: "Prove-and-Broadcast" Functionality $\mathcal{F}_{\text{ZK.BC}}^{R}$

time, is independent of the complexity of the function. In order to do this, we rely on the computation power of a new party called the *server* or *cloud*, who will perform the computation of the function f. The computation should remain secure even though the server is untrusted. We formalize this idea below.

Definition. For an N-input function f, we define a *cloud-assisted multiparty protocol* Π for f to be a protocol between N interactive Turing Machines $P_1, \ldots, P_N$, called *clients*, and a *server* S, also an interactive Turing Machine, such that for all $\vec{x} = (x_1, \ldots, x_N)$, the output of Π in an execution where P_i is given x_i as input (and the server S does not receive an input), is $y = f(\vec{x})$. We sometimes use P_0 to denote the server S. What differentiates a cloud-assisted MPC protocol from a standard MPC protocol is that we require the communication complexity of the protocol, as well as the computation time of the clients $P_1, \ldots, P_N$ to be *independent* of the complexity of the function f. On the other hand, the computation time of the server S must be *linear* in the circuit-size of f. [2]

Security. We prove security of a cloud-assisted MPC protocol in the Ideal/Real paradigm (in the standalone setting). In the *ideal world*, the computation of f is performed through a trusted functionality $\mathcal{F}_f$ that receives input x_i from each party P_i, computes $y = f(x_1, \ldots, x_N)$ and gives y to all parties $P_1, \ldots, P_N$, and the server S. It is clear that in the ideal world, the only information that any party learns is its own input and the output y. In the *real world*, however, this trusted functionality does not exist and so in order to compute $y = f(x_1, \ldots, x_N)$, parties $P_1, \ldots, P_N$ and

[2] When considering security against semi-honest adversaries, we achieve precisely this. However, to tolerate malicious adversaries, we relax the definition and require only that the communication and computation time of the clients be "small", much less than linear in the size of the circuit they are computing. Furthermore, we let the computation time of the server be polynomial in the size of the circuit.

server S run a protocol Π. Figure 5 gives a pictorial representation of the ideal world and the real world.

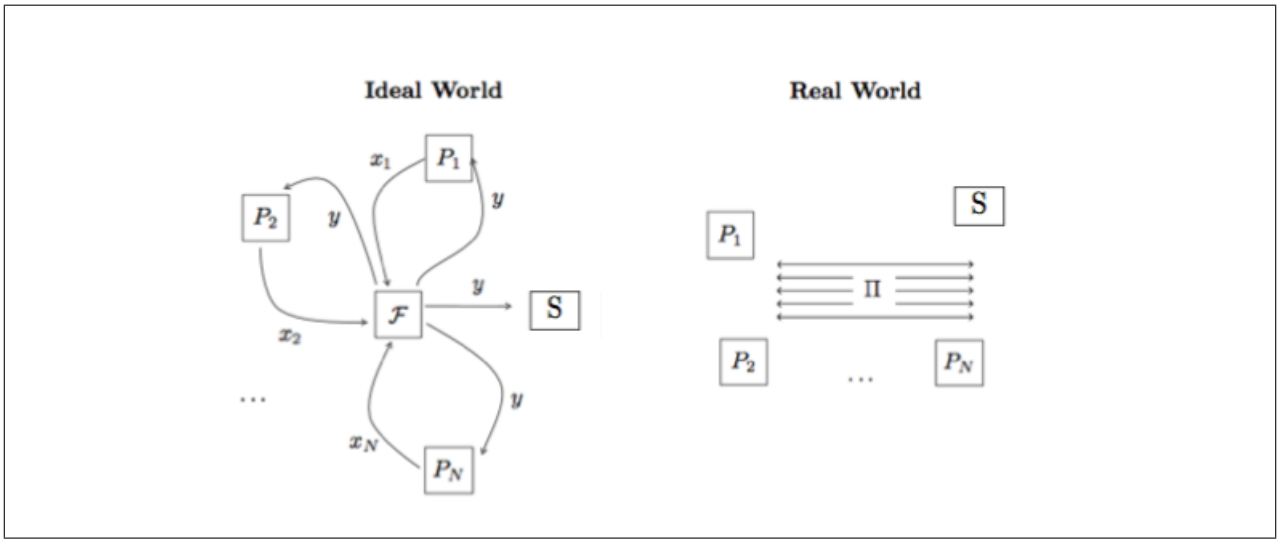

Figure 5: Ideal World vs. Real World

In either world, an adversary corrupting $t < N$ parties, and perhaps additionally the server S, receives all messages directed to the corrupted parties and controls the messages sent by them. We use $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\vec{x})$ to denote the joint output of the ideal-world adversary $\mathcal{S}$ and of parties $P_1, \ldots, P_N$ and server P_0 in an ideal execution with functionality $\mathcal{F}$ and inputs $\vec{x} = (x_1, \ldots, x_N)$. Similarly, we use $\text{REAL}_{\Pi,\mathcal{A}}(\vec{x})$ to denote the joint output of the real-world adversary $\mathcal{A}$ and the outputs of parties $P_1, \ldots, P_N$ and server P_0 in an execution of protocol Π with inputs $\vec{x} = (x_1, \ldots, x_N)$. We say that a protocol Π *securely realizes* $\mathcal{F}$ if for every real-world adversary $\mathcal{A}$, there exists an ideal-world adversary $\mathcal{S}$ such that

$$\text{IDEAL}_{\mathcal{F},\mathcal{S}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\Pi,\mathcal{A}}(\vec{x}),$$

4 Cloud-Assisted MPC From Key-Homomorphic Threshold FHE

In this section, we define a new primitive called *key-homomorphic threshold FHE* and show that it can be used to construct a cloud-assisted MPC protocol for any function f. Informally, a key-homomorphic threshold FHE scheme allows combining public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$ into a combined public key pk, evaluation keys $\mathsf{ek}_1, \ldots, \mathsf{ek}_N$ into a combined evaluation key ek, and secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$ into a combined secret key sk. The correctness property we require for this key combination is that if for all i, $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ is in the support of Keygen, then $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek})$ is also in the support of Keygen. In other words, if for all i, pk_i and ek_i are *valid* public and evaluation keys for secret key sk_i, then pk and ek are valid public and evaluation keys for sk. For security, we require semantic security under the joint public key pk even given $t < N$ secret keys sk_i. We additionally require that the combined evaluation key ek does not reveal any information about the individual evaluation and secret keys $\mathsf{ek}_i, \mathsf{sk}_i$. We formalize this by requiring that there exist an algorithm $\mathsf{SimCombKeys}$ that given any number of public/evaluations key pairs $\{(\mathsf{pk}_i, \mathsf{ek}_i)\}_{i \in T}$, can simulate the remaining public/evaluations key pairs $\{(\mathsf{pk}_j, \mathsf{ek}_j)\}_{j \in [N] \setminus T}$ and the joint evaluation key ek.

A key-homomorphic threshold FHE scheme also has threshold decryption properties that allow for decryption of a ciphertext under the combined public key pk by independently using the secret keys sk_i. More formally, given a ciphertext c encrypting a plaintext m under pk, each party P_i can independently

compute a decryption share μ_i using sk_i. Combining the ciphertext c with the shares $\mu_1, \ldots, \mu_N$ will yield the plaintext m. For security, we require that the share μ_i does not reveal anything about the secret key sk_i. We formalize this by requiring that there exist an algorithm $\mathsf{SimShareDec}$ that given a ciphertext c, a plaintext m, and any number of shares $\{\mu_i\}_{i \in T}$, can simulate the remaining shares $\{\mu_j\}_{j \in [N] \setminus T}$, such that when all shares are combined, they decrypt c to m.

Finally, we require ciphertexts of a key-homomorphic threshold FHE scheme to be re-randomizable. In other words, we require that there exists an algorithm ReRand that given the public key pk and ciphertext $c := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_\ell)$, where $c_1, \ldots, c_\ell$ are ciphertexts encrypting plaintexts $m_1, \ldots, m_\ell$ under pk, outputs a randomization $\hat{c}$ of c that is indistinguishable from a *fresh* encryption of $y = C(m_1, \ldots, m_\ell)$, for all circuits C. We require this property because we can only guarantee security of threshold decryption (i.e., that the shares do not reveal information about the individual secret keys) if the ciphertext c to be decrypted is a fresh encryption of y. Unfortunately, since our instantiation of key-homomorphic threshold FHE is lattice-based, the only way we know how randomize ciphertexts is by adding to c noise that is super-polynomially larger than the noise in the original ciphertexts c_i. Because of this, we need to define the encryption algorithm in key-homomorphic threshold FHE to take an extra bit mode $\in \{0, 1\}$ that decides the magnitude of the noise to be used in the encryption. There is no inherent reason why we need to work over two types of ciphertexts, this is just a technical artifact of our instantiation.

Our instantiation will also require the use of a common reference string, crs, that will be partitioned into N parts, $(\mathsf{crs}_1, \ldots, \mathsf{crs}_N)$, so that crs_i corresponds to party P_i. The public and evaluation keys for P_i will depend on crs_i. We model this by having the Setup algorithm take an additional input N and output crs as part of the public parameters, and having Keygen take an additional input, $i \in [N]$. Again, there is no inherent reason why we need these additional inputs to Setup and Keygen, this is just a technical artifact of our instantiation.

Definition 4.1. *A* public-key key-homomorphic threshold FHE *scheme $\mathcal{E}$ with secret, public, and evaluation key-spaces $\mathcal{SK}, \mathcal{PK}, \mathcal{EK}$ and message-space $\mathcal{M}$ is a tuple of algorithms* ($\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}, \mathsf{CombinePK}, \mathsf{CombineKeys}, \mathsf{ShareDec}, \mathsf{ShareCombine}, \mathsf{ReRand}$) *with the following syntax:*

- $\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Dec}$ *and* Eval *are as in a standard FHE. We only require that* Setup *takes an additional input N and* Keygen *takes an additional input i.*

From here on, all algorithms implicitly take as input params*, where* $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa, N)$*, even if not explicitly stated.* $\mathsf{Keygen}, \mathsf{Dec}$ *and* Eval *also implicitly take* params *as input.*

- $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m, \mathrm{mode})$*. Takes as input a public key* pk*, a message $m \in \mathcal{M}$, and a bit* $\mathrm{mode} \in \{0, 1\}$*, and outputs a ciphertext c. We will work with two types of ciphertexts; the bit* mode *will decide what type of ciphertext to output. Most of the time we will work with ciphertexts generated by* $\mathsf{Enc}(\cdot, 0)$*, and will only use* $\mathsf{Enc}(\cdot, 1)$ *in re-randomization (see below). When unspecified, we assume* $\mathrm{mode} = 0$*.*
- $\mathsf{pk} := \mathsf{CombinePK}(\{\mathsf{pk}\}_{i \in [N]})$*. Takes as input N public keys* $\{\mathsf{pk}_i\}_{i \in [N]}$*, and outputs a combined public key* pk*.*
- $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{CombineKeys}(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i \in [N]})$*. Takes as input N key pairs* $\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i \in [N]}$*, and outputs combined secret key* sk *and evaluation key* ek*.*
- $\mu_i \leftarrow \mathsf{ShareDec}_{\mathsf{sk}_i}(c)$*. Takes as input secret key* sk_i *and a ciphertext c and outputs a decryption share μ_i.*
- $m \leftarrow \mathsf{ShareCombine}(c, \mu_1, \ldots, \mu_N)$*. Takes as input a ciphertext c and N decryption shares $(\mu_1, \ldots, \mu_N)$ and outputs a plaintext m.*
- $\hat{c} \leftarrow \mathsf{ReRand}_{\mathsf{pk}}(c)$*. Takes as input a public key* pk *and ciphertext c and outputs a rerandomization $\hat{c}$ of c.*

Correctness and security are given the following properties. Define the relation

$$R^{(keys)} = \left\{ \left(\mathsf{params},\ \mathsf{pk}_i\ ,\ \mathsf{ek}_i\ ,\ i\ \right)\ ,\ \left(\ \mathsf{sk}_i\ ,\ r_i{}^{(gen)}\ \right)\ \mid\ (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i) := \mathsf{Keygen}(\mathsf{params}, i\ ; r_i{}^{(gen)}) \right\}$$

and say that the public/evaluation key pair $(\mathsf{pk}_i, \mathsf{ek}_i)$ *is* valid *for secret key* sk_i *with respect to* params*, if there exists* $r_i{}^{(gen)}$ *such that* $R^{(keys)}((\mathsf{params}, \mathsf{pk}_i, \mathsf{ek}_i, i)\ ,\ (\mathsf{sk}, r_i{}^{(gen)})) = 1$*. We say that* $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ *is a* valid key tuple *with respect to* params *if* $(\mathsf{pk}_i, \mathsf{ek}_i)$ *is a valid public/evaluation key pair for* sk_i *with respect to* params*. When* params *is clear from context, we simply say that* $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ *is a* valid key tuple*.*

Correct Key Combination: *If* $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}$ *are all valid key tuples,* $\mathsf{pk} := \mathsf{CombinePK}(\{\mathsf{pk}_i\}_{i\in[N]})$ *and* $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{CombineKeys}(\{\mathsf{sk}_i\}_{i\in[N]}, \{\mathsf{ek}_i\}_{i\in[N]})$ *then* $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek})$ *is a valid key tuple.*

Key-Simulation Indistinguishability: *There exists a* PPT *simulator* SimCombKeys *such that for any set* $T \subsetneq [N]$ *with* $t = |T|$ *and* $\overline{T} = [N]\backslash T$*,* $(\mathsf{ek}, \{(\mathsf{pk}_j, \mathsf{ek}_j)\}_{j\in\overline{T}}) \leftarrow \mathsf{SimCombKeys}(\{\mathsf{pk}_i, \mathsf{ek}_i\}_{i\in T})$ *takes as input* t *public/evaluation key pairs* $\{\mathsf{pk}_i, \mathsf{ek}_i\}_{i\in T}$ *and outputs* $N-t$ *public/evaluation key pairs* $\{(\mathsf{pk}_j, \mathsf{ek}_j)\}_{j\in\overline{T}}$ *and an evaluation key* ek*, satisfying the following.*

> *For all valid key tuples* $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}$ *and all subsets* $T \subsetneq [N]$*, the following two distributions are computationally indistinguishable:*
>
> $$\left(\mathsf{params}, \mathsf{SimCombKeys}(\{(\mathsf{pk}_i, \mathsf{ek}_i)\}_{i\in T})\right) \overset{c}{\approx} \left(\mathsf{params}, \mathsf{ek}, \{(\mathsf{pk}_j, \mathsf{ek}_j)\}_{j\in\overline{T}}\right)$$
>
> *where* $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa), \{(\cdot, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa)\}_{j\in\overline{T}}$*, and* $(\cdot, \mathsf{ek}) \leftarrow \mathsf{CombineKeys}(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]})$*.*

Semantic Security: *For all* $T \subsetneq N$*, every* $\mathrm{mode} \in \{0, 1\}$*, and all* PPT *adversaries* $\mathcal{A}$*, the advantage of* $\mathcal{A}$ *in the following game is negligible (in* κ*).*

> **Key Generation:** *The challenger runs* $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa)$*. The adversary* $\mathcal{A}$ *then sends* $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in T}$ *to the challeger. The challenger runs* $(\mathsf{pk}_j, \mathsf{sk}_j, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j)$ *for* $j \in \overline{T} = [N]\backslash T$*, computes* $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{CombineKeys}(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]})$ *and* $\mathsf{pk} := \mathsf{CombinePK}(\{\mathsf{pk}_i\}_{i\in[N]})$*, and sends* $\left(\mathsf{params}, \mathsf{ek}, \{(\mathsf{pk}_j, \mathsf{ek}_j)\}_{j\in\overline{T}}\right)$ *to* $\mathcal{A}$*.*
>
> **Challenge:** $\mathcal{A}$ *sends plaintexts* m_0, m_1 *to the challenger, who chooses* $b \leftarrow \{0, 1\}$*, and sends* $c^* = \mathsf{Enc}_{\mathsf{pk}}(m_b, \mathrm{mode})$ *to* $\mathcal{A}$*.*
>
> **Output:** $\mathcal{A}$ *outputs* b'*.*
>
> *We define the advantage of* $\mathcal{A}$ *to be* $\left|\frac{1}{2} - \Pr[b' = b]\right|$*.*

Correct Share Decryption of Combined Key: *For all valid key tuples* $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}$*, and all* $c \in \mathcal{C}_{\mathsf{pk}}$*, where* $\mathsf{pk} = \mathsf{CombinePK}(\{\mathsf{pk}_i\}_{i\in[N]})$ *and* $\mathcal{C}_{\mathsf{pk}}$ *is the ciphertext-space defined by* pk*,*

$$\mathsf{Dec}_{\mathsf{sk}}(c) = \mathsf{ShareCombine}(c, \mathsf{ShareDec}_{\mathsf{sk}_1}(c), \ldots, \mathsf{ShareDec}_{\mathsf{sk}_N}(c))$$

Share-Simulation Indistinguishability: *There exists a* PPT *simulator* SimShareDec*, where* $\{\mu_j\}_{j\in\overline{T}} \leftarrow \mathsf{SimShareDec}(c, m, \{\mu_i\}_{i\in T})$ *takes as input a ciphertext* c*, a plaintext* m*, and* t *decryption shares* $\{\mu_i\}_{i\in T}$ *(where* $T \subsetneq [N]$ *is any subset of cardinality* $t < N$*) and outputs* $N - t$ *decryption shares* $\{\mu_j\}_{j\in\overline{T}}$*, satisfying the following.*

For all valid key tuples $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}$*, all subsets* $T \subsetneq [N]$*, all* $m \in \mathcal{M}$*, and all random strings* $\{r_i^{(dec)}\}_{i\in T}$*, the following two distributions are computationally indistinguishable:*

$$\left(\mathsf{params}, c, \mathsf{SimShareDec}(c, m, \{\mu_i\}_{i\in T})\right) \overset{c}{\approx} \left(\mathsf{params}, c, \{\mu_j\}_{j\in\overline{T}}\right),$$

where $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa), c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$ *and for* $i \in T$, $\mu_i := \mathsf{ShareDec}_{\mathsf{sk}_i}(c\,; r_i^{(dec)})$.

Ciphertext Re-Randomization: *For all circuits* C*, all plaintexts* $(m_1, \ldots, m_\ell)$*, all random strings* $(r_1, \ldots, r_\ell)$*, the following distributions are statistically close:*

$$(\mathsf{params}, c_1, \ldots, c_t, \mathsf{ReRand}_{\mathsf{pk}}(c)) \overset{s}{\approx} (\mathsf{params}, c_1, \ldots, c_t, \mathsf{Enc}_{\mathsf{pk}}(y, 1)),$$

where $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa), c_i := \mathsf{Enc}_{\mathsf{pk}}(m_i, 0; r_i)$ *for all* $i \in [\ell]$, $c := \mathsf{Eval}_{\mathsf{ek}}(c_1, \ldots, c_\ell)$ *and* $y = C(m_1, \ldots, m_\ell)$.

4.1 Construction

Before presenting our construction of cloud-assisted MPC from key-homomorphic threshold FHE, we describe two functionalities, $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ (Figure 6) and $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$ (Figure 7) computing the CombineKeys and ReRand algorithms, respectively. For clarity, we will present our construction in the $(\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}, \mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}})$-hybrid model, as this allows us to separate the messages that are sent for the purpose of computing f from the messages sent for the purpose of combining keys and from those sent for re-randomization. In Section 7, we show protocols implementing $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ and $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$ for our specific instantiation of key-homomorphic threshold FHE.

Functionality $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$

$\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ interacts with parties $P_0, P_1, \ldots, P_N$ and adversary $\mathcal{S}$, and is parametrized by a key-homomorphic threshold FHE scheme $\mathcal{E}$ with key-combination algorithm CombineKeys. Parties $P_1, \ldots, P_N$ get as input a valid key tuple $(\mathsf{sk}_i, \mathsf{ek}_i)$, and for all $j \neq i$, P_i also receives ek_j. Server P_0 does not receive an input. For session ID sid:

> Upon receiving $(\mathsf{sid}, i, \mathsf{sk}_i, \mathsf{ek}_i)$ from P_i for all $i \in [N]$ (and no input from P_0), compute $(\mathsf{sk}, \mathsf{ek}) = \mathsf{CombineKeys}(\{\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]})$, and send $(\mathsf{sid}, \mathsf{ek}, \{(\mathsf{ek}_i)\}_{i\in[N]})$ to all parties $P_0, P_1, \ldots, P_N$ and the adversary $\mathcal{S}$.

Figure 6: "Key-Combination" Functionality $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$

Functionality $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$

$\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$ interacts with parties $P_0, P_1, \ldots, P_N$ and adversary $\mathcal{S}$. It is parametrized by a PK-FHE scheme $\mathcal{E}$ with ciphertext re-randomization algorithm ReRand. Server P_0 gets as input a ciphertext c. For session ID sid:

> Upon receiving $(\mathsf{sid}, 0, c)$ from P_0, compute $\hat{c} = \mathsf{ReRand}(c)$, and send $(\mathsf{sid}, c, \hat{c})$ to all parties $P_0, P_1, \ldots, P_N$ and the adversary $\mathcal{S}$.

Figure 7: "Ciphertext Re-Randomization" Functionality $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$

We describe our cloud-assisted MPC protocol Π^{SH} in detail in Figure 8. The basic idea behind our construction is essentially the same as in [Gen09b]: the parties run an MPC protocol to agree on

a joint public key pk and a joint evaluation key ek. The parties then encrypt their inputs x_i under pk, and the server homomorphically evaluates the function f on the encrypted inputs $(x_1, \ldots, x_N)$ using the joint evaluation key ek. Finally, the parties run an MPC protocol to threshold decrypt the output. What key-homomorphic threshold FHE gives us in comparison to the generic approach of [Gen09b] is *round efficiency*. Essentially, the joint public key can be computed deterministically from all public keys, the joint evaluation key ek, obtained from $\mathsf{CombineKeys}$, can be computed in 2 rounds, and threshold decryption can be computed in 1 round: the parties simply compute decryption shares μ_i. It is not yet aparent from the description of the protocol why we can achieve only 4 rounds of interaction since we only given a description of our protocol in the $(\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}, \mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}})$-hybrid model. We explain this fully in Section 8, but the intuition behind it is that we can send many of the messages for computing f, the joint evaluation key and the re-randomization of c *in parallel*.

Protocol Π^{SH}

Let C be a circuit computing f. Let $\mathcal{E} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{CombinePK}, \mathsf{CombineKeys}, \mathsf{ShareDec}, \mathsf{ShareCombine}, \mathsf{ReRand})$ be a key-homomorphic threshold FHE scheme.

For session ID sid:

Phase 1: Each P_i calls $\mathcal{F}^{\mathcal{D}}_{\text{CRS}}$ with (sid, i) and receives params. It chooses an encryption key tuple, $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i) \leftarrow \mathsf{Keygen}(1^\kappa, i)$. P_i then sends $(\mathsf{sid}, i, (\mathsf{sk}_i, \mathsf{ek}_i))$ to $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$, and $(\mathsf{sid}, i, \mathsf{pk}_i)$ to $\mathcal{F}_{\text{BC}}$.

Phase 2: Upon receiving $(\mathsf{sid}, \mathsf{ek}, \{\mathsf{ek}_i\}_{i\in[N]})$ from $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$, and $(\mathsf{sid}, i, \mathsf{pk}_i)$ from $\mathcal{F}_{\text{BC}}$ for all $i \in [N]$, each P_i computes $\mathsf{pk} = \mathsf{CombinePK}(\{\mathsf{pk}_i\}_{i\in[N]})$, and $c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_i)$ and sends (sid, i, c_i) to $\mathcal{F}_{\text{BC}}$.

Phase 3: Upon receiving (sid, i, c_i) from $\mathcal{F}_{\text{BC}}$ for all $i \in [N]$ and $(\mathsf{sid}, \mathsf{ek}, \{\mathsf{ek}_i\}_{i\in[N]})$ from $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$, server computes $c := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_N)$ and sends $(\mathsf{sid}, 0, c)$ to $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$.

Phase 4: Upon receiving $(\mathsf{sid}, c, \hat{c})$ from $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$, each P_i computes $\mu_i := \mathsf{ShareDec}_{\mathsf{sk}_i}(\hat{c})$ and sends (sid, i, μ_i) to $\mathcal{F}_{\text{BC}}$.

Local Computation: Upon receiving (sid, i, μ_i) from $\mathcal{F}_{\text{BC}}$ for all $i \in [N]$, each P_i computes and outputs $y := \mathsf{ShareCombine}(\hat{c}, \mu_1, \ldots, \mu_N)$.

Figure 8: Protocol Π^{SH}. Implements $\mathcal{F}$ in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}}, \mathcal{F}^{\mathcal{E}}_{\text{KEYS}}, \mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}})$-hybrid model, where $\mathcal{D}$ is the output distribution of $\mathsf{Setup}(1^\kappa, N)$. Secure against semi-honest adversaries.

Lemma 4.1. *Let f be an N-input function, and let $\mathcal{F}$ be the ideal functionality computing f. Let $\mathcal{E}$ be a key-homomorphic threshold FHE scheme. Then Π^{SH} described in Figure 8 securely implements $\mathcal{F}$ against (static) semi-honest adversaries, in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}}, \mathcal{F}^{\mathcal{E}}_{\text{KEYS}}, \mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}})$-hybrid model.*

Looking ahead, we will instantiate key-homomorphic threshold FHE based on the PLWE assumption, a special case of the "Ring LWE" assumption of [LPR10] (see Section 6). Combining Lemma 4.1 with protocols implementing $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ and $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$ for this instantiation (shown in Section 7), we arrive at the following theorem.

Theorem 4.2. *Assuming the hardness of $\mathsf{PLWE}_{f,q,\chi}$ for parameters f, q, χ as described in Section 6.1, there exists a protocol that securely implements $\mathcal{F}$ against (static) semi-honest adversaries, in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}})$-hybrid model.*

4.2 Security Under Semi-Honest Adversaries

Proof of Lemma 4.1: Correctness follows from correctness of key combination, correctness of homomorphic evaluation, and correctness of share decryption of $\mathcal{E}$.

We now turn to security. Let $\mathcal{A}$ be a real-world semi-honest adversary corrupting $t < N$ clients, and possibly the server. Let $T \subsetneq [N]$ be the set of corrupted clients. We describe the simulator $\mathcal{S}^{\text{SH}}$ in Figure 9.

Simulator $\mathcal{S}^{\text{SH}}$

1. Send $\{x_i\}_{i\in T}$ to $\mathcal{F}$ and receive y.
2. Run $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa)$. Run $\mathcal{A}$ on inputs $\{x_i\}_{i\in T}$ and params, and play the role of the honest parties:

   Phase 1: Upon receiving $(\mathsf{sid}, i, (\mathsf{sk}_i, \mathsf{ek}_i))$ as a message to $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ and $(\mathsf{sid}, i, \mathsf{pk}_i)$ as a message to $\mathcal{F}_{\text{BC}}$ for all $i \in T$, run $(\mathsf{ek}, \{(\mathsf{pk}_j, \mathsf{ek}_j)\}_{j\in\overline{T}}) \leftarrow \mathsf{SimCombKeys}(\{(\mathsf{pk}_i, \mathsf{ek}_i)\}_{i\in T})$ and $\mathsf{pk} := \mathsf{CombineKeys}(\{\mathsf{pk}_i\}_{i\in[N]})$, and send $(\mathsf{sid}, \mathsf{ek}, \{\mathsf{ek}_i\}_{i\in[N]})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$.

   Phase 2: For $j \in \overline{T}$, compute $c_j \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$ and send (sid, j, c_j) to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$.

   Phase 3: If server is *not* corrupted, compute $c := \mathsf{Eval}(C, c_1, \ldots, c_N)$. Otherwise, receive $(\mathsf{sid}, 0, c)$ from $\mathcal{A}$ as a message to $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$. In either case, compute $\hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(y, 1)$ and send $(\mathsf{sid}, c, \hat{c})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$.

   Phase 4: Upon receiving (sid, i, μ_i) from $\mathcal{A}$ as a message to $\mathcal{F}_{\text{BC}}$ for all $i \in T$, compute $\{\mu_j\}_{j\in\overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, y, \{\mu_i\}_{i\in T})$. For $j \in \overline{T}$, send (sid, j, μ_j) on behalf of $\mathcal{F}_{\text{BC}}$.
3. Output $\mathcal{A}$'s output.

Figure 9: Simulator $\mathcal{S}^{\text{SH}}$ for (static) semi-honest adversary $\mathcal{A}$ corrupting clients $T \subsetneq [N]$ and possibly server S.

We prove that $\text{REAL}_{\Pi^{\text{SH}},\mathcal{A}}(\vec{x}) \overset{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S}^{\text{SH}}}(\vec{x})$ via a series of hybrids. The view of the $\mathcal{A}$ consists of

$$\mathsf{View} = \left(\mathsf{params}, \{x_i, (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i), c_i, \mu_i\}_{i\in T}, \mathsf{pk}, \mathsf{ek}, \{(\mathsf{pk}_j, \mathsf{ek}_j)\}_{j\in\overline{T}}, \{c_j\}_{j\in\overline{T}}, c, \hat{c}, \{\mu_j\}_{j\in\overline{T}}\right)$$

We simply prove that for all $\{x_i, (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i), c_i, \mu_i\}_{i\in T}$ output by $\mathcal{A}$, the joint distribution of these variables in the simulation is computationally indistinguishable from the joint distribution of the same variables in a real-world execution of protocol Π^{SH}. We let View_i be the joint distribution of these variables in hybrid i. In all hybrids, $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa, N)$, $c := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_N)$ (this is true even when the server is corrupted since the adversary is semi-honest), and $\mathsf{pk} := \mathsf{CombinePK}(\{\mathsf{pk}_i\}_{i\in N}$, but we gradually change how each of the other variables is computed.

Hybrid 0: This is the real-world execution of Π^{SH}. We have:

$$\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j)\}_{j\in\overline{T}} \quad , \quad (\cdot, \mathsf{ek}) := \mathsf{CombineKeys}\left(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}\right)$$

$$\{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{ReRand}(c) \quad , \quad \{\mu_j \leftarrow \mathsf{ShareDec}_{\mathsf{sk}_j}(\hat{c})\}_{j\in\overline{T}}$$

Hybrid 1: We change how $\hat{c}$ is created. Instead of using the ReRand algorithm, we simply sample a fresh encryption of $y = C(x_1, \ldots, x_N)$ with mode $= 1$:

$$\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j)\}_{j\in\overline{T}} \quad , \quad (\cdot, \mathsf{ek}) := \mathsf{CombineKeys}\left(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}\right)$$

$$\{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(y, 1) \quad , \quad \{\mu_j \leftarrow \mathsf{ShareDec}_{\mathsf{sk}_j}(\hat{c})\}_{j\in\overline{T}}$$

We claim that $\mathsf{View}_0 \overset{s}{\approx} \mathsf{View}_1$ by *ciphertext re-randomization* of $\mathcal{E}$. The reduction gets $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in T}$ from $\mathcal{A}$ and computes $\left\{(\mathsf{pk}_j, \mathsf{sk}_j, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j)\right\}_{j\in\overline{T}}$ honestly. It computes $(\cdot, \mathsf{ek})$ and pk using $\mathsf{CombineKeys}$ and $\mathsf{CombinePK}$, respectively, and also computes $\{c_j \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}}$ and $c := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_N)$ if the server is honest (otherwise, it receives c from $\mathcal{A}$). Given $\hat{c}$ which is either $\hat{c} = \mathsf{ReRand}(c)$ or $\hat{c} = \mathsf{Enc}_{\mathsf{pk}}(y, 1)$, it computes $\left\{\mu_j \leftarrow \mathsf{ShareDec}_{\mathsf{sk}_j}(\hat{c})\right\}_{j\in\overline{T}}$. Since $\mathcal{A}$ is a semi-honest adversary, we know that $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ is a valid key tuple for all $i \in T$. Likewise, we know that for all $i \in T$, there exist $r_i^{(x)}$ such that $c_i = \mathsf{Enc}_{\mathsf{pk}}(x_i, 0\,; r_i^{(x)})$. Therefore, $\mathsf{View}_0 \overset{s}{\approx} \mathsf{View}_1$ by *ciphertext re-randomization* of $\mathcal{E}$.

Hybrid 2: We change the way the decryption shares $\{\mu_j\}_{j\in\overline{T}}$ are generated. Instead of using the $\mathsf{ShareDec}$ algorithm with secret key sk_j, we now use the $\mathsf{SimShareDec}$ algorithm with $y = C(x_1, \ldots, x_N)$ and $\{\mu_i\}_{i\in T}$:

$$\left\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j)\right\}_{j\in\overline{T}} \quad , \quad (\cdot, \mathsf{ek}) := \mathsf{CombineKeys}\left(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}\right)$$

$$\{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(y, 1) \quad , \quad \{\mu_j\}_{j\in\overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, y, \{\mu_i\}_{i\in T})$$

We claim that $\mathsf{View}_1 \overset{c}{\approx} \mathsf{View}_2$ by *share-simulation indistinguishability* of $\mathcal{E}$. The reduction takes $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in T}$ from $\mathcal{A}$ and computes $\left\{(\mathsf{pk}_j, \mathsf{sk}_j, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j)\right\}_{j\in\overline{T}}$ honestly. It computes pk and ek using $\mathsf{CombinePK}$ and $\mathsf{CombineKeys}$, respectively, and also computes $\{c_j \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}}$ and $\hat{c} = \mathsf{Enc}_{\mathsf{pk}}(y, 1)$. With $\{c_i\}_{i\in T}$ that $\mathcal{A}$ sends, it computes $c := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_N)$. Since $\mathcal{A}$ is a semi-honest adversary, we know that $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ is a valid key tuple for all $i \in T$. Likewise, we know that for all $i \in T$, there exists $r_i^{(dec)}$ such that $\mu_i := \mathsf{ShareDec}(\hat{c}\,; r_i^{(dec)})$. Thus, $\mathsf{View}_1 \overset{c}{\approx} \mathsf{View}_2$ by *share-simulation indistinguishability* of $\mathcal{E}$.

Hybrid 3: This is actually a series of $N - t$ hybrids. We change how the ciphertexts $\{c_j\}_{j\in\overline{T}}$ are created, one at a time. Instead of having c_{j^*} be an encryption of input x_{j^*}, we have it be an encryption of 0.

$$\left\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j)\right\}_{j\in\overline{T}} \quad , \quad (\cdot, \mathsf{ek}) := \mathsf{CombineKeys}\left(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}\right)$$

$$\{c_j = \mathsf{Enc}_{\mathsf{pk}}(0)\}_{j\in\overline{T}, j\le j^*} \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}, j> j^*} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(y, 1)$$
$$\{\mu_j\}_{j\in\overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, y, \{\mu_i\}_{i\in T})$$

We claim two consecutive hybrids are computationally indistinguishable by *semantic security* of $\mathcal{E}$. The reduction forwards $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in T}$ from $\mathcal{A}$ to the challenger and receives $\mathsf{ek}, \left\{(\mathsf{pk}_j, \mathsf{ek}_j)\right\}_{j\in\overline{T}}$ from the challenger, which it forwards to $\mathcal{A}$. It uses the challenge ciphertext as c_{j^*} and computes the other ciphertexts c_j accordingly. With $\{c_i\}_{i\in T}$ that $\mathcal{A}$ sends, it computes $c := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_N)$ (or receives it from $\mathcal{A}$ is the server is corrupted). It also computes $\hat{c} = \mathsf{Enc}_{\mathsf{pk}}(y, 1)$ and $\left\{\mu_j \leftarrow \mathsf{SimShareDec}(\hat{c}, y, \{\mu_i\}_{i\in T})\right\}_{j\in\overline{T}}$ from the values $\{\mu_i\}_{i\in T}$ sent by $\mathcal{A}$. We define View_3 to be the view having all c_j encrypting 0. Putting these $N - t$ hybrids together, we conclude that $\mathsf{View}_2 \overset{c}{\approx} \mathsf{View}_3$.

Hybrid 4: We change how the public/evaluation key pairs $\left\{(\mathsf{pk}_j, \mathsf{ek}_j)\right\}_{j\in\overline{T}}$ and the combined evaluation key ek is generated. Instead of creating them using Keygen and $\mathsf{CombineKeys}$, we use the $\mathsf{SimCombKeys}$ algorithm with $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in T}$.

$$\left(\mathsf{ek}, \left\{(\mathsf{pk}_j, \mathsf{ek}_j)\right\}_{j\in\overline{T}}\right) \leftarrow \mathsf{SimCombKeys}\left(\{(\mathsf{pk}_i, \mathsf{ek}_i)\}_{i\in T}\right)$$

$$\{c_j = \mathsf{Enc}_{\mathsf{pk}}(0)\}_{j\in\overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(y,1) \quad , \quad \{\mu_j\}_{j\in\overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, y, \{\mu_i\}_{i\in T})$$

We claim that $\mathsf{View}_3 \overset{c}{\approx} \mathsf{View}_4$ by *key-simulation indistinguishability* of $\mathcal{E}$. The reduction takes $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in T}$ from $\mathcal{A}$ and computes $(\mathsf{ek}, \{(\mathsf{pk}_j, \mathsf{ek}_j)\}_{j\in\overline{T}}) \leftarrow \mathsf{SimCombKeys}(\{(\mathsf{pk}_i, \mathsf{ek}_i)\}_{i\in T})$ and $\mathsf{pk} := \mathsf{CombinePK}(\{\mathsf{pk}_i\}_{i\in[N]})$. It also computes $\{c_j \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)\}_{j\in\overline{T}}$ and $\hat{c} = \mathsf{Enc}_{\mathsf{pk}}(y,1)$. With $\{c_i\}_{i\in T}$ that $\mathcal{A}$ sends, it computes $c := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_N)$ (or receives it from $\mathcal{A}$ is the server is corrupted). With $\{\mu_i\}_{i\in T}$ that $\mathcal{A}$ sends, it computes $\{\mu_j \leftarrow \mathsf{SimShareDec}(\hat{c}, y, \{\mu_i\}_{i\in T})\}_{j\in\overline{T}}$. Therefore, $\mathsf{View}_2 \overset{c}{\approx} \mathsf{View}_3$ by *key-simulation indistinguishability* of $\mathcal{E}$.

We have proved that $\mathsf{View}_0 \overset{c}{\approx} \mathsf{View}_4$. Since View_4 is the view that $\mathcal{S}^{\text{SH}}$ produces, and View_0 is the view produced in an execution of Π^{SH}, we have that $\text{IDEAL}_{\mathcal{F},\mathcal{S}^{\text{SH}}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\Pi^{\text{SH}},\mathcal{A}}(\vec{x})$.

□

5 Tolerating Malicious Adversaries

The protocol described in Section 4 is not secure against malicious adversaries. We now describe the changes we need to make for our protocol to be able to tolerate this kind of attack. We assume a *rushing adversary*, that is, we assume that the adversary can choose his messages *adaptively*, depending on the messages from the honest players. In order to protect against malicious players, we need to guarantee that the public and evaluation keys chosen by the players are independent. To achieve this, we add a commitment key ck for a trapdoor commitment scheme (e.g. a semantically secure encryption scheme) to the crs. This guarantees that ck is generated honestly. We then have all parties commit, using ck, to the randomness $r_i^{(gen)}$ that they will use in Keygen. The same is true for decryption shares; we must guarantee that they are independent, and therefore also have the parties commit to the the randomness $r_i^{(dec)}$ that they will use in $\mathsf{ShareDec}$. This is combined with having each player provide a zero-knowledge proof along with $(\mathsf{pk}_i, \mathsf{ek}_i)$ that proves that the keys were indeed created using the randomness $r_i^{(gen)}$ he commited to previously. Similarly, we require each player to prove that he computed his decryption share μ_i with the randomness he committed to, and the secret-key that was output by Keygen on randomness $r_i^{(gen)}$. Furthermore, we need to ensure that the ciphertexts c_i are independent, and so we have all players provide a proof of plaintext knowledge (POPK) for c_i.

There is a subtely here. Because we have a rushing adversary, in the proof the simulator will have to provide simulated proofs for the honest parties and still require soundness from the proofs created by the adverary on behalf of corrupt players. We therefore need to use simulation-sound NIZKs (SS-NIZKs). For the same reason, we need simulation-extractability (SE) for the POPK of c_i: the simulator must still be able to extract from an adversarial proof, even if the adversary has seen many simulated proofs.

To protect against a malicious server, the parties must also verify the computation performed by the server. The ciphertext c is a deterministic function of the set of ciphertexts provided by the parties $\{c_i\}_{i\in N}$, so they could potentially verify the correctness of c by doing the homomorphic operations themselves. This is obviously unsatisfactory, as they would perform computation proportional to the size of the computation. This is precisely the problem we're trying to solve! Note, however, that precisely because the parties need to verify the computation, we cannot hope to have the communication complexity and computation time of the parties be independent of the complexity of f, as required by the definition of cloud-assisted MPC given in Section 3. We therefore relax the definition and require the communication complexity and computation time of parties $P_1, \ldots, P_N$ to be *polylogarithmic* in $|C|$ and the computation time of the server to be *polynomial* in $|C|$.

To prove that it did the evaluation correctly, we require the server to provide a short argument φ, which the parties can verify in little time. This can be done in one of four different ways, each with its own benefits and drawbacks. We comment that the circuit in the discussion below is the evaluation circuit, which can be viewed as the circuit resulting from replacing each gate in C by $\text{poly}(\kappa)$ many gates. Thus, both the size and depth of the evaluation circuit are (at most) a polynomial factor (in κ) larger than the size and depth of C.

1. Use the argument system of [Kil92, Kil95]. The benefit in doing this is that the communication complexity of the argument is polylogarithmic in the size of the computation, and the computation time of the verifier is linear in the input length (up to polylogarithmic factors). On the other hand, the system of [Kil92, Kil95] is an *interactive* system consisting of 3 rounds, and therefore this approach implies increasing the number of rounds in our protocol by 2 rounds.
2. Use computationally sound (CS) proofs [Mic94]. The benefit of using this approach is that the verifier runs in nearly linear time and the communication complexity is again polylogarithmic in the size of the computation. The drawback, however, is that CS proofs require the use of a random oracle, and thus, we could only hope to achieve security in the *random oracle model*.
3. Use the one-round argument system of [GKR08]. On the one hand, using this approach allows the server to provide an argument for the correct computation of c without PCPs, but on the other hand, both the communication complexity and the computation time of the verifier are polynomial in the *depth* of the circuit, thus restricting the type of functions we can compute to those in **NC**. The construction of [GKR08] also needs to assume the existance of a secure PIR scheme.
4. Use the one-round argument system of [BCCT11] or [GLR11]. With this approach, the communication complexity is again polylogarithmic in the size of the computation, the computation time of the verifier is nearly linear in the input size, and security holds in the standard model. However, the use of either of these argument systems requires us to make a non-falsifiable assumption [Nao03].

We give a full description of the modified protocol that is secure against malicious adversaries in Figure 10. We require the same properties from $\mathcal{E}$, namely, we assume $\mathcal{E}$ is a key-homomorphic threshold FHE. We also need an argument system $\Phi = (\mathsf{Setup}^{\Phi}, \mathsf{Prove}, \mathsf{Verify})$ for verifiable computation, and a trapdoor commitment scheme $(\mathsf{Keygen}^{\mathsf{Com}}, \mathsf{Com})$. We describe the protocol in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{CRS}}^{\mathcal{D}}, \mathcal{F}_{\text{KEYS}}^{\mathcal{E}}, \mathcal{F}_{\text{RAND.BC}}^{\mathcal{E}}, \mathcal{F}_{\text{ZK.BC}}^{R^{(gen)}}, \mathcal{F}_{\text{ZK.BC}}^{R^{(x)}}, \mathcal{F}_{\text{ZK.BC}}^{R^{(dec)}})$-hybrid model, where $R^{(gen)}, R^{(x)}$ and $R^{(dec)}$ are NP relations:

$$R^{(gen)} = \Big\{ (\mathsf{params}, \mathsf{pk}_i, \mathsf{ek}_i, \mathsf{ck}, \rho_i{}^{(gen)}, i) \ , \ (r_i{}^{(gen)}, s_i{}^{(gen)}) \ \mid \ (\mathsf{pk}_i, \cdot, \mathsf{ek}_i) := \mathsf{Keygen}(\mathsf{params}, i \ ; r_i{}^{(gen)}) \wedge \rho_i{}^{(gen)} := \mathsf{Com}_{\mathsf{ck}}(r_i{}^{(gen)} \ ; s_i{}^{(gen)}) \Big\}$$

$$R^{(x)} = \{ (c, \mathsf{pk}) \ , \ (x, r) \ \mid \ c := \mathsf{Enc}_{\mathsf{pk}}(x, 0 \ ; r) \}$$

$$R^{(dec)} = \Big\{ (\mathsf{params}, c, \mu_i, \mathsf{pk}_i, \rho_i{}^{(dec)}, \rho_i{}^{(gen)}, \mathsf{ck}, i) \ , \ (r_i{}^{(dec)}, s_i{}^{(dec)}, r_i{}^{(gen)}, s_i{}^{(gen)}) \ \mid \ (\mathsf{pk}_i, \mathsf{sk}_i, \cdot) \leftarrow \mathsf{Keygen}(\mathsf{params}, i \ ; r_i{}^{(gen)}) \ \wedge \rho_i{}^{(gen)} := \mathsf{Com}_{\mathsf{ck}}(r_i{}^{(gen)} \ ; s_i{}^{(gen)}) \wedge \mu_i := \mathsf{ShareDec}_{\mathsf{sk}_i}(c \ ; r_i{}^{(dec)}) \ \wedge \ \rho_i{}^{(dec)} := \mathsf{Com}_{\mathsf{ck}}(r_i{}^{(dec)} \ ; s_i{}^{(dec)}) \Big\},$$

Lemma 5.1. *Let f be an N-input function, and let $\mathcal{F}$ be the ideal functionality computing f. Let $\mathcal{E}$ be a key-homomorphic threshold FHE scheme. Let Φ be an argument system for verifiable computation, and*

Protocol Π^{MAL}

Let C be a circuit computing f. Let $\mathcal{E} = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{CombinePK}, \mathsf{CombineKeys}, \mathsf{ShareDec}, \mathsf{ShareCombine}, \mathsf{ReRand})$ be a key-homomorphic threshold FHE scheme. Let $\Phi = (\mathsf{Setup}^{\Phi}, \mathsf{Prove}, \mathsf{Verify})$ be an argument system for verifiable computation, and let $(\mathsf{Keygen}^{\mathsf{Com}}, \mathsf{Com})$ be a trapdoor commitment scheme.

For session ID sid:

Phase 1: Each P_i calls $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ with (sid, i) and receives $(\mathsf{params}, \mathsf{ck})$. Samples randomness ${r_i}^{(gen)}$ and ${r_i}^{(dec)}$ and computes ${\rho_i}^{(gen)} := \mathsf{Com}_{\mathsf{ck}}(r_i^{(gen)}; s_i^{(gen)})$ and ${\rho_i}^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_i^{(dec)}; s_i^{(dec)})$. Sends $(\mathsf{sid}, i, ({\rho_i}^{(gen)}, {\rho_i}^{(dec)}))$ to $\mathcal{F}_{\text{BC}}$.

Phase 2: Each P_i chooses an encryption key tuple: $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i) := \mathsf{Keygen}(1^\kappa, i; {r_i}^{(gen)})$, and sends $(\mathsf{sid}, i, (\mathsf{sk}_i, \mathsf{ek}_i))$ to $\mathcal{F}_{\text{KEYS}}^{\mathcal{E}}$, $(\mathsf{sid}, i, \mathsf{pk}_i)$ to $\mathcal{F}_{\text{BC}}$, and $(\mathsf{sid}, i, (\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{ck}, {\rho_i}^{(gen)}, i), ({r_i}^{(gen)}, {s_i}^{(gen)}))$ to $\mathcal{F}_{\text{ZK.BC}}^{R^{(gen)}}$.

Phase 3: Upon receiving $(\mathsf{sid}, \mathsf{ek}, \{(\mathsf{ek}_i)\}_{i\in[N]})$ from $\mathcal{F}_{\text{KEYS}}^{\mathcal{E}}$, $(\mathsf{sid}, i, \mathsf{pk}_i)$ from $\mathcal{F}_{\text{BC}}$ and $(\mathsf{sid}, i, (\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{ck}, {\rho_i}^{(gen)}, i)$ from $\mathcal{F}_{\text{ZK.BC}}^{R^{(gen)}}$ for all $i \in [N]$, each P_i computes $\mathsf{pk} := \mathsf{CombinePK}(\{\mathsf{pk}_i\}_{i\in[N]}$ and $c_i := \mathsf{Enc}_{\mathsf{pk}}(x_i; {r_i}^{(x)})$ and sends $(\mathsf{sid}, i, (c_i, \mathsf{pk}), (x_i, {r_i}^{(x)}))$ to $\mathcal{F}_{\text{ZK.BC}}^{R^{(x)}}$.

Phase 4: Upon receiving $(\mathsf{sid}, i, (c_i, \mathsf{pk}))$ from $\mathcal{F}_{\text{ZK.BC}}^{R^{(x)}}$ for all $i \in [N]$ and $(\mathsf{sid}, \mathsf{ek}, \{\mathsf{ek}_i\}_{i\in[N]})$ from $\mathcal{F}_{\text{KEYS}}^{\mathcal{E}}$, server S computes $c := \mathsf{Eval}(c_1, \ldots, c_N)$, and sends $(\mathsf{sid}, 0, c)$ to $\mathcal{F}_{\text{RAND.BC}}^{\mathcal{E}}$. Also computes $\varphi \leftarrow \mathsf{Prove}(c, c_1, \ldots, c_N)$ and sends $(\mathsf{sid}, 0, \varphi)$ to $\mathcal{F}_{\text{BC}}$.

Phase 5: Upon receiving $(\mathsf{sid}, c, \hat{c})$ from $\mathcal{F}_{\text{RAND.BC}}^{\mathcal{E}}$, and φ from $\mathcal{F}_{\text{BC}}$, each client P_i verifies φ, computes $\mu_i := \mathsf{ShareDec}_{\mathsf{sk}_i}(\hat{c}; {r_i}^{(dec)})$ and sends $(\mathsf{sid}, i, (\hat{c}, \mu_i, \mathsf{pk}_i, {\rho_i}^{(dec)}, \mathsf{ck}, i), (\mathsf{sk}_i, {r_i}^{(dec)}, {s_i}^{(dec)}, {r_i}^{(gen)}))$ to $\mathcal{F}_{\text{ZK.BC}}^{R^{(dec)}}$.

Local Computation: Upon receiving $(\mathsf{sid}, i, (\hat{c}, \mu_i, \mathsf{pk}_i, {\rho_i}^{(dec)}, \mathsf{ck}, i))$ from $\mathcal{F}_{\text{ZK.BC}}^{R^{(dec)}}$ for all $i \in [N]$, each client P_i computes and outputs $y := \mathsf{ShareCombine}(c, \mu_1, \ldots, \mu_N)$.

Figure 10: Protocol Π^{MAL}. Implements $\mathcal{F}$ in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{CRS}}^{\mathcal{D}}, \mathcal{F}_{\text{KEYS}}^{\mathcal{E}}, \mathcal{F}_{\text{RAND.BC}}^{\mathcal{E}}, \mathcal{F}_{\text{ZK.BC}}^{R^{(gen)}}, \mathcal{F}_{\text{ZK.BC}}^{R^{(x)}}, \mathcal{F}_{\text{ZK.BC}}^{R^{(dec)}})$-hybrid model, where $\mathcal{D}$ is the distribution of $(\mathsf{params}, \mathsf{ck})$, where $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa, N)$ and $(\mathsf{ck}, \cdot) \leftarrow \mathsf{Keygen}^{\mathsf{Com}}(1^\kappa)$. Secure against malicious adversaries.

let $R^{(x)}, R^{(dec)}$ be NP-relations as described above. Then the protocol Π^{MAL} described in Figure 10 securely implements $\mathcal{F}$ against (static) malicious adversaries, in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{KEYS}}^{\mathcal{E}}, \mathcal{F}_{\text{RAND.BC}}^{\mathcal{E}}, \mathcal{F}_{\text{ZK.BC}}^{R^{(x)}}, \mathcal{F}_{\text{ZK.BC}}^{R^{(dec)}})$-hybrid model.

The proof of Lemma 5.1 is given in Appendix A.1. Instantiating key-homomorphic threshold FHE based on the PLWE assumption, a special case of the "Ring LWE" assumption of [LPR10], as in Section 6, and combining Lemma 5.1 with protocols implementing $\mathcal{F}_{\text{KEYS}}^{\mathcal{E}}$ and $\mathcal{F}_{\text{RAND.BC}}^{\mathcal{E}}$ for this instantiation (shown in Section 7), SS-NIZKs for $\mathcal{F}_{\text{ZK.BC}}^{R^{(gen)}}, \mathcal{F}_{\text{ZK.BC}}^{R^{(dec)}}$, and SE-NIZKs for $\mathcal{F}_{\text{ZK.BC}}^{R^{(x)}}$, we arrive at the following theorem:

Theorem 5.2. *Assuming the hardness of $\mathsf{PLWE}_{f,q,\chi}$ for parameters f, q, χ as described in Section 6.1, there exists a protocol that securely implements $\mathcal{F}$ against (static) malicious adversaries, in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{CRS}}^{\mathcal{D}})$-hybrid model.*

6 Instantiating Key-Homomorphic Threshold FHE

We now show how to instantiate public-key key-homomorphic threshold FHE (KHT-FHE, Definition 4.1). Our construction will be based on PLWE assumption [BV11b], which is a special case of the Ring LWE assumption of [LPR10]. We therefore first review the PLWE assumption (Section 6.1) and then present the scheme. To do the latter, we first review the somewhat homomorphic scheme of [BV11b] (Section 6.2) and show how to use the re-linearization and modulus reduction techniques of [BV11a, BGV11] to convert it into a (leveled) fully homomorphic scheme (Section 6.3). We then show that the scheme satisfies key homomorphism and also show how to instantiate the algorithms CombinePK, CombineKeys, ShareDec, ShareCombine, ReRand and simulators SimCombKeys, SimShareDec (Section 6.4).

6.1 The Polynomial-LWE Assumption

In this section, we describe the *polynomial LWE* (or, PLWE) assumption from [BV11b], which is a special case of the "ring learning with errors" (RLWE) assumption of Lyubaskevsky, Peikert and Regev [LPR10]. The PLWE assumption is analogous to the by now standard "learning with errors" (LWE) assumption, first defined by Regev [Reg05, Reg09] (generalizing the learning parity with noise assumption of Blum et al. [BFKL93]). In the PLWE assumption, we consider rings $R \doteq \mathbb{Z}[x]/\langle f(x)\rangle$ and $R_q \doteq R/qR$ for some degree n integer polynomial $f(x) \in \mathbb{Z}[x]$ and a prime integer $q \in \mathbb{Z}$. Note that $R_q \equiv \mathbb{Z}_q[x]/\langle f(x)\rangle$, i.e. the ring of degree n polynomials modulo $f(x)$ with coefficients in $\mathbb{Z}_q$. Addition in these rings is done component-wise in their coefficients (thus, their additive group is isomorphic to $\mathbb{Z}^n$ and $\mathbb{Z}_q^n$ respectively). Multiplication is simply polynomial multiplication modulo $f(x)$ (and also q, in the case of the ring R_q). Thus an element in R (or R_q) can be viewed as a degree n polynomial over $\mathbb{Z}$ (or $\mathbb{Z}_q$). We represent such an element using the vector of its coefficients. For an element $a(x) = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1} \in R$, we let $\|a\| = \max |a_i|$ denote its ℓ_∞ norm.

The $\mathsf{PLWE}_{f,q,\chi}$ assumption is that for a random ring element $s \leftarrow R_q$, given any polynomial number of samples of the form $(a_i, b_i = a_i \cdot s + e_i) \in (R_q)^2$, where a_i is uniformly random in R_q and e_i is drawn from the error distribution χ, the b_i's are computationally indistinguishable from uniform in R_q. We use the *hermite normal form* of the assumption, as in [BV11b], where the secret s is sampled from the noise distribution χ rather than being uniform in R_q. This presentation is more useful for the purposes of this paper and it turns out that to be equivalent to the original one up to obtaining one additional sample [AiCPS09, LPR10].

Definition 6.1 (The PLWE Assumption - Hermite Normal Form [BV11b, LPR10])**.** *For all $\kappa \in \mathbb{N}$, let $f(x) = f_\kappa(x) \in \mathbb{Z}[x]$ be a polynomial of degree $n = n(\kappa)$, let $q = q(\kappa) \in \mathbb{Z}$ be a prime integer, let the ring $R \doteq \mathbb{Z}[x]/\langle f(x)\rangle$ and $R_q \doteq R/qR$, and let χ denote a distribution over the ring R.*

The polynomial LWE assumption $\mathsf{PLWE}_{f,q,\chi}$ *states that for any $\ell = \mathrm{poly}(\kappa)$ it holds that*

$$\{(a_i, a_i \cdot s + e_i)\}_{i\in[\ell]} \overset{c}{\approx} \{(a_i, u_i)\}_{i\in[\ell]} \ ,$$

where s is sampled from the noise distribution χ, a_i are uniform in R_q, the "error polynomials" e_i are sampled from the error distribution χ, and finally, the ring elements u_i are uniformly random over R_q.

Fact 6.2 ([BV11b])**.** *The $\mathsf{PLWE}^\ell_{f,q,\chi}$ assumption implies that,*

$$\{(a_i, a_i \cdot s + 2 \cdot e_i)\}_{i\in[\ell]} \overset{c}{\approx} \{(a_i, u_i)\}_{i\in[\ell]} \ .$$

where a_i, s, e_i and u_i are as in Definition 6.1.

Choice of Parameters The correctness, security, and homomorphic properties of the scheme of [BV11b] rely on specific choices of the polynomial $f(x)$, the modulus q, and the error distribution χ. For security parameter κ:

- We set $f(x)$ to be the $2^{\lfloor \log k \rfloor \text{th}}$ cyclotomic polynomial. This implies that $f(x) = \Phi_{2^{\lfloor \log \kappa \rfloor}}(x) = x^n + 1$ for $n = 2^{\lfloor \log \kappa \rfloor - 1}$. Since $n \in (\kappa/4, \kappa]$, all asymptotics can be stated in terms of n.
- The error distribution χ is the (truncated) discrete Gaussian distribution $D_{\mathbb{Z}^n, r, B(r)}$ for standard deviation $r > 0$ and truncation bound $B(r)$. A sample from this distribution defines a polynomial $e(x) \in R$ with each coefficient bounded by $B(r)$. If $B(r) = \omega(\log \kappa \cdot r$, then the truncated distribution $D_{\mathbb{Z}^n, r, B(r)}$ is statistically close from the (non-truncated) discrete Gaussian $D_{\mathbb{Z}^n, r}$

The Worst-case to Average-case Connection. We state a worst-case to average-case reduction from the shortest vector problem on ideal lattices to the PLWE problem for our setting of parameters. The reduction stated below is a special case the results of [LPR10].

Theorem 6.1 (A special case of [LPR10])**.** *Let κ be the security parameter. Let $k \in \mathbb{N}$ and let $m = 2^{\lfloor \log \kappa \rfloor}$ be a power of two. Let $\Phi_m(x) = x^n + 1$ be the m^{th} cyclotomic polynomial of degree $n = \varphi(m) = m/2$. Let $r \geq \omega(\sqrt{\log n})$ be a real number, and let $q \equiv 1 \pmod{m}$ be a prime integer. Let $R = \mathbb{Z}[x] / \langle \Phi_{2^{\lfloor \log \kappa \rfloor}}(x) \rangle$. Then:*

- *There is a randomized reduction from $2^{\omega(\log n)} \cdot (q/r)$-approximate R-SVP to $\mathsf{PLWE}_{\Phi_{2^{\lfloor \log \kappa \rfloor}}, q, \chi}$ where $\chi = D_{\mathbb{Z}^n, r}$ is the discrete Gaussian distribution. The reduction runs in time $\mathrm{poly}(n, q)$.*
- *There is a randomized reduction from $(n^2 q/r) \cdot (n(\ell+1)/\log(n(\ell+1)))^{1/4}$-approximate R-SVP to $\mathsf{PLWE}^{(\ell)}_{\Phi_{2^{\lfloor \log \kappa \rfloor}}, q, \chi}$ where $\chi = D_{\mathbb{Z}^n, r}$ is the discrete Gaussian distribution. The reduction runs in time $\mathrm{poly}(n, q, \ell)$.*

6.2 A Somewhat-Homomorphic Scheme

We describe the scheme of Brakerski and Vaikuntanathan [BV11b] based on PLWE.[3] For security parameter κ, the scheme is parameterized by a prime number q, a degree n polynomial $f(x) \in \mathbb{Z}[x]$, and an error distribution χ over the ring $R_q \doteq \mathbb{Z}_q[x] / \langle f(x) \rangle$. The parameters n, f, q and χ are public and we assume that given κ, there are polynomial-time algorithms that output f and q, and sample from the error distribution χ.

- $\mathsf{Setup}(1^\kappa)$: Sample a ring element vector $\mathbf{a} \leftarrow R_q^n$, and set $\mathsf{params} := \mathbf{a}$.
- $\mathsf{Keygen}(\mathsf{params})$: Sample a ring element $s \leftarrow \chi$ and a ring element vector $\mathbf{x} \leftarrow \chi^n$, and set

$$\mathsf{sk} := s \quad \text{and} \quad \mathsf{pk} := \mathbf{p} = \mathbf{a}s + 2\mathbf{x} \in R_q^n$$

- $\mathsf{Enc}(\mathsf{sk}, m)$: To encrypt a bit $m \in \{0, 1\}$, sample $\mathbf{e} \leftarrow \chi^n$, and compute

$$c_0 := \langle \mathbf{p}, \mathbf{e} \rangle + m \in R_q \qquad \text{and} \qquad c_1 := \langle \mathbf{a}, \mathbf{e} \rangle \in R_q$$

and output the ciphertext $\mathbf{c} := (c_0, c_1) \in R_q^2$.

[3] The scheme presented in [BV11b] is a private-key FHE. We make the natural modifications and present it as a public-key scheme.

- $\mathsf{Dec}(\mathsf{sk}, \mathbf{c})$: To decrypt, we first compute $c_0 - c_1 s \in R_q$, and then output $m = c_0 - c_1 s \pmod 2$ as the message. Note that the condition for correct decryption is that the ℓ_∞ norm of the polynomial $c_0 - c_1 s$ is smaller than $q/2$.

Theorem 6.2 ([BV11b]). *Let $n, q, f(x)$ be as in the scheme, let $r = \mathrm{poly}(n)$ and $q = 2^{n^\epsilon}$ for some $0 < \epsilon < 1$. Then, the scheme allows evaluation of degree-$O(n^\epsilon / \log n)$ polynomials with at most $2^{O(n^\epsilon/\log n)}$ terms, and is secure under the worst-case hardness of approximating shortest vectors on ideal lattices to within a factor of $O(2^{n^\epsilon})$.*

We will use the fact that ciphertexts of this scheme are *pseudorandom*. This follows from the fact that $\mathbf{p}$ is a PLWE sample and therefore pseudorandom, and the fact that for $\mathbf{e}$ chosen according to the discrete gaussian, $\langle \mathbf{a}, \mathbf{e} \rangle$ is statistically close to uniform.

6.3 Getting a (Leveled) Fully-Homomorphic Scheme via Re-linearization and Modulus Reduction

We now describe how to transform the somewhat homomorphic scheme from Section 6.2 into a (leveled) fully homomorphic scheme. We use the re-linearization and modulus reduction techniques of [BV11a, BGV11], as opposed to the "squashing" and "bootstrapping" blueprint of [Gen09b] used in [BV11b]. We do this in order to keep the $\mathsf{CombineKeys}$ algorithm and thus the N-party protocol implementing the "combine keys" functionality $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$, simple.

6.3.1 Re-linearization

Let us first look at the symbolic linear function $f_{\mathbf{c}}(x) : R_q \to \{0,1\}$ defined as: $f_{\mathbf{c}}(x) = c_1 - c_0 x \mod 2$. We can describe homomorphic addition and multiplication in terms of f. First note that decrypting a ciphertext $\mathbf{c}$ corresponds to simply evaluating $f_{\mathbf{c}}(s)$. For ciphertexts $\mathbf{c}$ and $\mathbf{c}'$ encrypting m and m' respectively, we have:

$$m + m' = f_{\mathbf{c}}(s) + f_{\mathbf{c}'}(s) = (c_0 + c'_0) - (c_1 + c'_1)s$$

Therefore, it is natural to define $\mathbf{c}_{\text{add}} = \mathbf{c} + \mathbf{c}' = (c_0 + c_0, c_1 + c'_1)$. Multiplication is a bit more tricky. We have:

$$m \cdot m' = f_{\mathbf{c}}(s) \cdot f_{\mathbf{c}'}(s) = (c_0 - c_1 s)(c'_0 - c'_1 s) = c_0 c'_0 - (c_0 c'_1 + c_1 c'_0)s + c_1 c'_1 s^2 = h_0 + h_1 s + h_2 s^2$$

In [BV11b], the authors let $\mathbf{c}_{\text{mult}} = (h_0, h_1, h_2)$, so that the size of the ciphertext grows linearly with the number of multiplications performed. This requires the secret key to be $\mathbf{s} = (1, s, s^2, \ldots, s^D)$, where D is the maximal degree of homomorphism tolerated by the scheme. In our setting of multiparty computation, this would require a very complex protocol Π_{KEYS} to compute $\mathsf{CombineKeys}$, and therefore we opt for another strategy. Instead of letting the ciphertext grow with each multiplication, we use the re-linearization technique of Brakerski and Vaikuntanathan [BV11a] to reduce the size of the resulting ciphertext c_{mult} *after each multiplication*.

To be able to homomorphically evaluate a polynomial of degree D while keeping the ciphertext size constant, we have Keygen sample D different polynomials $s_0, \ldots, s_D \leftarrow \chi$, one for each level, and set $\mathsf{sk} = (s_0, \ldots, s_D)$, where s_0 is used to create the public key $\mathsf{pk} := \mathbf{p} = \mathbf{a}s_0 + \mathbf{x}$ and s_D is used for decryption.[4] The evaluation key is computed as follows. For all $d \in [D], \tau \in \{0, \ldots, \lfloor \log q \rfloor\}$, sample

[4] The secret key sk could potentially be composed of only s_D. However, we let it contain all s_i's because we will need them when we implement the $\mathsf{CombineKeys}$ algorithm.

$(a_{d,\tau}, b_{d,\tau} = a_{d,\tau}s_d + 2e_{d,\tau}) \in R_q^2$ and $(a'_{d,\tau}, b'_{d,\tau} = a'_{d,\tau}s_d + 2e'_{d,\tau}) \in R_q^2$, where $a_{d,\tau}, a'_{d,\tau} \leftarrow R_q$ and $e_{d,\tau}, e'_{d,\tau} \leftarrow \chi$. Compute

$$\widetilde{c}_{0,d,\tau} := b_{d,\tau} + 2^\tau s_{d-1} \in R_q \qquad \text{and} \qquad \widetilde{c}_{1,d,\tau} := a_{d,\tau}$$
$$\widetilde{z}_{0,d,\tau} := b'_{d,\tau} + 2^\tau s_{d-1}^2 \in R_q \qquad \text{and} \qquad \widetilde{z}_{1,d,\tau} := a'_{d,\tau}$$ [5]

We let

$$\widetilde{\mathbf{c}}_{0,d} = (\widetilde{c}_{0,d,0}, \ldots, \widetilde{c}_{0,d,\lfloor \log q \rfloor}) \quad \text{and} \quad \widetilde{\mathbf{c}}_1 = (\widetilde{c}_{1,d,0}, \ldots, \widetilde{c}_{1,d,\lfloor \log q \rfloor})$$
$$\widetilde{\mathbf{z}}_{0,d} = (\widetilde{z}_{0,d,0}, \ldots, \widetilde{z}_{0,d,\lfloor \log q \rfloor}) \quad \text{and} \quad \widetilde{\mathbf{z}}_1 = (\widetilde{z}_{1,d,0}, \ldots, \widetilde{z}_{1,d,\lfloor \log q \rfloor})$$

and set the evaluation key

$$\mathsf{ek} := \{\widetilde{\mathbf{c}}_{0,d}, \widetilde{\mathbf{c}}_{1,d}, \widetilde{\mathbf{z}}_{0,d}, \widetilde{\mathbf{z}}_{1,d}\}_{d \in [D]}$$

$\mathsf{SH.Eval}(p, (\mathbf{c}_1, \ldots, \mathbf{c}_N))$: We show how to evaluate an N-variate polynomial $p : \{0,1\}^N \to \{0,1\}$ of degree D. To this end, we show how to homomorphically add and multiply two elements in $\{0,1\}$.

- Given two ciphertexts $\mathbf{c} = (c_0, c_1)$ and $\mathbf{c}' = (c'_0, c'_1)$ *under the same secret key* s_{d-1}, output the ciphertext $\mathbf{c}_{\text{add}} = \mathbf{c} + \mathbf{c}' = (c_0 + c'_0, c_1 + c'_1) \in R_q^2$, as an encryption of the *sum* of the underlying messages.
- Given two ciphertexts $\mathbf{c} = (c_0, c_1)$ and $\mathbf{c}' = (c'_0, c'_1)$ *under the same secret key* s_{d-1}, an encryption of their *product* is computed as follows.
  Define $h_0, h_1, h_2 \in R_q$ as before: $h_0 + h_1 x + h_2 x^2 = (c_0 - c_1 x)(c'_0 - c'_1 x)$. Then in particular, $h_0 + h_1 s_{d-1} + h_2 s_{d-1}^2 \pmod 2 = m \cdot m'$. Let $\mathbf{u} = (u_0, \ldots, u_{\lfloor \log q \rfloor}) \in R_2^{\lceil \log q \rceil}$ for $u_i \in R_2$ be the binary representation of h_1, and let $\mathbf{v} = (v_0, \ldots, v_{\lfloor \log q \rfloor}) \in R_2^{\lceil \log q \rceil}$ for $v_i \in R_2$ be the binary representation of h_2, so that:

  $$h_1 = \sum_{\tau=0}^{\lfloor \log q \rfloor} 2^\tau u_\tau \quad \text{and} \quad h_2 = \sum_{\tau=0}^{\lfloor \log q \rfloor} 2^\tau v_\tau$$

  Output

  $$\mathbf{c}_{\text{mult}} = (h_0 + \langle \mathbf{u}, \widetilde{\mathbf{c}}_{0,d} \rangle + \langle \mathbf{v}, \widetilde{\mathbf{z}}_{0,d} \rangle, \langle \mathbf{u}, \widetilde{\mathbf{c}}_{1,d} \rangle + \langle \mathbf{v}, \widetilde{\mathbf{z}}_{1,d} \rangle),$$

  which is a valid encryption of $m \cdot m'$ under secret key s_d. To see why this is the case, simply notice that:

  $$\langle \mathbf{u}, \widetilde{\mathbf{c}}_{0,d} \rangle = \sum_{\tau=0}^{\lfloor \log q \rfloor} u_\tau c_{0,d,\tau} = \sum_{\tau=0}^{\lfloor \log q \rfloor} u_\tau a_{d,\tau} s_d + \sum_{\tau=0}^{\lfloor \log q \rfloor} u_\tau 2^\tau s_{d-1} + 2e = \langle \mathbf{u}, \widetilde{\mathbf{c}}_{1,d} \rangle s_d + h_1 + s_{d-1} + 2e$$

  and

  $$\langle \mathbf{v}, \widetilde{\mathbf{z}}_{0,d} \rangle = \sum_{\tau=0}^{\lfloor \log q \rfloor} v_\tau z_{0,d,\tau} = \sum_{\tau=0}^{\lfloor \log q \rfloor} v_\tau a'_{d,\tau} s_d + \sum_{\tau=0}^{\lfloor \log q \rfloor} v_\tau 2^\tau s_{d-1}^2 + 2e' = \langle \mathbf{v}, \widetilde{\mathbf{z}}_{1,d} \rangle s_d + h_2 + s_{d-1}^2 + 2e',$$

  where e and e' are "small" error terms. Thus,

  $$h_0 + \langle \mathbf{u}, \widetilde{\mathbf{c}}_{0,d} \rangle + \langle \mathbf{v}, \widetilde{\mathbf{z}}_{0,d} \rangle - (\langle \mathbf{u}, \widetilde{\mathbf{c}}_{1,d} \rangle + \langle \mathbf{v}, \widetilde{\mathbf{z}}_{1,d} \rangle) s_d \pmod 2 = h_0 + h_1 s_{d-1} + h_2 s_{d-1}^2 \pmod 2 = m \cdot m'$$

[5] The reader may notice that $(\widetilde{c}_{0,d,\tau}, \widetilde{c}_{1,d,\tau})$ and $(\widetilde{z}_{0,d,\tau}, \widetilde{z}_{1,d,\tau})$ are private-key *pseudo-encryptions* of $2^\tau s_{d-1}$ and $2^\tau s_{d-1}$,respectively, under secret key s_d. We say they are pseudo-encryptions because even though they were created as normal ciphertexts, they cannot be decrypted in the usual way since $2^\tau s_{d-1}$ and $2^\tau s_{d-1}^2$ are not in the message space $\{0,1\}$.

6.3.2 Modulus Reduction

We have shown how to keep the ciphertext size constant when performing evaluation. However, the magnitude of the noise in the ciphertext still grows with each operation. In order to handle this, we use the modulus reduction technique of [BV11a, BGV11] as a noise management technique. We use the notation of [BGV11]. For a secret key s, we let $\mathbf{s} = (1, s)$ and re-write the decryption function as: $[\langle \mathbf{c}, \mathbf{s} \rangle]_q \pmod 2$, where $[\ \cdot\ ]_q$ denotes reducing modulo q.

Modulus reduction allows us to transform a ciphertext $\mathbf{c} \in R_q^2$ into a different ciphertext $\mathbf{c}' \in R_p^2$ while preserving correctness:

$$[\langle \mathbf{c}', \mathbf{s} \rangle]_p = [\langle \mathbf{c}, \mathbf{s} \rangle]_q \pmod 2$$

The transformation from $\mathbf{c}$ to $\mathbf{c}'$ involves simply scaling by (p/q) and rounding appropriately. Moreover, if s is short and p is sufficiently smaller than q, the "noise" in the ciphertext actually decreases.

Lemma 6.3 ([BGV11]). *Let p and q be two odd moduli, and let $\mathbf{c}$ be an integer vector. Define $\mathbf{c}'$ to be the integer vector closest to $(p/q) \cdot \mathbf{c}$ such that $\mathbf{c}' = \mathbf{c} \pmod 2$. Then, for any $\mathbf{s}$ with $\|[\langle \mathbf{c}, \mathbf{s} \rangle]_q\| < q/2 - (q/p) \cdot \|\mathbf{s}\|_1$, we have*

$$[\langle \mathbf{c}', \mathbf{s} \rangle]_p = [\langle \mathbf{c}, \mathbf{s} \rangle]_q \pmod 2 \quad \text{and} \quad \left\|[\langle \mathbf{c}', \mathbf{s} \rangle]_p\right\| < (p/q) \cdot \|[\langle \mathbf{c}, \mathbf{s} \rangle]_q\| + \|\mathbf{s}\|_1$$

where $\|\mathbf{s}\|$ and $\|\mathbf{s}\|_1$ are the ℓ_∞ and ℓ_1 norms of $\mathbf{s}$, respectively.

To see why the lemma statement is true, consider a coordinate $(\langle \mathbf{c}, \mathbf{s} \rangle)_i$ of $\langle \mathbf{c}, \mathbf{s} \rangle$. We know that there exists $k \in \mathbb{Z}$ such that:

$$[(\langle \mathbf{c}, \mathbf{s} \rangle)_i]_q = (\langle \mathbf{c}, \mathbf{s} \rangle)_i - kq \in \left[-\frac{q}{2} + \frac{q}{p} \|\mathbf{s}\|_1 , \frac{q}{2} - \frac{q}{p} \|\mathbf{s}\|_1 \right],$$

so that

$$(\langle (p/q) \cdot \mathbf{c}, \mathbf{s} \rangle)_i - kp \in \left[-\frac{p}{2} + \|\mathbf{s}\|_1 , \frac{p}{2} - \|\mathbf{s}\|_1 \right]$$

Substituting $(p/q) \cdot \mathbf{c}$ by $\mathbf{c}'$ incurrs and error of at most $\|\mathbf{s}\|_1$. Therefore,

$$(\langle \mathbf{c}', \mathbf{s} \rangle)_i - kp \in \left[-\frac{p}{2}, \frac{p}{2} \right] \quad \text{and} \quad [(\langle \mathbf{c}', \mathbf{s} \rangle)_i]_p = (\langle \mathbf{c}', \mathbf{s} \rangle)_i - kp$$

This proves the second part of the lemma. To prove the first part, notice that since p and q are both odd, we know $kp \equiv kq \pmod 2$. Moreover, we chose $\mathbf{c}'$ such that $\mathbf{c} \equiv \mathbf{c} \pmod 2$. We thus have

$$\begin{gathered} (\langle \mathbf{c}', \mathbf{s} \rangle)_i - kp \equiv (\langle \mathbf{c}, \mathbf{s} \rangle)_i - kq \pmod 2 \\ [(\langle \mathbf{c}, \mathbf{s} \rangle)_i]_p \equiv [(\langle \mathbf{c}, \mathbf{s} \rangle)_i]_q \pmod 2 \end{gathered}$$

We now show the changes we need to make to the evaluation algorithm. Let D be the maximum level of homomorphism we wish to achieve, and let $q_0, \ldots, q_D$ be a ladder of decreasing moduli. As before, Keygen samples D different polynomials $s_0, \ldots, s_D \leftarrow \chi$, one for each level, and sets $\mathsf{sk} = (s_0, \ldots, s_D)$, where $\mathsf{pk} := \mathbf{p} = \mathbf{a}s_0 + \mathbf{x}$ and s_D is used for decryption. The evaluation key is computed as before, except that $\widetilde{c}_{0,d,\tau}, \widetilde{c}_{1,d,\tau}, \widetilde{z}_{0,d,\tau}, \widetilde{z}_{1,d,\tau}$ are reduced modulo q_{d-1}. In more detail: for all $d \in [D], \tau \in \{0, \ldots, \lfloor \log q_{d-1} \rfloor\}$, sample $(a_{d,\tau}, b_{d,\tau} = a_{d,\tau} s_d + 2e_{d,\tau}) \in R_{q_{d-1}}^2$ and $(a'_{d,\tau}, b'_{d,\tau} = a'_{d,\tau} s_d + 2e'_{d,\tau}) \in R_{q_{d-1}}^2$, where $a_{d,\tau}, a'_{d,\tau} \leftarrow R_{q_{d-1}}$ and $e_{d,\tau}, e'_{d,\tau} \leftarrow \chi$. Compute

$$\widetilde{c}_{0,d,\tau} := b_{d,\tau} + 2^\tau s_{d-1} \in R_{q_{d-1}} \qquad \text{and} \qquad \widetilde{c}_{1,d,\tau} := a_{d,\tau}$$

$$\widetilde{z}_{0,d,\tau} := b'_{d,\tau} + 2^\tau s_{d-1}^2 \in R_{q_{d-1}} \qquad \text{and} \qquad \widetilde{z}_{1,d,\tau} := a'_{d,\tau}$$

Let

$$\widetilde{\mathbf{c}}_{0,d} = (\widetilde{c}_{0,d,0}, \ldots, \widetilde{c}_{0,d,\lfloor \log q \rfloor}) \quad \text{and} \quad \widetilde{\mathbf{c}}_1 = (\widetilde{c}_{1,d,0}, \ldots, \widetilde{c}_{1,d,\lfloor \log q \rfloor})$$
$$\widetilde{\mathbf{z}}_{0,d} = (\widetilde{z}_{0,d,0}, \ldots, \widetilde{z}_{0,d,\lfloor \log q \rfloor}) \quad \text{and} \quad \widetilde{\mathbf{z}}_1 = (\widetilde{z}_{1,d,0}, \ldots, \widetilde{z}_{1,d,\lfloor \log q \rfloor})$$

and set the evaluation key

$$\mathsf{ek} := \{\widetilde{\mathbf{c}}_{0,d}, \widetilde{\mathbf{c}}_{1,d}, \widetilde{\mathbf{z}}_{0,d}, \widetilde{\mathbf{z}}_{1,d}\}_{d \in [D]}$$

$\mathsf{SH.Eval}(p, (\mathbf{c}_1, \ldots, \mathbf{c}_N))$: We show how to evaluate an N-variate polynomial $p : \{0,1\}^N \to \{0,1\}$ of degree D. To this end, we show how to homomorphically add and multiply two elements in $\{0,1\}$.

- Given two ciphertexts $\mathbf{c} = (c_0, c_1)$ and $\mathbf{c}' = (c'_0, c'_1)$ *under the same secret key* s_{d-1} *and modulus* q_{d-1}, output the ciphertext $\mathbf{c}_{\text{add}} = \mathbf{c} + \mathbf{c}' = (c_0 + c'_0, c_1 + c'_1) \in R^2_{q_{d-1}}$, as an encryption of the *sum* of the underlying messages.
- Given two ciphertexts $\mathbf{c} = (c_0, c_1)$ and $\mathbf{c}' = (c'_0, c'_1)$ *under the same secret key* s_{d-1} *and modulus* q_{d-1}, an encryption of their *product* is computed as follows.
  First, re-linearize: define $h_0, h_1, h_2 \in R_{q_{d-1}}$ as before, $h_0 + h_1 x + h_2 x^2 = (c_0 - c_1 x)(c'_0 - c'_1 x)$. Let $\mathbf{u} = (u_0, \ldots, u_{\lfloor \log q \rfloor}) \in R_2^{\lceil \log q \rceil}$ for $u_i \in R_2$ be the binary representation of h_1, and let $\mathbf{v} = (v_0, \ldots, v_{\lfloor \log q \rfloor}) \in R_2^{\lceil \log q \rceil}$ for $v_i \in R_2$ be the binary representation of h_2. Let

  $$\mathbf{c}_{\text{mult}} = (h_0 + \langle \mathbf{u}, \widetilde{\mathbf{c}}_{0,d} \rangle + \langle \mathbf{v}, \widetilde{\mathbf{z}}_{0,d} \rangle, \langle \mathbf{u}, \widetilde{\mathbf{c}}_{1,d} \rangle + \langle \mathbf{v}, \widetilde{\mathbf{z}}_{1,d} \rangle)$$

  Finally, reduce the modulus: let $\mathbf{c}'_{\text{mult}}$ be the integer vector closest to $(q_d/q_{d-1}) \cdot \mathbf{c}_{\text{mult}}$ such that $\mathbf{c}'_{\text{mult}} = \mathbf{c}_{\text{mult}} \pmod 2$. Output $\mathbf{c}'_{\text{mult}}$.

6.4 Key-Homomorphic Threshold FHE from PLWE

We describe an instantiation of key-homomorphic threshold FHE based on the FHE described in Section 6.2 and Section 6.3. Our scheme is parametrized by two noise distributions, χ_0 and χ_1, with $\chi_b = D_{\mathbb{Z}^n, r_b, B(r_b)}$, as described in Section 6.1.

Algorithms. We instantiate algorithms Dec and Eval as described in Section 6.2 and Section 6.3. Below we show how to instantiate the algorithms Keygen, Enc, CombinePK, CombineKeys, ShareDec, ShareCombine, ReRand and simulators SimCombKeys, SimShareDec. For threshold decryption we use a scheme analogous to the one of [BD10] for standard LWE. We note, however, that threshold decryption in [BD10] is only secure (ie. shares do not reveal information about the secret key) if the ciphertext that is being decrypted has a random (or statistically close to random) c_1. This is the reason why in our cloud-assisted MPC protocol, we need the parties to jointly *re-randomize* the ciphertext $c = \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_N)$ before they can threshold decrypt it.

- $\mathsf{Setup}(1^\kappa, N)$: Let $\mathbf{a} \leftarrow R_q^n$. For $i \in [N], d \in \{0, \ldots, D\}, \tau \in \{0, \ldots, \lfloor \log q_d \rfloor\}$, sample $a^{(i)}_{d,\tau}, a'^{(i)}_{d,\tau} \leftarrow R_{q_d}$. Output

  $$\mathsf{params} = \left(\mathbf{a}, \left\{a^{(i)}_{d,\tau}, a'^{(i)}_{d,\tau}\right\}_{d,\tau,i}\right)$$

- $\mathsf{Keygen}(1^\kappa, i, \mathsf{params})$: Sample D different polynomials $s^{(i)}_0, \ldots, s^{(i)}_D \leftarrow \chi_0$ and a vector $\mathbf{x} \leftarrow \chi^n$. Set $\mathsf{sk}_i := (s^{(i)}_0, \ldots, s^{(i)}_D)$ and $\mathsf{pk}_i := \mathbf{a}s_0 + \mathbf{x}$. The evaluation key is computed as before, using

the $a_{d,\tau}^{(i)}, a'^{(i)}_{d,\tau}$ in params instead of sampling them randomly from $R_{q_{d-1}}$. In more detail: for all $d \in [D], \tau \in \{0, \ldots, \lfloor \log q_{d-1} \rfloor\}$, sample $e_{d,\tau}^{(i)}, e'^{(i)}_{d,\tau} \leftarrow \chi_0$ and compute

$$\widetilde{c}_{0,d,\tau}^{(i)} := a_{d,\tau}^{(i)} + 2e_{d,\tau}^{(i)} + 2^\tau s_{d-1}^{(i)} \in R_{q_{d-1}} \qquad \text{and} \qquad \widetilde{c}_{1,d,\tau}^{(i)} := a_{d,\tau}^{(i)}$$

$$\widetilde{z}_{0,d,\tau}^{(i)} := a'^{(i)}_{d,\tau} + 2e'^{(i)}_{d,\tau} + 2^\tau (s_{d-1}^{(i)})^2 \in R_{q_{d-1}} \qquad \text{and} \qquad \widetilde{z}_{1,d,\tau}^{(i)} := a'^{(i)}_{d,\tau}$$

Let

$$\widetilde{\mathbf{c}}_{0,d}^{(i)} = (\widetilde{c}_{0,d,0}^{(i)}, \ldots, \widetilde{c}_{0,d,\lfloor \log q \rfloor}^{(i)}) \quad \text{and} \quad \widetilde{\mathbf{c}}_{1}^{(i)} = (\widetilde{c}_{1,d,0}^{(i)}, \ldots, \widetilde{c}_{1,d,\lfloor \log q \rfloor}^{(i)})$$
$$\widetilde{\mathbf{z}}_{0,d}^{(i)} = (\widetilde{z}_{0,d,0}^{(i)}, \ldots, \widetilde{z}_{0,d,\lfloor \log q \rfloor}^{(i)}) \quad \text{and} \quad \widetilde{\mathbf{z}}_{1}^{(i)} = (\widetilde{z}_{1,d,0}^{(i)}, \ldots, \widetilde{z}_{1,d,\lfloor \log q \rfloor}^{(i)})$$

and set the evaluation key

$$\mathsf{ek}_i := \left\{ \widetilde{\mathbf{c}}_{0,d}^{(i)}, \widetilde{\mathbf{c}}_{1,d}^{(i)}, \widetilde{\mathbf{z}}_{0,d}^{(i)}, \widetilde{\mathbf{z}}_{1,d}^{(i)} \right\}_{d \in [D]}$$

- $\mathsf{CombinePK}(\{\mathsf{pk}_i\}_{i \in [N]})$: Set $\mathsf{pk} := \mathsf{pk}_1 + \ldots + \mathsf{pk}_N$.

- $\mathsf{CombineKeys}(\{\mathsf{sk}_i\}_{i \in [N]}, \{\mathsf{ek}_i\}_{i \in [N]})$: Parse $\mathsf{sk}_i = (s_0^{(i)}, \ldots, s_D^{(i)})$ and define $s_j = \sum_{i=1}^N s_j^{(i)}$. Set

$$\mathsf{sk} := (s_0, \ldots, s_D)$$

  Parse $\mathsf{ek}_i = \left\{ \widetilde{\mathbf{c}}_{0,d}^{(i)}, \widetilde{\mathbf{c}}_{1,d}^{(i)}, \widetilde{\mathbf{z}}_{0,d}^{(i)}, \widetilde{\mathbf{z}}_{1,d}^{(i)} \right\}_{d \in [D]}$. The evaluation key is computed as in Section 6.3 for secret key $\mathsf{sk} = (s_0, \ldots, s_D)$. For all $d \in [D], \tau \in \{0, \ldots, \lfloor \log q_{d-1} \rfloor\}$, sample $(a_{d,\tau}, b_{d,\tau} = a_{d,\tau} s_d + 2e_{d,\tau}) \in R_{q_{d-1}}^2$ and $(a'_{d,\tau}, b'_{d,\tau} = a'_{d,\tau} s_d + 2e'_{d,\tau}) \in R_{q_{d-1}}^2$, where $a_{d,\tau}, a'_{d,\tau} \leftarrow R_{q_{d-1}}$ and $e_{d,\tau}, e'_{d,\tau} \leftarrow \chi$. Compute

$$\widetilde{c}_{0,d,\tau} := b_{d,\tau} + 2^\tau s_{d-1} \in R_{q_{d-1}} \qquad \text{and} \qquad \widetilde{c}_{1,d,\tau} := a_{d,\tau}$$
$$\widetilde{z}_{0,d,\tau} := b'_{d,\tau} + 2^\tau s_{d-1}^2 \in R_{q_{d-1}} \qquad \text{and} \qquad \widetilde{z}_{1,d,\tau} := a'_{d,\tau}$$

  Let

$$\widetilde{\mathbf{c}}_{0,d} = (\widetilde{c}_{0,d,0}, \ldots, \widetilde{c}_{0,d,\lfloor \log q \rfloor}) \quad \text{and} \quad \widetilde{\mathbf{c}}_{1} = (\widetilde{c}_{1,d,0}, \ldots, \widetilde{c}_{1,d,\lfloor \log q \rfloor})$$
$$\widetilde{\mathbf{z}}_{0,d} = (\widetilde{z}_{0,d,0}, \ldots, \widetilde{z}_{0,d,\lfloor \log q \rfloor}) \quad \text{and} \quad \widetilde{\mathbf{z}}_{1} = (\widetilde{z}_{1,d,0}, \ldots, \widetilde{z}_{1,d,\lfloor \log q \rfloor})$$

  and set

$$\mathsf{ek} := \{\widetilde{\mathbf{c}}_{0,d}, \widetilde{\mathbf{c}}_{1,d}, \widetilde{\mathbf{z}}_{0,d}, \widetilde{\mathbf{z}}_{1,d}\}_{d \in [D]}$$

- $\mathsf{Enc}_{\mathsf{sk}}(m, \mathrm{mode}; \mathbf{e})$: If $\|\mathbf{e}\| \geq B_{\mathrm{mode}}$, output $\perp$. Otherwise compute

$$c_0 := \langle \mathbf{p}, \mathbf{e} \rangle + m \in R_{q_0} \qquad \text{and} \qquad c_1 := \langle \mathbf{a}, \mathbf{e} \rangle$$

  and output $\mathbf{c} = (c_0, c_1)$. Also recall that when unspecified, we assume mode $= 0$.

Without loss of generality, assume below that ciphertext $\mathbf{c}$ is a ciphertext under secret key s_D and modulus q_D.

- $\mathsf{ShareDec}_{\mathsf{sk}_i}(\mathbf{c})$: Parse $\mathbf{c} = (c_0, c_1)$ and $\mathsf{sk}_i = (s_0^{(i)}, \ldots, s_D^{(i)})$. Sample $e^{(i)} \leftarrow \chi_0$ and output

$$\mu_i := c_1 s_D^{(i)} + 2e^{(i)} \in R_{q_D}$$

- $\mathsf{ShareCombine}(\mathbf{c}, \mu_1, \ldots, \mu_N)$: Parse $\mathbf{c} = (c_0, c_1)$. Output $c_0 - \sum_{i=1}^{N} \mu_i \pmod 2$.
- $\mathsf{ReRand}_{\mathsf{pk}}(\mathbf{c}, 1)$: Sample $\mathbf{e} \leftarrow \chi_1^n$ and output $\hat{\mathbf{c}} \leftarrow \mathbf{c} + \mathsf{Enc}_{\mathsf{pk}}(0\,; \mathbf{e})$.

Theorem 6.4. *The scheme described above is a key-homomorphic threshold FHE encryption scheme.*

Proof. (Sketch)

Correct Key Combination: It is easily seen that

$$\mathsf{pk} = \mathsf{pk}_1 + \ldots + \mathsf{pk}_N = \sum_{i=1}^{N} (\mathbf{a}s_0^{(i)} + \mathbf{x}^{(i)}) = \mathbf{a}\left(\sum_{i=1}^{N} s_0^{(i)}\right) + \left(\sum_{i=0}^{N} \mathbf{x}^{(i)}\right) = \mathbf{a}s_0 + \mathbf{x}'$$

Furthermore, ek is correct by construction.

Key-Simulation Indistinguishability: We define:

> $\mathsf{SimCombKeys}(\{(\mathsf{pk}_i, \mathsf{ek}_i)\}_{i\in T})$: Let $\overline{T} = [N]\backslash T$. For $j \in \overline{T}$, lets $\mathsf{pk}_j \leftarrow R_{q_0}$. Sets $\widetilde{\mathbf{c}}_{1,d}^{(j)} = (a_{d,1}^{(j)}, \ldots, a_{d,\lfloor \log q_{d-1} \rfloor}^{(j)})$ and $\widetilde{\mathbf{z}}_{1,d}^{(j)} = (a'^{(j)}_{d,1}, \ldots, a'^{(j)}_{d,\lfloor \log q_{d-1} \rfloor})$. Chooses $\widetilde{\mathbf{c}}_{0,d}^{(j)}, \widetilde{\mathbf{c}}_{0,d}^{(j)} \leftarrow R_{q_d}^{\lfloor \log q_{d-1} \rfloor}$, and sets $\mathsf{ek}_j = \left\{\widetilde{\mathbf{c}}_{0,d}^{(j)}, \widetilde{\mathbf{c}}_{1,d}^{(j)}, \widetilde{\mathbf{z}}_{0,d}^{(j)}, \widetilde{\mathbf{z}}_{1,d}^{(j)}\right\}_{d\in[D]}$. Chooses $\mathsf{ek} = (\widetilde{\mathbf{c}}_{0,d}, \widetilde{\mathbf{c}}_{1,d}, \widetilde{\mathbf{z}}_{0,d}, \widetilde{\mathbf{z}}_{1,d}) \leftarrow R_{q_{d-1}}^{4\lfloor \log q_{d-1} \rfloor}$.

We use a hybrid argument, one per each $d \in \{0, \ldots, D\}$. First, by PLWE with secret $s_0^{(j)}$, we know that pk_j and $(\widetilde{\mathbf{c}}_{0,d}^{(j)}, \widetilde{\mathbf{c}}_{1,d}^{(j)}, \widetilde{\mathbf{z}}_{0,d}^{(j)}, \widetilde{\mathbf{z}}_{1,d}^{(j)})$ as output by $\mathsf{SimCombKeys}$ is computationally indistinguishable from an honestly created public key and honestly created tuples. The remaining hybrids are similar. For $d \neq 0$, by PLWE with secret $s_d^{(j)}$, we know that $(\widetilde{\mathbf{c}}_{0,d}^{(j)}, \widetilde{\mathbf{c}}_{1,d}^{(j)}, \widetilde{\mathbf{z}}_{0,d}^{(j)}, \widetilde{\mathbf{z}}_{1,d}^{(j)})$ as output by $\mathsf{SimCombKeys}$ is computationally indistinguishable from honestly created tuples, and therefore so is ek_j. By PLWE with secret $s_d = \sum_{i=1}^{N} s_d^{(i)}$, which is computationally hidden from $\mathcal{A}$ (since $\mathcal{A}$ only knows $\{s_d^{(i)}\}_{i\in T}$), we know that ek as output by $\mathsf{SimCombKeys}$ is computationally indistinguishable from an honestly created ek.

Correct Share Decryption of Combined Key: Let $\mathbf{c} = (c_0, c_1)$ be an encryption of m under secret key s_D. Then $c_0 = c_1 s_D + 2e + m$ and $\mathsf{Dec}_{\mathsf{sk}}(\mathbf{c}) = m$ as long as $\|e\| \ll q_D/2$. Let $\mu_i = c_1 s_D^{(i)} + 2e^{(i)}$. Then

$$\begin{aligned}\mathsf{ShareCombine}(\mathbf{c}, \mu_1, \ldots, \mu_N) &= c_0 - \sum_{i=1}^{N} \mu_i = (c_1 s_D + 2e + m) - \left(\sum_{i=1}^{N} c_1 s_D^{(i)} + 2e^{(i)}\right) \pmod 2 \\ &= c_1 s_D - c_1 \sum_{i=1}^{N} s_D^{(i)} + 2\left(e - \sum_{i=1}^{N} e^{(i)}\right) + m \pmod 2 = m,\end{aligned}$$

as long as $\left\| e - \sum_{i=1}^{N} e^{(i)} \right\| \ll q_D/2$

Share-Simulation Indistinguishability: We define:

> $\mathsf{SimShareDec}(\mathbf{c}, m, \{\mu_i\}_{i\in T})$: Parse $\mathbf{c} = (c_0, c_1)$. Fix $j^* \in \overline{T}$. For all $j \neq j^*$, let $\mu_j \leftarrow R_{q_D}$. Let $\mu_{j^*} := c_0 - m \ - \sum_{i\in[N], i\neq j^*} \mu_i$ and output $\{\mu_j\}_{j\in\overline{T}}$

By correct share decryption, we know that regardless of how $\{\mu_j\}_{j\in\overline{T}}$ were created, $\mu_{j^*} := c_0 - m - \sum_{i\in[N],i\neq j^*}\mu_i$. Since this is a deterministic function of the rest of the variables, we simply need to prove that

$$\left\{\mu_j := c_1 s_D^{(j)} + 2e^{(j)}\right\}_{j\in\overline{T},j\neq j^*} \stackrel{c}{\approx} \{u_j \leftarrow R_{q_D}\}_{j\in\overline{T},j\neq j^*} \tag{1}$$

Because $\mathbf{c} \leftarrow \mathsf{Enc}_{\mathsf{sk}}(m)$, we know that the distribution of c_1 is statistically close to uniform in R_{q_D} and thus, (1) simply follows from a series of hybrid arguments, one for each j, in which we use the pseudo-randomness of a PLWE sample with secret $s_D^{(j)}$.

Semantic Security: Semantic security follows from the pseudorandomness of PLWE. Given any $\{\mathsf{pk}_i\}_{i\in T}$, if $\{\mathsf{pk}_j\}_{j\in\overline{T}}$ are honestly generated then in partiparticular, they are pseudorandom (by PLWE), and thus $\mathsf{pk} = \mathsf{pk}_1 + \ldots + \mathsf{pk}_N$ is also pseudorandom. Thus, a ciphertext encrypted under this public key is pseudorandom and semantic security follows.

Ciphertext Re-Randomization: The output of $c = \mathsf{Eval}_{\mathsf{ek}}(c_1,\ldots,c_\ell)$ is not indistinguishable from a fresh encryption $c' \leftarrow \mathsf{Enc}_{\mathsf{pk}}(y,0)$ since the noise in c might be much larger than in c'. We remedy this by adding noise that is super-polynomially bigger than the noise in c (in the form of an encryption of 0, $\mathsf{Enc}_{\mathsf{pk}}(0,1)$). We choose χ_0 and χ_1 appropriately. Recall that χ_0 and χ_1 are (truncated) discrete Gaussians $D_{\mathbb{Z}^n,r_0,B(r_0)}$ and $D_{\mathbb{Z}^n,r_1,B(r_1)}$, which are statistically close to the discrete Gaussian distributions $D_{\mathbb{Z}^n,r_0}$ and $D_{\mathbb{Z}^n,r_1}$. We choose r_1 to a be a super-polynomial factor larger than r_0: $r_1 \geq 2^{\omega(\log\kappa)}r_0$. Recall that

$$\mathsf{ReRand}(c) = \mathsf{Eval}_{\mathsf{ek}}(c_1,\ldots,c_\ell) + \mathsf{Enc}_{\mathsf{pk}}(0,1)$$

where the first term, $\mathsf{Eval}_{\mathsf{ek}}(c_1,\ldots,c_\ell)$ has noise distributed according to $D_{\mathbb{Z}^n,\mathrm{poly}(\kappa)\cdot r_0}$, while the second has noise distributed according to $D_{\mathbb{Z}^n,r_1}$. Because r_1 is super-polynomially bigger than r_0, we have that the noise distributions of $\mathsf{ReRand}(c)$ and a fresh encryption $\mathsf{Enc}_{\mathsf{pk}}(y,1)$ are statistically close.

□

7 Implementing $\mathcal{F}^{\mathcal{E}}_{\mathbf{rand.bc}}$ and $\mathcal{F}^{\mathcal{E}}_{\mathbf{keys}}$

We show how to implement $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ and $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$ for our key-homomorphic threshold FHE scheme.

7.1 Ciphertext Re-Randomization

Protocol $\Pi^{\text{SH}}_{\text{RAND}}$ shown in Figure 11 securely implements $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$ in the $\mathcal{F}_{\text{BC}}$-hybrid model, against (static) semi-honest adversaries. In $\Pi^{\text{SH}}_{\text{RAND}}$, we simply ask each party to send a random encryption of 0. Since corrupted parties are semi-honest, this re-randomizes the ciphertext.

Theorem 7.1. *Protocol $\Pi^{\text{SH}}_{\text{RAND}}$ described in Figure 11 securely implements $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$ against (static) semi-honest adversaries, in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}}, \mathcal{F}^{\mathcal{E}}_{\text{KEYS}})$-hybrid model.*

The proof of Theorem 7.1 is given in Appendix A.2. Protocol $\Pi^{\text{MAL}}_{\text{RAND}}$ shown in Figure 12 securely implements $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}}, \mathcal{F}^{R(zero)}_{\text{ZK.BC}})$-hybrid model. To tolerate malicious adversaries, we need to guarantee that the encryption z_i is a "fresh" encryption; in particular, we need to guarantee that z_i is not correlated to any other z_j. In order to guarantee this, we place a commitment key for a trapdoor

Protocol $\Pi^{\text{SH}}_{\text{RAND}}$

Inputs: All parties $P_1, \ldots, P_N$ and server S get public key pk as input. Server S additionally gets a ciphertext $c \in \mathcal{C}_{\mathsf{pk}}$, where $\mathcal{C}_{\mathsf{pk}}$ is the ciphertext space defined by pk.

For session ID sid:

Round 1: Each party P_i samples $z_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0,1)$ and sends (sid, i, z_i) to $\mathcal{F}_{\text{BC}}$.

Round 2: Upon receiving (sid, i, z_i) for all $i \in [N]$ from $\mathcal{F}_{\text{BC}}$, server computes $\hat{c} := c + \sum_{i=1}^{N} z_i$ and sends $(\mathsf{sid}, 0, c, \hat{c})$ to $\mathcal{F}_{\text{BC}}$.

Figure 11: Protocol $\Pi^{\text{SH}}_{\text{RAND}}$. Implements $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$ in the $\mathcal{F}_{\text{BC}}$-hybrid model. Secure against semi-honest adversaries.

commitment scheme $(\mathsf{Keygen}^{\mathsf{Com}}, \mathsf{Com})$ in the crs. This way we know that it is correctly distributed. We then have each party commit in advance, using ck, to the randomess $r_i{}^{(0)}$ they will use to create z_i. Along with z_i parties also provide a SS-NIZK proof that z_i was indeed computed with randomness $r_i{}^{(0)}$. Formally, we require each party to compute a SS-NIZK for the NP-relation:

$$R^{(zero)} = \{ \ (\ \rho \ , \ \mathsf{ck}, z, \ \mathsf{pk} \ ) \ , \ (\ r \ , \ s \ ) \ \mid \ \rho \leftarrow \mathsf{Com}_{\mathsf{ck}}(r \ ; s) \ \wedge \ z := \mathsf{Enc}_{\mathsf{pk}}(0, 1 \ ; r) \ \}$$

Protocol $\Pi^{\text{MAL}}_{\text{RAND}}$

Inputs: All parties $P_1, \ldots, P_N$ and server S get public key pk as input. Server S additionally gets a ciphertext $c \in \mathcal{C}_{\mathsf{pk}}$, where $\mathcal{C}_{\mathsf{pk}}$ is the ciphertext space defined by pk.

For session ID sid:

Round 1 Each party P_i:

- Calls $\mathcal{F}^{\mathcal{D}}_{\text{CRS}}$ with sid and receives $(\mathsf{sid}, \mathsf{ck})$. Chooses encryption and commitment randomness $r_i{}^{(0)}$ and $s_i{}^{(0)}$, and computes a commitment $\rho_i{}^{(0)} \leftarrow \mathsf{Com}(r_i{}^{(0)}; s_i{}^{(0)})$.
- Sends $(\mathsf{sid}, i, \rho_i{}^{(0)})$ to $\mathcal{F}_{\text{BC}}$.

Round 2: Each party P_i, samples $z_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0, 1; r_i{}^{(0)})$ and sends $(\mathsf{sid}, i, (\rho_i{}^{(0)}, z_i, \mathsf{pk}), (r_i{}^{(0)}, s_i{}^{(0)}))$ to $\mathcal{F}^{R^{(zero)}}_{\text{ZK.BC}}$.

Round 3: The server S:

- Upon receiving $(\mathsf{sid}, i, (\rho_i{}^{(0)}, z_i, \mathsf{pk}))$ from $\mathcal{F}^{R^{(zero)}}_{\text{ZK.BC}}$ for every $i \in [N]$, compute $\hat{c} := c + \sum_{i=1}^{N} z_i$ and send $(\mathsf{sid}, S, c, \hat{c})$ to $\mathcal{F}_{\text{BC}}$.

Figure 12: Protocol $\Pi^{\text{MAL}}_{\text{RAND}}$. Implements $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$ in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}}, \mathcal{F}^{R^{(zero)}}_{\text{ZK.BC}})$-hybrid model, where $\mathcal{D}$ is the distribution defined by $\mathsf{Keygen}^{\mathsf{Com}}$. Secure against malicious adversaries.

Theorem 7.2. *Protocol $\Pi^{\text{MAL}}_{\text{RAND}}$ described in Figure 12 securely implements $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$ against (static) malicioius adversaries, in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}}, \mathcal{F}^{\mathcal{E}}_{\text{KEYS}}, \mathcal{F}^{R^{(zero)}}_{\text{ZK.BC}})$-hybrid model.*

The proof of Theorem 7.2 is given in Appendix A.2.

7.2 Key Combination

Protocol $\Pi^{\text{SH}}_{\text{KEYS}}$ shown in Figure 13 securely implements $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ in the $\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}}$-hybrid model, against (static) semi-honest adversaries. We wish to obtain pseudoencryptions of s_{d-1} and $(s_{d-1})^2$, where s_{d-1} is the combined secret key $s_d = \sum_{i=1}^{N} s_{d-1}^{(i)}$. In $\Pi^{\text{SH}}_{\text{KEYS}}$, parties begin by sending encryptions of

0, $w_{d,\tau}^{(k)}, w'^{(k)}_{d,\tau}$ using each of the $a_{d,\tau}^{(k)}, a'^{(k)}_{d,\tau}$ in params. Using these and the pseudoencryptions in the evaluation keys ek_k, by key-homomorphism they can then compute pseudoencryptions $\widetilde{x}_{d,\tau}^{(k)}$ of $2^\tau s_{d-1}^{(k)}$ under the *combined* secret key s_d. Adding these yields a pseudoencryption of $2^\tau s_{d-1}$. Furthermore, multiplying these by $s_{d-1}^{(i)}$ and re-randomizing, yields a pseudoencryption of $2^\tau s_{d-1}^{(i)} s_{d-1}^{(k)}$, under the combined secret key s_d. Adding these pseudoencryptions for all k, i yields a pseudoencryption of $2^\tau (s_d)^2$:

$$2^\tau \sum_{i,k=1}^{N} s_{d-1}^{(i)} s_{d-1}^{(k)} = 2^\tau (\sum_{i}^{N} s_d^{(i)})^2 = 2^\tau (s_d)^2$$

Theorem 7.3. *Protocol $\Pi^{\text{SH}}_{\text{KEYS}}$ described in Figure 13 securely implements $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ against (static) semi-honest adversaries, in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}})$-hybrid model.*

The proof of Theorem 7.3 is given in Appendix A.3. Figure 14 describes protocol $\Pi^{\text{MAL}}_{\text{KEYS}}$ that securely implements $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}}, \mathcal{F}^{R^{(enc)}}_{\text{ZK.BC}}, \mathcal{F}^{R^{(rand)}}_{\text{ZK.BC}})$-hybrid model, against (static) malicious adversaries. To tolerate malicious adversaries, all we have to change is to have the parties provide SS-NIZK proofs that they computed the $w_{d,\tau}^{(i)}$'s correctly, as encryptions of 0, using secret $s_d^{(i)}$, and also that multiplied in $s_{d-1}^{(i)}$ and re-randomized correctly. More formally, they provide SS-NIZK proofs for the following NP relations:

$$R^{(enc)} = \{ \ (a, c) \ , \ (s, e) \ \mid \ c = as + 2e \ \}$$
$$R^{(rand)} = \{ \ (\mathbf{c}, \mathbf{c}', \mathsf{pk}) \ , \ (s, r) \ \mid \ \mathbf{c}' = s\mathbf{c} + \mathsf{Enc}_{\mathsf{pk}}(0 \ ; r) \ \}$$

Theorem 7.4. *Protocol $\Pi^{\text{MAL}}_{\text{KEYS}}$ described in Figure 14 securely implements $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ against (static) malicious adversaries, in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}}, \mathcal{F}^{R^{(enc)}}_{\text{ZK.BC}}, \mathcal{F}^{R^{(rand)}}_{\text{ZK.BC}})$-hybrid model.*

The proof of Theorem 7.4 is given in Appendix A.3.

8 Performance

In this section we argue that the protocol from Section 4 can be executed in 4 rounds in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}})$-hybrid model. We also analyze the communication complexity and the computation time of the parties P_i and server S. We first look at the semi-honest case and then turn our attention to the changes needed to the protocol secure against malicious adversaries.

In Section 7, we described protocols $\Pi^{\text{SH}}_{\text{RAND}}$ and $\Pi^{\text{SH}}_{\text{KEYS}}$ that securely implement the functionality $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$, each in 2 rounds. Since the circuit C is never used in these protocols, the communication complexity and the computation time of all parties and the server in $\Pi^{\text{SH}}_{\text{RAND}}$ and $\Pi^{\text{SH}}_{\text{KEYS}}$ is independent of $|C|$. We want to run these protocols inside Π^{SH} in the least number of rounds, subject to the following restrictions:

- Parties must know pk before Round 1 of $\Pi^{\text{SH}}_{\text{RAND}}$.
- Round 2 of $\Pi^{\text{SH}}_{\text{RAND}}$ must be placed in Phase 3 of Π^{SH}.
- Server must know ek in (and therefore $\Pi^{\text{SH}}_{\text{KEYS}}$ must be completed by) Phase 3 of Π^{SH}, so that the server can perform homomorphic operations.

We do this by sending some of the messages in parallel:

Protocol $\Pi^{\text{SH}}_{\text{KEYS}}$

Inputs: All parties P_i get as input a valid key tuple $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$, and for $j \neq i$, P_i and S also receive $(\mathsf{pk}_j, \mathsf{ek}_j)$.

For session ID sid, each party P_i:

Round 1: Calls $\mathcal{F}^{\mathcal{D}}_{\text{CRS}}$ with sid and receives $(\mathsf{sid}, \mathsf{params})$, where $\mathsf{params} = \left(\ldots, \left\{a^{(k)}_{d,\tau}, a'^{(k)}_{d,\tau}\right\}_{d,\tau,k}, \ldots\right)$.

For $k \in [N]$:

- Parses $\mathsf{ek}_k := \left\{\widetilde{\mathbf{c}}^{(k)}_{0,d}, \widetilde{\mathbf{c}}^{(k)}_{1,d}, \widetilde{\mathbf{z}}^{(k)}_{0,d}, \widetilde{\mathbf{z}}^{(k)}_{1,d}\right\}_{d\in[D]}$.
- Verifies that $\widetilde{\mathbf{c}}^{(k)}_{1,d} = (a^{(k)}_{d,1}, \ldots, a^{(k)}_{d,\lfloor \log q_{d-1}\rfloor})$ and $\widetilde{\mathbf{z}}^{(k)}_{1,d} = (a'^{(k)}_{d,1}, \ldots, a'^{(k)}_{d,\lfloor \log q_{d-1}\rfloor})$
- Samples $e^{(i,k)}_{d,\tau} \leftarrow \chi_0$ and computes

$$w^{(i,k)}_{d,\tau} := a^{(k)}_{d,\tau} s^{(i)}_d + 2e^{(i,k)}_{d,\tau}$$

- Sends $(\mathsf{sid}, i, w^{(i,k)}_{d,\tau})$ to $\mathcal{F}_{\text{BC}}$.

Round 2: Upon receiving $(\mathsf{sid}, j, w^{(j,k)}_{d,\tau})$ from $\mathcal{F}_{\text{BC}}$ for all $j, k \in [N]$, $d \in [D]$, $\tau \in \{0, \ldots, \lfloor \log q_{d-1}\rfloor\}$:

- Computes

$$\widetilde{x}^{(k)}_{0,d,\tau} := \sum_{j\neq k} w^{(j,k)}_{d,\tau} + c^{(k)}_{0,d,\tau} = a^{(k)}_{d,\tau} s_d + 2\left(\sum_{j\neq k} e^{(j,k)}_{d,\tau} + e^{(k)}_{d,\tau}\right) + 2^\tau s^{(k)}_{d-1} \quad \text{and} \quad \widetilde{x}^{(k)}_{1,d,\tau} = a^{(k)}_{d,\tau}$$

$$\widetilde{y}^{(i,k)}_{0,d,\tau} = s^{(i)}_{d-1}\widetilde{x}^{(k)}_{d,\tau} \quad \text{and} \quad \widetilde{y}^{(i,k)}_{1,d,\tau} = s^{(i)}_{d-1} a^{(k)}_{d,\tau}$$

- Let $\widetilde{\mathbf{x}}^{(k)}_{d,\tau} = (\widetilde{x}^{(k)}_{0,d,\tau}, \widetilde{x}^{(k)}_{1,d,\tau})$ and $\widetilde{\mathbf{y}}^{(i,k)}_{d,\tau} = (\widetilde{y}^{(i,k)}_{0,d,\tau}, \widetilde{y}^{(i,k)}_{1,d,\tau})$. Computes

$$\widetilde{\mathbf{c}}^{(k)}_{d,\tau} = (\widetilde{c}^{(k)}_{0,d,\tau}, \widetilde{c}^{(k)}_{1,d,\tau}) \leftarrow \widetilde{\mathbf{x}}^{(k)}_{d,\tau} + \mathsf{Enc}_{\mathsf{pk}}(0)$$

$$\widetilde{\mathbf{z}}^{(i,k)}_{d,\tau} = (\widetilde{z}^{(i,k)}_{0,d,\tau}, \widetilde{z}^{(i,k)}_{1,d,\tau}) \leftarrow \widetilde{\mathbf{y}}^{(i,k)}_{d,\tau} + \mathsf{Enc}_{\mathsf{pk}}(0)$$

- Sends $(\mathsf{sid}, i, \widetilde{\mathbf{c}}^{(k)}_{d,\tau}, \widetilde{\mathbf{z}}^{(i,k)}_{d,\tau})$ to $\mathcal{F}_{\text{BC}}$.

(Local Computation): Upon receiving $(\mathsf{sid}, j, \widetilde{\mathbf{c}}^{(k)}_{d,\tau}, \widetilde{\mathbf{z}}^{(j,k)}_{d,\tau})$ from $\mathcal{F}_{\text{BC}}$ for all $j, k \in [N]$, $d \in [D]$, $\tau \in \{0, \ldots, \lfloor \log q_{d-1}\rfloor\}$:

- Computes

$$\widetilde{c}_{0,d,\tau} := \sum_k \widetilde{c}^{(k)}_{0,d,\tau} \quad \text{and} \quad \widetilde{c}_{1,d,\tau} := \sum_k \widetilde{c}^{(k)}_{1,d,\tau}$$

$$\widetilde{z}_{0,d,\tau} := \sum_{j,k} \widetilde{z}^{(j,k)}_{0,d,\tau} \quad \text{and} \quad \widetilde{z}_{1,d,\tau} := \sum_{j,k} \widetilde{z}^{(j,k)}_{1,d,\tau}$$

- For $b \in \{0,1\}$, sets $\widetilde{\mathbf{c}}_{b,d} := (\widetilde{c}_{b,d,1}, \ldots, \widetilde{c}_{b,d,\lfloor \log q_{d-1}\rfloor})$ and $\widetilde{\mathbf{z}}_{b,d} := (\widetilde{z}_{b,d,1}, \ldots, \widetilde{z}_{b,d,\lfloor \log q_{d-1}\rfloor})$, and $\mathsf{ek} := (\widetilde{\mathbf{c}}_{0,d}, \widetilde{\mathbf{c}}_{1,d}, \widetilde{\mathbf{z}}_{0,d}, \widetilde{\mathbf{z}}_{1,d})$.

Figure 13: Protocol $\Pi^{\text{SH}}_{\text{KEYS}}$. Implements $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}})$-hybrid model, where $\mathcal{D}$ is the distribution defined by Setup. Secure against semi-honest adversaries.

Round 1: Each party computes its key tuple, as in Π^{SH}, and also runs Round 1 of $\Pi^{\text{SH}}_{\text{KEYS}}$

Round 2: Each party computes an encryption of its input, as in Π^{SH}. Since all parties know pk, also runs Round 1 of $\Pi^{\text{SH}}_{\text{RAND}}$ and Round 2 of $\Pi^{\text{SH}}_{\text{KEYS}}$.

Protocol $\Pi^{\text{MAL}}_{\text{KEYS}}$

Inputs: All parties P_i get as input a valid key tuple $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$, and for $j \neq i$, P_i and S also receive $(\mathsf{pk}_j, \mathsf{ek}_j)$.

For session ID sid, each party P_i:

Round 1: Calls $\mathcal{F}^{\mathcal{D}}_{\text{CRS}}$ with sid and receives $(\mathsf{sid}, \mathsf{params})$, where $\mathsf{params} = \left(\ldots, \left\{a^{(k)}_{d,\tau}, a'^{(k)}_{d,\tau}\right\}_{d,\tau,k}, \ldots\right)$.

For $k \in [N]$,

- Parses $\mathsf{ek}_k := \left\{\widetilde{\mathbf{c}}^{(k)}_{0,d}, \widetilde{\mathbf{c}}^{(k)}_{1,d}, \widetilde{\mathbf{z}}^{(k)}_{0,d}, \widetilde{\mathbf{z}}^{(k)}_{1,d}\right\}_{d\in[D]}$.
- Verifies that $\widetilde{\mathbf{c}}^{(k)}_{1,d} = (a^{(k)}_{d,1}, \ldots, a^{(k)}_{d,\lfloor \log q_{d-1}\rfloor})$ and $\widetilde{\mathbf{z}}^{(k)}_{1,d} = (a'^{(k)}_{d,1}, \ldots, a'^{(k)}_{d,\lfloor \log q_{d-1}\rfloor})$
- Samples $e^{(i,k)}_{d,\tau} \leftarrow \chi_0$ and computes

$$w^{(i,k)}_{d,\tau} := a^{(k)}_{d,\tau} s^{(i)}_d + 2e^{(i,k)}_{d,\tau}$$

- Sends $(\mathsf{sid}, i, (w^{(i,k)}_{d,\tau}, a^{(i,k)}_{d,\tau}), (s^{(i)}_d, e^{(i,\tau)}_{d,\tau}))$ to $\mathcal{F}^{R^{(enc)}}_{\text{ZK.BC}}$.

Round 2: Upon receiving $(\mathsf{sid}, j, (w^{(j,k)}_{d,\tau}, a^{(j,k)}_{d,\tau}))$ from $\mathcal{F}^{R^{(enc)}}_{\text{ZK.BC}}$ for all $j, k \in [N]$, $d \in [D]$, $\tau \in \{0, \ldots, \lfloor \log q_{d-1}\rfloor\}$:

- Computes

$$\widetilde{x}^{(k)}_{0,d,\tau} := \sum_{j\neq k} w^{(j,k)}_{d,\tau} + c^{(k)}_{0,d,\tau} = a^{(k)}_{d,\tau} s_d + 2\left(\sum_{j\neq k} e^{(j,k)}_{d,\tau} + e^{(k)}_{d,\tau}\right) + 2^\tau s^{(k)}_{d-1} \quad \text{and} \quad \widetilde{x}^{(k)}_{1,d,\tau} = a^{(k)}_{d,\tau}$$

$$\widetilde{y}^{(i,k)}_{0,d,\tau} = s^{(i)}_{d-1}\widetilde{x}^{(k)}_{d,\tau} \quad \text{and} \quad \widetilde{y}^{(i,k)}_{1,d,\tau} = s^{(i)}_{d-1} a^{(k)}_{d,\tau}$$

- Let $\widetilde{\mathbf{x}}^{(k)}_{d,\tau} = (\widetilde{x}^{(k)}_{0,d,\tau}, \widetilde{x}^{(k)}_{1,d,\tau})$ and $\widetilde{\mathbf{y}}^{(i,k)}_{d,\tau} = (\widetilde{y}^{(i,k)}_{0,d,\tau}, \widetilde{y}^{(i,k)}_{1,d,\tau})$. Chooses randomness $r^{(k)}_{d,\tau}, r^{(i,k)}_{d,\tau}$ and computes

$$\widetilde{\mathbf{c}}^{(k)}_{d,\tau} = (\widetilde{c}^{(k)}_{0,d,\tau}, \widetilde{c}^{(k)}_{1,d,\tau}) \leftarrow \widetilde{\mathbf{x}}^{(k)}_{d,\tau} + \mathsf{Enc}_{\mathsf{pk}}(0\,; r^{(k)}_{d,\tau})$$

$$\widetilde{\mathbf{z}}^{(i,k)}_{d,\tau} = (\widetilde{z}^{(i,k)}_{0,d,\tau}, \widetilde{z}^{(i,k)}_{1,d,\tau}) \leftarrow \widetilde{\mathbf{y}}^{(i,k)}_{d,\tau} + \mathsf{Enc}_{\mathsf{pk}}(0\,; r^{(i,k)}_{d,\tau})$$

- Sends $(\mathsf{sid}, i, (\widetilde{\mathbf{c}}^{(k)}_{d,\tau}, \widetilde{\mathbf{x}}^{(k)}_{d,\tau}), (1, r^{(k)}_{d,\tau}))$ and $(\mathsf{sid}, i, (\widetilde{\mathbf{z}}^{(i,k)}_{d,\tau}, \widetilde{\mathbf{x}}^{(k)}_{d,\tau}), (s^{(i)}_d, r^{(i,k)}_{d,\tau}))$ to $\mathcal{F}^{R^{(rand)}}_{\text{ZK.BC}}$.

(Local Computation): Upon receiving $(\mathsf{sid}, j, (\widetilde{\mathbf{c}}^{(k)}_{d,\tau}, \widetilde{\mathbf{x}}^{(k)}_{d,\tau}))$ and $(\mathsf{sid}, j, (\widetilde{\mathbf{z}}^{(j,k)}_{d,\tau}, \widetilde{\mathbf{x}}^{(k)}_{d,\tau})$ from $\mathcal{F}^{R^{(rand)}}_{\text{ZK.BC}}$ for all $j, k \in [N]$, $d \in [D]$, $\tau \in \{0, \ldots, \lfloor \log q_{d-1}\rfloor\}$:

- Computes

$$\widetilde{c}_{0,d,\tau} := \sum_k \widetilde{c}^{(k)}_{0,d,\tau} \quad \text{and} \quad \widetilde{c}_{1,d,\tau} := \sum_k \widetilde{c}^{(k)}_{1,d,\tau}$$

$$\widetilde{z}_{0,d,\tau} := \sum_{j,k} \widetilde{z}^{(j,k)}_{0,d,\tau} \quad \text{and} \quad \widetilde{z}_{1,d,\tau} := \sum_{j,k} \widetilde{z}^{(j,k)}_{1,d,\tau}$$

- For $b \in \{0,1\}$, sets $\widetilde{\mathbf{c}}_{b,d} := (\widetilde{c}_{b,d,1}, \ldots, \widetilde{c}_{b,d,\lfloor \log q_{d-1}\rfloor})$ and $\widetilde{\mathbf{z}}_{b,d} := (\widetilde{z}_{b,d,1}, \ldots, \widetilde{z}_{b,d,\lfloor \log q_{d-1}\rfloor})$, and $\mathsf{ek} := (\widetilde{\mathbf{c}}_{0,d}, \widetilde{\mathbf{c}}_{1,d}, \widetilde{\mathbf{z}}_{0,d}, \widetilde{\mathbf{z}}_{1,d})$.

Figure 14: Protocol $\Pi^{\text{MAL}}_{\text{KEYS}}$. Implements $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}}, \mathcal{F}^{R^{(enc)}}_{\text{ZK.BC}}, \mathcal{F}^{R^{(rand)}}_{\text{ZK.BC}})$-hybrid model, where $\mathcal{D}$ is the distribution defined by Setup. Secure against malicious adversaries.

Round 3: The server knows ek because $\Pi^{\text{SH}}_{\text{KEYS}}$ was completed in the previous round. The server performs the homomorphic evaluation, as in Π^{SH}, and runs Round 2 or $\Pi^{\text{SH}}_{\text{RAND}}$.

Round 4: The parties share decryption shares as in $\Pi^{\text{SH}}_{\text{KEYS}}$.

(Local Computation): The parties combine the ciphertext and decryption shares to obtain the output.

It is clear that the communication complexity and the computation time of the parties in the resulting protocol is *independent* of $|C|$, and polynomial in $(\sum |x_i|, |y|, N, \kappa)$. The computation time of the server, on the other hand, is linear in $|C|$ and polynomial in $(\sum |x_i|, |y|, N, \kappa)$.

In the malicious case, $\Pi^{\text{MAL}}_{\text{KEYS}}$ has exactly the same number of rounds and same restrictions as $\Pi^{\text{SH}}_{\text{KEYS}}$, but the protocols Π^{MAL} and $\Pi^{\text{MAL}}_{\text{RAND}}$ require one more round than their semi-honest counterparts, where the parties commit to the randomness they will use throughout the protocol. Therefore, we can achieve 5 rounds, in exactly the same way as before, except that we have one more round in the beginning, where the parties send commitments to randomness. Furthermore, the server must provide and the parties verify a short argument for the validity of c, that is, for the correct homomorphic evaluation of $|C|$. As we saw in Section 5, this can be done in one of four ways. Table 1 summarizes the round and communication complexity of the resulting protocol for each of the 4 options, together the computation time of the parties. The computation time of the server is always polynomial in $|C|$, as well as in $(\sum |x_i|, |y|, N, \kappa)$.

VC Proof	**RC**	**CC**(poly(·))	**CT**(poly(·))	**Model / Assumptions**
[Kil92, Kil95]	7	$\log(|C|)$	$\log(|C|)$	CRHF
[Mic94]	5	$\log(|C|)$	$\log(|C|)$	Random Oracle
[BCCT11, GLR11]	5	$\log(|C|)$	$\log(|C|)$	Non-Falsifiable
[GKR08]	5	$d, \log(|C|)$	$d, \log(|C|)$	Secure PIR Scheme

Table 1: Performance of Π^{MAL} in the $(\mathcal{F}_{\text{BC}}, \mathcal{F}^{\mathcal{D}}_{\text{CRS}})$-hybrid model, for N clients and circuit C of depth d. Communication complexity (**CC**) and computation time (**CT**) displayed are in addition to poly$(\sum |x_i|, |y|, N, \kappa)$.

References

[AiCPS09] Benny Applebaum, David Cash inand Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, 2009.

[BCCT11] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. *Cryptology ePrint Archive: Report 2011/443*, 2011.

[BD10] Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 201–218. Springer, 2010.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2011.

[BFKL93] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 1993.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Manuscript (in submission)*, 2011.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *To Appear in FOCS*, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Rogaway [Rog11], pages 505–524.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[DPSZ11] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2011/535, 2011. http://eprint.iacr.org/.

[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, *STOC*, pages 169–178. ACM, 2009.

[Gil10] Henri Gilbert, editor. *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*. Springer, 2010.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Cynthia Dwork, editor, *STOC*, pages 113–122. ACM, 2008.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. *Cryptology ePrint Archive: Report 2011/456*, 2011.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Rogaway [Rog11], pages 132–150.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732. ACM, 1992.

[Kil95] Joe Kilian. Improved efficient arguments (preliminary version). In Don Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 311–324. Springer, 1995.

[KMR11] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011. http://eprint.iacr.org/.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Gilbert [Gil10], pages 1–23.

[Mic94] Silvio Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453. IEEE, 1994.

[MSas11] Steven Myers, Mona Sergi, and abhi shelat. Threshold fully homomorphic encryption and secure computation. Cryptology ePrint Archive, Report 2011/454, 2011. http://eprint.iacr.org/.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2003.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *STOC*, pages 84–93. ACM, 2005.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[Rog11] Phillip Rogaway, editor. *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*. Springer, 2011.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.

[SP92] Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *FOCS*, pages 427–436, 1992.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography*, pages 420–443, 2010.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Gilbert [Gil10], pages 24–43.

A Proofs

A.1 Security Under Malicious Adversaries

Proof of Lemma 5.1: Correctness follows from correctness of key combination, correctness of homomorphic evaluation, correctness of share decryption of $\mathcal{E}$, and correctness of the NIZK proof systems $\Pi^{(x)}, \Pi^{(gen)}, \Pi^{(dec)}$ and the short argument system Φ.

We now turn to security. To show more clearly why we need simulation-sound and simulation-extractable NIZKs (as opposed to standard NIZKs), we instantiate in Π^{MAL} from Figure 10, $\mathcal{F}_{\text{ZK.BC}}^{R^{(gen)}}$, $\mathcal{F}_{\text{ZK.BC}}^{R^{(dec)}}$ with SS-NIZKs, and $\mathcal{F}_{\text{ZK.BC}}^{R^{(x)}}$ with a SE-NIZK and present the proof of Lemma 5.1 after this instantiation. Let $\mathcal{A}$ be a real-world malicious adversary corrupting $t < N$ clients, and possibly the server. Let $T \subsetneq [N]$ be the set of corrupted clients. We describe the simulator $\mathcal{S}^{\text{MAL}}$ in Figure 15. We prove that $\text{REAL}_{\Pi^{\text{MAL}},\mathcal{A}}(\vec{x}) \overset{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S}^{\text{MAL}}}(\vec{x})$ via a series of hybrids. The view of the $\mathcal{A}$ consists of

$$\mathsf{View} = \Big(\mathsf{params}, \Big\{x_i, \rho_i^{(gen)}, \rho_i^{(dec)}, (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i), \pi_i^{(gen)}, c_i, \pi_i^{(x)}, \mu_i, \pi_i^{(dec)}\Big\}_{i\in T},$$
$$\Big\{(\rho_j^{(gen)}, \rho_j^{(dec)})\Big\}_{j\in\overline{T}}, \mathsf{pk}, \mathsf{ek}, \Big\{(\mathsf{pk}_j, \mathsf{ek}_j), \pi_j^{(gen)}\Big\}_{j\in\overline{T}}, \Big\{c_j, \pi_j^{(x)}\Big\}_{j\in\overline{T}}, c, \varphi, \hat{c}, \Big\{\mu_j, \pi_j^{(dec)}\Big\}_{j\in\overline{T}}\Big)$$

We simply prove that for all $\{x_i, \rho_i^{(gen)}, \rho_i^{(dec)}, (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i), \pi_i^{(gen)}, c_i, \pi_i^{(x)}, \mu_i, \pi_i^{(dec)}\}_{i\in T}$ output by $\mathcal{A}$, the joint distribution of these variables in the simulation is computationally indistinguishable from the joint distribution of the same variables in a real-world execution of protocol Π^{MAL}. We let View_i be the joint distribution of these variables in hybrid i. In all hybrids, $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa, N)$, $(\mathsf{crs}^{(x)}, \mathsf{tk}^{(x)}, \mathsf{xtk}) \leftarrow \mathsf{Setup}^{\mathsf{X}}(1^\kappa)$, $(\mathsf{crs}^{(dec)}, \mathsf{tk}^{(dec)}) \leftarrow \mathsf{Setup}^{\mathsf{Gen}}(1^\kappa)$, $(\mathsf{crs}^{(gen)}, \mathsf{tk}^{(gen)}) \leftarrow \mathsf{Setup}^{\mathsf{Dec}}(1^\kappa)$, $(\mathsf{ck}, \mathsf{xtk}^{(com)}) \leftarrow \mathsf{Keygen}^{\mathsf{Com}}(1^\kappa)$, $\mathsf{pk} \leftarrow \mathsf{CombinePK}(\{\mathsf{pk}_i\}_{i\in[N]})$. Also, we can assume $c = \mathsf{Eval}(C, c_1, \ldots, c_N)$, even when the server is corrupted, as this is guaranteed by soundness of the short argument system Φ. We gradually change how each of the other variables is computed.

Hybrid 0: This is the real-world execution of Π^{MAL}. We have:

$$\rho_j^{(gen)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_j^{(gen)}; s_j^{(gen)}) \quad , \quad \rho_j^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_j^{(dec)}; s_j^{(dec)})$$
$$\Big\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j; r_j^{(gen)})\Big\}_{j\in\overline{T}}$$
$$\Big\{\pi_j^{(gen)} \leftarrow \mathsf{Prove}^{\mathsf{Gen}}((\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j^{(gen)}, j), (r_j^{(gen)}, s_j^{(dec)}))\Big\}_{j\in\overline{T}}$$
$$(\cdot, \mathsf{ek}) := \mathsf{CombineKeys}\Big(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}\Big) \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}}$$
$$\Big\{\pi_j^{(x)} \leftarrow \mathsf{Prove}^{\mathsf{X}}((c_j, \mathsf{pk}), (x_j, r_j^{(x)}))\Big\}_{j\in\overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{ReRand}(c) \quad , \quad \{\mu_j \leftarrow \mathsf{ShareDec}_{\mathsf{sk}_j}(\hat{c})\}_{j\in\overline{T}}$$
$$\Big\{\pi_j^{(x)} \leftarrow \mathsf{Prove}^{\mathsf{Dec}}((\hat{c}, \mu_j, \mathsf{pk}_j, \rho_j^{(gen)}, \rho_j^{(dec)}, \mathsf{ck}, j), (r_j^{(dec)}, s_j^{(dec)}, r_j^{(gen)}, s_j^{(gen)}))\Big\}_{j\in\overline{T}}$$

Hybrid 1: We now change the proofs $\pi_j^{(x)}$ so that they are simulated proofs, created with the $\mathsf{Sim}^{\mathsf{X}}$ algorithm, instead of created honestly using $\mathsf{Prove}^{\mathsf{X}}$.

$$\rho_j^{(gen)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_j^{(gen)}; s_j^{(gen)}) \quad , \quad \rho_j^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_j^{(dec)}; s_j^{(dec)})$$
$$\Big\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j; r_j^{(gen)})\Big\}_{j\in\overline{T}}$$
$$\Big\{\pi_j^{(gen)} \leftarrow \mathsf{Prove}^{\mathsf{Gen}}((\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j^{(gen)}, j), (r_j^{(gen)}, s_j^{(dec)}))\Big\}_{j\in\overline{T}}$$
$$(\cdot, \mathsf{ek}) := \mathsf{CombineKeys}\Big(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}\Big) \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}}$$
$$\Big\{\pi_j^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{X}}((c_j, \mathsf{pk}))\Big\}_{j\in\overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{ReRand}(c) \quad , \quad \{\mu_j \leftarrow \mathsf{ShareDec}_{\mathsf{sk}_j}(\hat{c})\}_{j\in\overline{T}}$$
$$\Big\{\pi_j^{(x)} \leftarrow \mathsf{Prove}^{\mathsf{Dec}}((\hat{c}, \mu_j, \mathsf{pk}_j, \rho_j^{(gen)}, \rho_j^{(dec)}, \mathsf{ck}, j), (r_j^{(dec)}, s_j^{(dec)}, r_j^{(gen)}, s_j^{(gen)}))\Big\}_{j\in\overline{T}}$$

We have that $\mathsf{View}_0 \overset{c}{\approx} \mathsf{View}_1$ by the *zero-knowledge* property of $\Pi^{(x)}$.

Simulator $\mathcal{S}^{\text{MAL}}$

1. Run $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa, N)$, $(\mathsf{crs}^{(x)}, \mathsf{tk}^{(x)}, \mathsf{xtk}) \leftarrow \mathsf{Setup}^{\mathsf{X}}(1^\kappa)$, $(\mathsf{crs}^{(dec)}, \mathsf{tk}^{(dec)}) \leftarrow \mathsf{Setup}^{\mathsf{Gen}}(1^\kappa)$, $(\mathsf{crs}^{(gen)}, \mathsf{tk}^{(gen)}) \leftarrow \mathsf{Setup}^{\mathsf{Dec}}(1^\kappa)$, and $(\mathsf{ck}, \mathsf{xtk}^{(com)}) \leftarrow \mathsf{Keygen}^{\mathsf{Com}}(1^\kappa)$. Let $\mathsf{params}' := (\mathsf{params}, \mathsf{crs}^{(x)}, \mathsf{crs}^{(dec)}, \mathsf{crs}^{(gen)}, \mathsf{ck})$ and run $\mathcal{A}$ on inputs $\{x_i\}_{i\in T}$ and params' while playing the role of the honest parties:

   Phase 1: For all $j \in \overline{T}$, compute $\rho_j{}^{(gen)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0)$ and $\rho_j{}^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0)$. Send $(\mathsf{sid}, j, \rho_j{}^{(gen)}, \rho_j{}^{(dec)})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$.

   Phase 2:

   - Upon receiving $(\mathsf{sid}, i, \rho_i{}^{(gen)}, \rho_i{}^{(dec)})$ as a message to $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ for all $i \in T$, use the commitment trapdoor to extract randomness $r_i{}^{(gen)}, r_i{}^{(dec)}$:

$$r_i{}^{(gen)} := \mathsf{Ext}^{\mathsf{Com}}_{\mathsf{xtk}^{(com)}}(\rho_i{}^{(gen)}) \quad , \quad r_i{}^{(dec)} := \mathsf{Ext}^{\mathsf{Com}}_{\mathsf{xtk}^{(com)}}(\rho_i{}^{(dec)})$$

   - For all $i \in T$, let $(\widetilde{\mathsf{pk}_i}, \widetilde{\mathsf{sk}_i}, \widetilde{\mathsf{ek}_i}) \leftarrow \mathsf{Keygen}(1^\kappa, i; r_i{}^{(gen)})$.
   - For all $j \in \overline{T}$, compute $\pi_j{}^{(gen)} \leftarrow \mathsf{Sim}^{\mathsf{Gen}}_{\mathsf{tk}^{(dec)}}(\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j{}^{(gen)}, j)$ and send $(\mathsf{sid}, j, (\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j{}^{(gen)}, j), \pi_j{}^{(gen)})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$.
   - Upon receiving $(\mathsf{sid}, i, (\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{ck}, \rho_i{}^{(gen)}, i), \pi_i{}^{(gen)})$ as a message to $\mathcal{F}_{\text{BC}}$ for all $i \in T$, verify that $\pi_i{}^{(gen)}$ is valid. Upon receiving $(\mathsf{sid}, i, (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i))$ as a message to $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$ for all $i \in T$, check that $(\widetilde{\mathsf{pk}_i}, \widetilde{\mathsf{sk}_i}, \widetilde{\mathsf{ek}_i}) = (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ and halt if the key tuples are not the same.
   - Let $\mathsf{pk} := \mathsf{CombinePK}(\{\mathsf{pk}_i\}_{[N]})$, and run $(\mathsf{ek}, \{(\mathsf{pk}_j, \mathsf{ek}_j)\}_{j\in\overline{T}}) \leftarrow \mathsf{SimCombKeys}\left(\left\{(\widetilde{\mathsf{pk}_i}, \widetilde{\mathsf{sk}_i}, \widetilde{\mathsf{ek}_i})\right\}_{i\in T}\right)$. Send $(\mathsf{sid}, \mathsf{ek}, \{(\mathsf{pk}_i, \mathsf{ek}_i)\}_{i\in[N]})$ to $\mathcal{A}$ on behalf of $\mathcal{F}^{\mathcal{E}}_{\text{KEYS}}$.

   Phase 3:

   - For $j \in \overline{T}$, compute $c_j \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$ and $\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{X}}_{\mathsf{tk}^{(x)}}(c_j, \mathsf{pk})$. Send $(\mathsf{sid}, j, (c_j, \mathsf{pk}), \pi_j{}^{(x)})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$.
   - Upon receiving $(\mathsf{sid}, i, (c_i, \mathsf{pk}), \pi_i{}^{(x)})$ as a message to $\mathcal{F}_{\text{BC}}$ for all $i \in T$, verify that $\pi_i{}^{(x)}$ is valid, and if so, use the extraction trapdoor to extract the witness $\widetilde{x_i}$:

$$\widetilde{x_i} := \mathsf{Ext}^{\mathsf{X}}_{\mathsf{xtk}^{(x)}}((c_i, \mathsf{pk}), \pi_i{}^{(x)})$$

     Send $\{\widetilde{x_i}\}_{i\in T}$ to $\mathcal{F}$ and receive $\widetilde{y}$.

   Phase 4: If server is *not* corrupted, compute $c := \mathsf{Eval}(C, c_1, \ldots, c_N)$ and $\varphi \leftarrow \mathsf{Prove}(c, c_1, \ldots, c_N)$ honestly, and send $(\mathsf{sid}, 0, \varphi)$ to $\mathcal{F}_{\text{BC}}$. Otherwise, receive $(\mathsf{sid}, 0, c)$ from $\mathcal{A}$ as a message to $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$, and $(\mathsf{sid}, 0, \varphi)$ as a message to $\mathcal{F}_{\text{BC}}$. In the latter case, verify that φ is valid. In either case, compute $\hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(y, 1)$ and send $(\mathsf{sid}, 0, c, \hat{c})$ on behalf of $\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}$.

   Phase 5:

   - For $i \in T$, compute $\widetilde{\mu_i} := \mathsf{ShareDec}_{\mathsf{sk}_i}(\hat{c}\,; r_i{}^{(dec)})$.
   - Compute $\{\mu_j\}_{j\in\overline{T}} \leftarrow \mathsf{SimShareDec}_{\mathsf{sk}_i}(\hat{c}, \widetilde{y}, \{\widetilde{\mu_i}\}_{i\in T})$, and for $j \in \overline{T}$, send (sid, j, μ_j) on behalf of $\mathcal{F}_{\text{BC}}$.
   - Upon receiving (sid, i, μ_i) from $\mathcal{A}$ as a message to $\mathcal{F}_{\text{BC}}$ for all $i \in T$, verify that $\widetilde{\mu_i} = \mu_i$, and halt if they are not the same.

2. Output $\mathcal{A}$'s output.

Figure 15: Simulator $\mathcal{S}^{\text{MAL}}$ for (static) malicious adversary $\mathcal{A}$ corrupting clients $T \subsetneq [N]$ and possibly server S.

Hybrid 2: We change how $\hat{c}$ is created. We let

$$\widetilde{x_i} \leftarrow \mathsf{Dec}_{\mathsf{sk}}(c_i) \quad \text{for } i \in T \quad \text{and} \quad \widetilde{x_j} := x_j \quad \text{for } j \in \overline{T}$$

Instead of using the ReRand algorithm, we simply sample a fresh encryption of $\widetilde{y} = C(\widetilde{x}_1, \ldots, \widetilde{x}_N)$ with mode = 1:

$$\rho_j{}^{(gen)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_j{}^{(gen)}\ ; s_j{}^{(gen)}) \quad , \quad \rho_j{}^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_j{}^{(dec)}\ ; s_j{}^{(dec)})$$

$$\left\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j\ ; r_j{}^{(gen)})\right\}_{j \in \overline{T}}$$

$$\left\{\pi_j{}^{(gen)} \leftarrow \mathsf{Prove}^{\mathsf{Gen}}((\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j{}^{(gen)}, j)\ , (r_j{}^{(gen)}, s_j{}^{(dec)}))\right\}_{j \in \overline{T}}$$

$$(\mathsf{sk}, \mathsf{ek}) := \mathsf{CombineKeys}\left(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i \in [N]}\right) \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j \in \overline{T}}$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{X}}(\ (c_j, \mathsf{pk})\ )\right\}_{j \in \overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\widetilde{y}, 1) \quad , \quad \left\{\mu_j \leftarrow \mathsf{ShareDec}_{\mathsf{sk}_j}(\hat{c})\right\}_{j \in \overline{T}}$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Prove}^{\mathsf{Dec}}(\ (\hat{c}, \mu_j, \mathsf{pk}_j, \rho_j{}^{(gen)}, \rho_j{}^{(dec)}, \mathsf{ck}, j)\ , (r_j{}^{(dec)}, s_j{}^{(dec)}, r_j{}^{(gen)}, s_j{}^{(gen)})\ )\right\}_{j \in \overline{T}}$$

We claim that $\mathsf{View}_1 \overset{s}{\approx} \mathsf{View}_2$ by *ciphertext re-randomization* of $\mathcal{E}$. The reduction computes $\{\rho_j{}^{(gen)}, \rho_j{}^{(dec)}\}_{j \in \overline{T}}$ honestly and receives $\{\rho_i{}^{(gen)}, \rho_i{}^{(dec)}\}_{i \in T}$. It also computes $\{(\mathsf{pk}_j, \mathsf{sk}_j, \mathsf{ek}_j), \pi_j{}^{(gen)}\}_{j \in \overline{T}}$ honestly, receives $\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i), \pi_i{}^{(gen)}\}_{i \in T}$ from $\mathcal{A}$, and verifies that all $\pi_i{}^{(gen)}$ are valid. It computes $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek})$ using $\mathsf{CombinePK}$ and $\mathsf{CombineKeys}$ and computes $\{c_j \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_j)\ ,\ \pi_j{}^{(x)}\}_{j \in \overline{T}}$ honestly. It receives $\{c_i, \pi_i{}^{(x)}\}_{i \in T}$ from $\mathcal{A}$ and verifies that all $\pi_i{}^{(x)}$ are valid. If the server is honest, it computes $c := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_N)$ and φ honestly; otherwise it receives c, φ from $\mathcal{A}$, in which case it verifies φ. It computes $\{\widetilde{x}_i \leftarrow \mathsf{Dec}_{\mathsf{sk}}(c_i)\}_{i \in T}$ and $\widetilde{y} = C(\widetilde{x}_1, \ldots, \widetilde{x}_N)$. Given $\hat{c}$ which is either $\hat{c} = \mathsf{ReRand}(c)$ or $\hat{c} = \mathsf{Enc}_{\mathsf{pk}}(y, 1)$, it computes $\{\mu_j \leftarrow \mathsf{ShareDec}_{\mathsf{sk}_j}(\hat{c}), \pi_j{}^{(dec)}\}_{j \in \overline{T}}$ honestly.

If all proofs $\pi_i{}^{(gen)}$ verify, by soundness of $\Pi^{(gen)}$ we know that $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ is a valid key tuple for all $i \in T$. Likewise, by soundness of $\Pi^{(x)}$ we know that for all $i \in T$ there exists $x_i, r_i{}^{(x)}$ such that $c_i = \mathsf{Enc}_{\mathsf{pk}}(x_i, 0; r_i{}^{(x)})$, and by construction we have $\widetilde{x}_i = x_i$ (except with negligible probability). Therefore, $\mathsf{View}_1 \overset{s}{\approx} \mathsf{View}_2$ by *ciphertext re-randomization* of $\mathcal{E}$.

Hybrid 3: We now change how the $\widetilde{x}_i$ are created. Instead of decrypting the ciphertext c_i, we obtain $\widetilde{x}_i$ by using the proof of knowledge extractor of $\Pi^{(x)}$.

$$\widetilde{x}_i \leftarrow \mathsf{Ext}^{\mathsf{X}}(\pi_i{}^{(x)}) \quad \text{for } i \in T \quad \text{and} \quad \widetilde{x}_j := x_j \quad \text{for } j \in \overline{T}$$

We claim that $\mathsf{View}_2 \overset{c}{\approx} \mathsf{View}_3$ by the *simulation-extractability* property of $\Pi^{(x)}$. Note that we must rely on the stronger property of simulation-extractability instead of standard proof of knowledge since the adversary sees the simulated proofs $\pi_j{}^{(x)}$ *before* producing the proofs $\pi_i{}^{(x)}$. We therefore need to be able to extract the witness $\widetilde{x}_i$ from the proof $\pi_i{}^{(x)}$ even if the adversary sees simulated proofs before creating $\pi_i{}^{(x)}$. This is precisely what simulation-extractability guarantees.

Hybrid 4: We now change the proofs $\pi_j{}^{(dec)}$ so that they are simulated proofs, created with the $\mathsf{Sim}^{\mathsf{Dec}}$ algorithm, instead of created honestly using $\mathsf{Prove}^{\mathsf{Dec}}$.

$$\rho_j{}^{(gen)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_j{}^{(gen)}\ ; s_j{}^{(gen)}) \quad , \quad \rho_j{}^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_j{}^{(dec)}\ ; s_j{}^{(dec)})$$

$$\left\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j\ ; r_j{}^{(gen)})\right\}_{j \in \overline{T}}$$

$$\left\{\pi_j{}^{(gen)} \leftarrow \mathsf{Prove}^{\mathsf{Gen}}((\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j{}^{(gen)}, j)\ , (r_j{}^{(gen)}, s_j{}^{(dec)}))\right\}_{j\in\overline{T}}$$

$$(\cdot, \mathsf{ek}) := \mathsf{CombineKeys}\left(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}\right) \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}}$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{X}}(\ (c_j, \mathsf{pk})\ )\right\}_{j\in\overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\widetilde{y}, 1) \quad , \quad \{\mu_j \leftarrow \mathsf{ShareDec}_{\mathsf{sk}_j}(\hat{c})\}_{j\in\overline{T}}$$

$$\left\{\pi_j{}^{(dec)} \leftarrow \mathsf{Sim}^{\mathsf{Dec}}(\ (\hat{c}, \mu_j, \mathsf{pk}_j, \rho_j{}^{(gen)}, \rho_j{}^{(dec)}, \mathsf{ck}, j)\ )\right\}_{j\in\overline{T}}$$

We have that $\mathsf{View}_3 \overset{c}{\approx} \mathsf{View}_4$ by the *zero-knowledge* property of $\Pi^{(dec)}$.

Hybrid 5: We now change the commitment $\rho_j{}^{(dec)}$ so that it is a commitment of 0, instead of the real randomness used in decryption.

$$\rho_j{}^{(gen)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_j{}^{(gen)}\ ; s_j{}^{(gen)}) \quad , \quad \rho_j{}^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0)$$

$$\left\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j\ ; r_j{}^{(gen)})\right\}_{j\in\overline{T}}$$

$$\left\{\pi_j{}^{(gen)} \leftarrow \mathsf{Prove}^{\mathsf{Gen}}((\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j{}^{(gen)}, j)\ , (r_j{}^{(gen)}, s_j{}^{(dec)}))\right\}_{j\in\overline{T}}$$

$$(\cdot, \mathsf{ek}) := \mathsf{CombineKeys}\left(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}\right) \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}}$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{X}}(\ (c_j, \mathsf{pk})\ )\right\}_{j\in\overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\widetilde{y}, 1) \quad , \quad \{\mu_j \leftarrow \mathsf{ShareDec}_{\mathsf{sk}_j}(\hat{c})\}_{j\in\overline{T}}$$

$$\left\{\pi_j{}^{(dec)} \leftarrow \mathsf{Sim}^{\mathsf{Dec}}(\ (\hat{c}, \mu_j, \mathsf{pk}_j, \rho_j{}^{(gen)}, \rho_j{}^{(dec)}, \mathsf{ck}, j)\ )\right\}_{j\in\overline{T}}$$

We have that $\mathsf{View}_4 \overset{c}{\approx} \mathsf{View}_5$ by the *hiding* property of the trapdoor commitment scheme Com.

Hybrid 6: In this hybrid, for $i \in T$ we let

$$r_i{}^{(dec)} := \mathsf{Ext}^{\mathsf{Com}}(\rho_i{}^{(dec)}) \quad \text{and} \quad \widetilde{\mu}_i := \mathsf{ShareDec}_{\mathsf{sk}_i}(\hat{c}\ ; r_i{}^{(dec)})$$

and halt if $\widetilde{\mu}_i \neq \mu_i$, where $\{\mu_i\}_{i\in T}$ are the shares sent by $\mathcal{A}$. We claim that $\mathsf{View}_5 \overset{c}{\approx} \mathsf{View}_6$. This is guaranteed by the *soundness* of $\Pi^{(dec)}$ and the *extraction* property of the trapdoor commitment.

Hybrid 7: We change the way the decryption shares $\{\mu_j\}_{j\in\overline{T}}$ are generated. Instead of using the $\mathsf{ShareDec}$ algorithm with secret key sk_j, we now use the $\mathsf{SimShareDec}$ algorithm with $\widetilde{y} = C(\widetilde{x}_1, \ldots, \widetilde{x}_N)$ and $\{\widetilde{\mu}_i\}_{i\in T}$:

$$\rho_j{}^{(gen)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_j{}^{(gen)}\ ; s_j{}^{(gen)}) \quad , \quad \rho_j{}^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0)$$

$$\left\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j\ ; r_j{}^{(gen)})\right\}_{j\in\overline{T}}$$

$$\left\{\pi_j{}^{(gen)} \leftarrow \mathsf{Prove}^{\mathsf{Gen}}((\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j{}^{(gen)}, j)\ , (r_j{}^{(gen)}, s_j{}^{(dec)}))\right\}_{j\in\overline{T}}$$

$$(\cdot, \mathsf{ek}) := \mathsf{CombineKeys}\left(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}\right) \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}}$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{X}}(\ (c_j, \mathsf{pk})\ )\right\}_{j\in\overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\widetilde{y}, 1) \quad , \quad \{\mu_j\}_{j\in\overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, \widetilde{y}, \{\widetilde{\mu}_i\}_{i\in T})$$

$$\left\{\pi_j{}^{(dec)} \leftarrow \mathsf{Sim}^{\mathsf{Dec}}(\ (\hat{c}, \mu_j, \mathsf{pk}_j, \rho_j{}^{(gen)}, \rho_j{}^{(dec)}, \mathsf{ck}, j)\ )\right\}_{j\in\overline{T}}$$

We claim that $\mathsf{View}_6 \overset{c}{\approx} \mathsf{View}_7$ by *share-simulation indistinguishability* of $\mathcal{E}$. The reduction computes $\left\{\rho_j{}^{(gen)}, \rho_j{}^{(dec)}\right\}_{j\in\overline{T}}$ and receives $\left\{\rho_i{}^{(gen)}, \rho_i{}^{(dec)}\right\}_{i\in T}$. It also computes $\left\{(\mathsf{pk}_j, \mathsf{sk}_j, \mathsf{ek}_j), \pi_j{}^{(gen)}\right\}_{j\in\overline{T}}$ honestly, receives $\left\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i), \pi_i{}^{(gen)}\right\}_{i\in T}$ from $\mathcal{A}$, and verifies that all $\pi_i{}^{(gen)}$ are valid. It computes $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek})$ using $\mathsf{CombinePK}$ and $\mathsf{CombineKeys}$ and computes $\left\{c_j \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_j)\ ,\ \pi_j{}^{(x)}\right\}_{j\in\overline{T}}$ with a simulated proof. It receives $\left\{c_i, \pi_i{}^{(x)}\right\}_{i\in T}$ from $\mathcal{A}$ and verifies that all $\pi_i{}^{(x)}$ are valid. If the server is honest, it computes $c := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_N)$ and φ honestly; otherwise it receives c, φ from $\mathcal{A}$, in which case it verifies φ. It computes $\left\{\widetilde{x}_i \leftarrow \mathsf{Ext}^{\mathsf{X}}(\pi_i{}^{(x)})\right\}_{i\in T}$ and $\widetilde{y} = C(\widetilde{x}_1, \ldots, \widetilde{x}_N)$ and $\hat{c} \leftarrow \mathsf{Enc}(\widetilde{y}, 1)$. It then computes $r_i{}^{(dec)} \leftarrow \mathsf{Ext}^{\mathsf{Com}}(\rho_i{}^{(dec)})$ and $\widetilde{\mu}_i := \mathsf{ShareDec}_{\mathsf{sk}_i}(\hat{c}\ ; r_i{}^{(dec)})$, $\{\mu_j\}_{j\in\overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, \widetilde{y}, \{\widetilde{\mu}_i\}_{i\in T})$ and simulated proofs $\left\{\pi_j{}^{(dec)}\right\}_{j\in\overline{T}}$. It receives $\left\{\mu_i, \pi_i{}^{(dec)}\right\}_{i\in T}$ from $\mathcal{A}$, verifies all proofs $\pi_i{}^{(dec)}$, and halts if $\widetilde{\mu}_i \neq \mu_i$ for any $i \in T$.

If all proofs $\pi_i{}^{(gen)}$ verify, by soundness of $\Pi^{(gen)}$ we know that $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ is a valid key tuple for all $i \in T$. Likewise, by *simulation-soundness* of $\Pi^{(dec)}$ we know that for all $i \in T$ there exists $r_i{}^{(dec)}$ such that $\mu_i := \mathsf{ShareDec}_{\mathsf{sk}_i}(\hat{c}\ ; r_i{}^{(dec)})$. By construction, $\hat{c} \leftarrow \mathsf{Enc}(\widetilde{y})$. Thus, $\mathsf{View}_5 \overset{c}{\approx} \mathsf{View}_6$ by *share-simulation indistinguishability* of $\mathcal{E}$.

Hybrid 8: We now change the proofs $\pi_j{}^{(gen)}$ so that they are simulated proofs, created with the $\mathsf{Sim}^{\mathsf{Gen}}$ algorithm, instead of created honestly using $\mathsf{Prove}^{\mathsf{Gen}}$.

$$\rho_j{}^{(gen)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(r_j{}^{(gen)}\ ; s_j{}^{(gen)}) \quad , \quad \rho_j{}^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0)$$

$$\left\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j\ ; r_j{}^{(gen)})\right\}_{j\in\overline{T}} \quad , \quad \left\{\pi_j{}^{(gen)} \leftarrow \mathsf{Sim}^{\mathsf{Gen}}(\ (\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j{}^{(gen)}, j)\ )\right\}_{j\in\overline{T}}$$

$$(\mathsf{pk}, \cdot, \mathsf{ek}) := \mathsf{CombineKeys}\left(\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}\right) \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}}$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{X}}(\ (c_j, \mathsf{pk})\ )\right\}_{j\in\overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\widetilde{y}, 1) \quad , \quad \{\mu_j\}_{j\in\overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, y, \{\widetilde{\mu}_i\}_{i\in T})$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{Dec}}(\ (\hat{c}, \mu_j, \mathsf{pk}_j, \rho_j{}^{(gen)}, \rho_j{}^{(dec)}, \mathsf{ck}, j)\ )\right\}_{j\in\overline{T}}$$

We have that $\mathsf{View}_7 \overset{c}{\approx} \mathsf{View}_8$ by the *zero-knowledge* property of $\Pi^{(gen)}$.

Hybrid 9: We change the commitment $\rho_j{}^{(gen)}$ so that it is a commitment of 0, instead of the real randomness used in key generation.

$$\rho_j{}^{(gen)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0) \quad , \quad \rho_j{}^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0)$$

$$\left\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j\ ; r_j{}^{(gen)})\right\}_{j\in\overline{T}} \quad , \quad \left\{\pi_j{}^{(gen)} \leftarrow \mathsf{Sim}^{\mathsf{Gen}}(\ (\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j{}^{(gen)}, j)\ )\right\}_{j\in\overline{T}}$$

$$(\cdot, \mathsf{ek}) := \mathsf{CombineKeys}\left(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i\in[N]}\right) \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j\in\overline{T}}$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{X}}(\ (c_j, \mathsf{pk})\ )\right\}_{j\in\overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\widetilde{y}, 1) \quad , \quad \{\mu_j\}_{j\in\overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, y, \{\widetilde{\mu}_i\}_{i\in T})$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{Dec}}(\ (\hat{c}, \mu_j, \mathsf{pk}_j, \rho_j{}^{(gen)}, \rho_j{}^{(dec)}, \mathsf{ck}, j)\ )\right\}_{j\in\overline{T}}$$

We have that $\mathsf{View}_8 \overset{c}{\approx} \mathsf{View}_9$ by the *hiding* property of the trapdoor commitment scheme Com.

Hybrid 10: In this hybrid, for $i \in T$ we let

$$r_i{}^{(gen)} := \mathsf{Ext}^{\mathsf{Com}}(\rho_i{}^{(gen)}) \quad \text{and} \quad (\widetilde{\mathsf{pk}_i}, \cdot, \cdot) := \mathsf{Keygen}(1^\kappa, i\ ; r_i{}^{(gen)})$$

and halt if $\widetilde{\mathsf{pk}_i} \neq \mathsf{pk}_i$, where $\{(\mathsf{pk}_i, \cdot, \cdot)\}_{i \in T}$ are the public keys sent by $\mathcal{A}$. We claim that $\mathsf{View}_9 \overset{c}{\approx} \mathsf{View}_{10}$. This is guaranteed by the *simulation-soundness* of $\Pi^{(gen)}$ and the *extraction* property of the trapdoor commitment.

Hybrid 11: This is actually a series of $N - t$ hybrids. We change how the ciphertexts $\{c_j\}_{j \in \overline{T}}$ are created, one at a time. Instead of having c_{j^*} be an encryption of input x_{j^*}, we have it be an encryption of 0.

$$\rho_j{}^{(gen)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0) \quad , \quad \rho_j{}^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0)$$

$$\left\{(\mathsf{pk}_j, \cdot, \mathsf{ek}_j) \leftarrow \mathsf{Keygen}(1^\kappa, j\ ; r_j{}^{(gen)})\right\}_{j \in \overline{T}} \quad , \quad \left\{\pi_j{}^{(gen)} \leftarrow \mathsf{Sim}^{\mathsf{Gen}}(\ (\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j{}^{(gen)}, j)\ )\right\}_{j \in \overline{T}}$$

$$(\cdot, \mathsf{ek}) := \mathsf{CombineKeys}\left(\{(\mathsf{sk}_i, \mathsf{ek}_i)\}_{i \in [N]}\right) \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(0)\}_{j \in \overline{T}, j \leq j^*} \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(x_j)\}_{j \in \overline{T}, j > j^*}$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{X}}(\ (c_j, \mathsf{pk})\ )\right\}_{j \in \overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\widetilde{y}, 1) \quad , \quad \{\mu_j\}_{j \in \overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, \widetilde{y}, \{\widetilde{\mu}_i\}_{i \in T})$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{Dec}}(\ (\hat{c}, \mu_j, \mathsf{pk}_j, \rho_j{}^{(gen)}, \rho_j{}^{(dec)}, \mathsf{ck}, j)\ )\right\}_{j \in \overline{T}}$$

We claim two consecutive hybrids are computationally indistinguishable by *semantic security* of $\mathcal{E}$. The reduction computes $\left\{\rho_j{}^{(gen)}, \rho_j{}^{(dec)}\right\}_{j \in \overline{T}}$ and receives $\left\{\rho_i{}^{(gen)}, \rho_i{}^{(dec)}\right\}_{i \in T}$. It computes $r_i{}^{(gen)} := \mathsf{Ext}^{\mathsf{Com}}(\rho_i{}^{(gen)})$ and sends $\left\{(\widetilde{\mathsf{pk}_i}, \widetilde{\mathsf{sk}_i}, \widetilde{\mathsf{ek}_i}) := \mathsf{Keygen}(1^\kappa, i\ ; r_i{}^{(gen)})\right\}_{i \in T}$ to the challenger. It forwards $(\mathsf{ek}, \left\{(\mathsf{pk}_j, \mathsf{ek}_j)\right\}_{j \in \overline{T}}$ from the challenger to $\mathcal{A}$ and sends to $\mathcal{A}$ simulated proofs $\pi_j{}^{(gen)}$. Upon receiving $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i), \pi_i{}^{(gen)}$ from $\mathcal{A}$, it verifies $\pi_i{}^{(gen)}$ and checks that $\widetilde{\mathsf{pk}_i} = \mathsf{pk}_i$ and halts otherwise. It uses the challenge ciphertext as c_{j^*} and computes the other ciphertexts c_j accordingly, with simulated proofs $\pi_i{}^{(x)}$ for all $j \in \overline{T}$. When it receives $c_i, \pi_i{}^{(x)}$, it verifies $\pi_i{}^{(x)}$. If the server is honest, it computes $c := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_N)$ and φ honestly; otherwise it receives c, φ from $\mathcal{A}$, in which case it verifies φ. It computes $r_i{}^{(dec)} \leftarrow \mathsf{Ext}^{\mathsf{Com}}(\rho_i{}^{(dec)})$ and $\widetilde{\mu}_i := \mathsf{ShareDec}_{\mathsf{sk}_i}(\hat{c}\ ; r_i{}^{(dec)})$, $\{\mu_j\}_{j \in \overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, \widetilde{y}, \{\widetilde{\mu}_i\}_{i \in T})$ and simulated proofs $\left\{\pi_j{}^{(dec)}\right\}_{j \in \overline{T}}$. It receives $\left\{\mu_i, \pi_i{}^{(dec)}\right\}_{i \in T}$ from $\mathcal{A}$, verifies all proofs $\pi_i{}^{(dec)}$, and halts if $\widetilde{\mu}_i \neq \mu_i$ for any $i \in T$. We define View_{11} to be the view having all c_j encrypting 0. Putting these $N - t$ hybrids together, we conclude that $\mathsf{View}_{10} \overset{c}{\approx} \mathsf{View}_{11}$.

Hybrid 12: We change how the public/evaluation key pairs $\left\{(\mathsf{pk}_j, \mathsf{ek}_j)\right\}_{j \in \overline{T}}$ and the combined evaluation key ek are generated. Instead of creating them using Keygen and $\mathsf{CombineKeys}$, we use the $\mathsf{SimCombKeys}$ algorithm.

$$\rho_j{}^{(gen)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0) \quad , \quad \rho_j{}^{(dec)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0)$$

$$\left(\mathsf{ek}, \left\{(\mathsf{pk}_j, \mathsf{ek}_j)\right\}_{j \in \overline{T}}\right) \leftarrow \mathsf{SimCombKeys}\left(\{\mathsf{pk}_i, \mathsf{ek}_i\}_{i \in T}\right)$$

$$\left\{\pi_j{}^{(gen)} \leftarrow \mathsf{Sim}^{\mathsf{Gen}}(\ (\mathsf{pk}_j, \mathsf{ek}_j, \mathsf{ck}, \rho_j{}^{(gen)}, j)\ )\right\}_{j \in \overline{T}} \quad , \quad \{c_j = \mathsf{Enc}_{\mathsf{pk}}(0)\}_{j \in \overline{T}}$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{X}}(\ (c_j, \mathsf{pk})\ )\right\}_{j \in \overline{T}} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\widetilde{y}, 1) \quad , \quad \{\mu_j\}_{j \in \overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, y, \{\widetilde{\mu}_i\}_{i \in T})$$

$$\left\{\pi_j{}^{(x)} \leftarrow \mathsf{Sim}^{\mathsf{Dec}}(\ (\hat{c}, \mu_j, \mathsf{pk}_j, \rho_j{}^{(gen)}, \rho_j{}^{(dec)}, \mathsf{ck}, j)\ )\right\}_{j\in\overline{T}}$$

We claim that $\mathsf{View}_{11} \overset{c}{\approx} \mathsf{View}_{12}$ by *key-simulation indistinguishability* of $\mathcal{E}$. The reduction computes $\left\{\rho_j{}^{(gen)}, \rho_j{}^{(dec)}\right\}_{j\in\overline{T}}$ honestly and receives $\left\{\rho_i{}^{(gen)}, \rho_i{}^{(dec)}\right\}_{i\in T}$. It computes $r_i{}^{(gen)} := \mathsf{Ext}^{\mathsf{Com}}(\rho_i{}^{(gen)})$ and $\left\{(\widetilde{\mathsf{pk}_i}, \widetilde{\mathsf{sk}_i}, \widetilde{\mathsf{ek}_i}) := \mathsf{Keygen}(1^\kappa, i\ ; r_i{}^{(gen)})\right\}_{i\in T}$. Given keys $(\mathsf{ek}, \left\{(\widetilde{\mathsf{pk}_j}, \widetilde{\mathsf{ek}_j})\right\}_{j\in\overline{T}})$ which are either created honestly using Keygen and $\mathsf{CombineKeys}$, or simulated using $\mathsf{SimCombKeys}$, it forwards these to $\mathcal{A}$. It then receives $\left\{(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i), \pi_i{}^{(gen)}\right\}_{i\in T}$ from $\mathcal{A}$, and verifies $\pi_i{}^{(gen)}$. It halts if $\widetilde{\mathsf{pk}_i} \neq \mathsf{pk}_i$ for any $i \in T$; otherwise it computes $\mathsf{pk} := \mathsf{CombinePK}\left(\{\mathsf{pk}_i\}_{i\in[N]}\right)$. It also computes $\{c_j \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)\}_{j\in\overline{T}}$ with simulated proofs $\pi_j{}^{(x)}$, receives $\left\{(c_i, \pi_i{}^{(x)})\right\}_{i\in T}$ from $\mathcal{A}$ and verifies $\pi_i{}^{(x)}$. If the server is honest, it computes $c := \mathsf{Eval}_{\mathsf{ek}}(C, c_1, \ldots, c_N)$ and φ honestly; otherwise it receives c, φ from $\mathcal{A}$, in which case it verifies φ. It computes $\left\{\widetilde{x}_i \leftarrow \mathsf{Ext}^{\mathsf{X}}(\pi_i{}^{(x)})\right\}_{i\in T}$ and $\widetilde{y} = C(\widetilde{x}_1, \ldots, \widetilde{x}_N)$ and $\hat{c} \leftarrow \mathsf{Enc}(\widetilde{y}, 1)$. It then computes $r_i{}^{(dec)} \leftarrow \mathsf{Ext}^{\mathsf{Com}}(\rho_i{}^{(dec)})$ and $\widetilde{\mu}_i := \mathsf{ShareDec}_{\mathsf{sk}_i}(\hat{c}\ ; r_i{}^{(dec)})$, $\{\mu_j\}_{j\in\overline{T}} \leftarrow \mathsf{SimShareDec}(\hat{c}, \widetilde{y}, \{\widetilde{\mu}_i\}_{i\in T})$ and simulated proofs $\left\{\pi_j{}^{(dec)}\right\}_{j\in\overline{T}}$. It receives $\left\{\mu_i, \pi_i{}^{(dec)}\right\}_{i\in T}$ from $\mathcal{A}$, verifies all proofs $\pi_i{}^{(dec)}$, and halts if $\widetilde{\mu}_i \neq \mu_i$ for any $i \in T$.

If all proofs $\pi_i{}^{(gen)}$ verify, by *simulation-soundness* of $\Pi^{(gen)}$ we know that $(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{ek}_i)$ is a valid key tuple for all $i \in T$. Thus, $\mathsf{View}_{11} \overset{c}{\approx} \mathsf{View}_{12}$ by *key-simulation indistinguishability* of $\mathcal{E}$.

We have proved that $\mathsf{View}_0 \overset{c}{\approx} \mathsf{View}_{12}$. Since View_{12} is the view that $\mathcal{S}^{\text{MAL}}$ produces, and View_0 is the view produced in an execution of Π^{MAL}, we have that $\text{IDEAL}_{\mathcal{F},\mathcal{S}^{\text{MAL}}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\Pi^{\text{MAL}},\mathcal{A}}(\vec{x})$.

□

A.2 Ciphertext Re-Randomization

For security, we are not concerned with *privacy* as the parties do not have any inputs that they wish to keep private. However, we are concerned with *correctness*; we need to ensure that the output $\hat{c}$ of the protocol, is indistinguishable from a fresh encryption of the plaintext y. This is fairly straight-forward in the case of a semi-honest adversary but requires more care when dealing with a malicious adversary. In this case, we also rely on simulation-sound NIZK proofs. As before, we show the proof of security for protocol $\Pi^{\text{MAL}}_{\text{RAND}}$ after the functionality $\mathcal{F}^{R^{(gen)}}_{\text{ZK.BC}}$ has been instantiated with simulation-sound NIZK proofs.

Proof of Theorem 7.1: We describe the simulator $\mathcal{S}^{\text{SH}}_{\text{RAND}}$ for a (static) semi-honest adversary $\mathcal{A}$ corrupting parties in $T \subsetneq [N]$ (and possibly server S) in Figure 17. We prove that $\text{IDEAL}_{\Pi^{\text{SH}}_{\text{RAND}},\mathcal{S}^{\text{SH}}_{\text{RAND}}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}},\mathcal{A}}(\vec{x})$ using a series of hybrids. The view of $\mathcal{A}$ consists of

$$\mathsf{View} = (\mathsf{pk}, c, z_1, \ldots, z_N, \hat{c})$$

We simply prove that the joint distribution of these variables in the simulation (where the ciphertext output is a fresh encryption of y) is computationally indistinguishable from the joint distribution of the same variables in a real-world execution of protocol $\Pi^{\text{SH}}_{\text{RAND}}$. This is easy to see because z_{j^*} is a deterministic function of $\hat{c}$ and $\{z_j\}_{i\in N, i\neq j^*}$, all z_i's for $i \neq j^*$ are distributed the same in both cases (since the adversary is semi-honest), and the distribution of $\hat{c}$ is statistically close to $\mathsf{Enc}_{\mathsf{pk}}(y, 1)$ since the noise in each z_i is a super-polynomial factor larger than the noise in c. □

Proof of Theorem 7.2: We describe the simulator $\mathcal{S}^{\text{MAL}}_{\text{RAND}}$ for a (static) semi-honest adversary $\mathcal{A}$ corrupting parties in $T \subsetneq [N]$ (and possibly server S) in Figure 17. We prove that $\text{IDEAL}_{\Pi^{\text{MAL}}_{\text{RAND}},\mathcal{S}^{\text{MAL}}_{\text{RAND}}}(\vec{x}) \overset{c}{\approx}$

Simulator $\mathcal{S}^{\text{SH}}_{\text{RAND}}$

Inputs: All parties $P_1, \ldots, P_N$ and server S get public key pk as input. Server S additionally gets a ciphertext $c \in \mathcal{C}_{\mathsf{pk}}$. The simulator also receives input y, which is the plaintext of c.

- Upon receiving (sid, i, z_i) as a message to $\mathcal{F}_{\text{BC}}$ for all $i \in T$:
  - Let $\hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(y, 1)$, and let $j^* \in \overline{T}$. For all $j \in \overline{T}, j \neq j^*$, set $z_j := \mathsf{Enc}_{\mathsf{pk}}(0, 1)$. Set $z_{j^*} := \hat{c} - \sum_{i \in [N], i \neq j^*} z_i$
  - For all $j \in \overline{T}$, send (sid, j, z_j) to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$.
- Output whatever $\mathcal{A}$ outputs.

Figure 16: Simulator $\mathcal{S}^{\text{SH}}_{\text{RAND}}$ for (static) semi-honest adversary $\mathcal{A}$ corrupting parties in $T \subsetneq [N]$ in protocol $\Pi^{\text{SH}}_{\text{RAND}}$

$\text{REAL}_{\mathcal{F}^{\mathcal{E}}_{\text{RAND.BC}}, \mathcal{A}}(\vec{x})$ using a series of hybrids. The view of the $\mathcal{A}$ consists of

$$\mathsf{View} = \left(\mathsf{pk}, \mathsf{ck}, c, \left\{\rho_i{}^{(0)}, z_i, \pi_i{}^{(0)}\right\}_{i \in [N]}, \hat{c}\right)$$

Simulator $\mathcal{S}^{\text{MAL}}_{\text{RAND}}$

Inputs: All parties $P_1, \ldots, P_N$ and server S get public key pk, and a commitment key ck as input. Server S additionally gets a ciphertext $c \in \mathcal{C}_{\mathsf{pk}}$. The simulator also receives input y, which is the plaintext of c.

Round 1: For $j \in \overline{T}$, compute commitments $\rho_j{}^{(0)} \leftarrow \mathsf{Com}_{\mathsf{ck}}(0)$ and send $(\mathsf{sid}, j, \rho_j{}^{(0)})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$.

Round 2: Upon receiving $(\mathsf{sid}, i, \rho_i{}^{(0)})$ as a message to $\mathcal{F}_{\text{BC}}$ for all $i \in T$

- Let
$$r_i{}^{(0)} := \mathsf{Ext}^{\mathsf{Com}}(\rho_i{}^{(0)}) \quad \text{and} \quad \widetilde{z}_i := \mathsf{Enc}_{\mathsf{pk}}(0, 1\,; r_i{}^{(0)})$$
- Compute $\hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(y, 1)$, and let $j^* \in \overline{T}$. For all $j \in \overline{T}, j \neq j^*$, set $\widetilde{z}_j := \mathsf{Enc}_{\mathsf{pk}}(0, 1)$. Set $\widetilde{z}_{j^*} := \hat{c} - \sum_{i \in [N], i \neq j^*} \widetilde{z}_i$.
- For $j \in \overline{T}$, compute $\pi_j{}^{(0)} \leftarrow \mathsf{Sim}^{\mathsf{Zero}}(\rho_j{}^{(0)}, z_j, \mathsf{pk})$.
- For $j \in \overline{T}$, send $(\mathsf{sid}, j, \widetilde{z}_j, \pi_j{}^{(0)})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$

Round 3: Upon receiving $(\mathsf{sid}, i, z_i, \pi_i{}^{(0)})$ as a message to $\mathcal{F}_{\text{BC}}$ for all $i \in T$, verify $\pi_i{}^{(0)}$ and check that $\widetilde{z}_i = z_i$; halt if they are not equal. If the server S is honest, send $(\mathsf{sid}, 0, c, \hat{c})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$.

Output whatever $\mathcal{A}$ outputs.

Figure 17: Simulator $\mathcal{S}^{\text{MAL}}_{\text{RAND}}$ for (static) malicious adversary $\mathcal{A}$ corrupting parties in $T \subsetneq [N]$ in protocol $\Pi^{\text{MAL}}_{\text{RAND}}$

We simply prove that the joint distribution of these variables in the simulation is computationally indistinguishable from the joint distribution of the same variables in a real-world execution of protocol $\Pi^{\text{MAL}}_{\text{RAND}}$. We let View_i be the joint distribution of these variables in hybrid i.

Hybrid 0: This is the real-world execution of $\Pi^{\text{MAL}}_{\text{RAND}}$. We have

$$\left\{\rho_j{}^{(0)} := \mathsf{Com}_{\mathsf{ck}}(r_j{}^{(0)}\,; s_j{}^{(0)})\right\}_{j \in \overline{T}} \quad , \quad \left\{z_j \leftarrow \mathsf{Enc}(0, 1\,; r_j{}^{(0)})\right\}_{j \in \overline{T}}$$

$$\left\{\pi_j{}^{(0)} \leftarrow \mathsf{Prove}^{\mathsf{Zero}}((\rho_j{}^{(0)}, z_j, \mathsf{pk}), (r_j{}^{(0)}, s_j{}^{(0)}))\right\} \quad , \quad \hat{c} := c + \sum_{i=1}^{N} z_i$$

Hybrid 1: We now change the proofs $\pi_j{}^{(0)}$ so that they are simulated proofs, created with the $\mathsf{Sim}^{\mathsf{Zero}}$ algorithm, instead of created honestly using $\mathsf{Prove}^{\mathsf{Zero}}$.

$$\left\{\rho_j{}^{(0)} := \mathsf{Com}_{\mathsf{ck}}(r_j{}^{(0)}\ ; s_j{}^{(0)})\right\}_{j\in\overline{T}} \quad , \quad \left\{z_j \leftarrow \mathsf{Enc}(0, 1\ ; r_j{}^{(0)})\right\}_{j\in\overline{T}}$$

$$\left\{\pi_j{}^{(0)} \leftarrow \mathsf{Sim}^{\mathsf{Zero}}(\rho_j{}^{(0)}, z_j, \mathsf{pk})\right\} \quad , \quad \hat{c} := c + \sum_{i=1}^{N} z_i$$

$\mathsf{View}_0 \overset{c}{\approx} \mathsf{View}_1$ by the *zero-knowledge* property of $\Pi^{(zero)}$.

Hybrid 2: We change the commitments $\rho_j{}^{(0)}$ so that they are now commitments of 0 instead of the randomness $r_j{}^{(0)}$.

$$\left\{\rho_j{}^{(0)} := \mathsf{Com}_{\mathsf{ck}}(0)\right\}_{j\in\overline{T}} \quad , \quad \left\{z_j \leftarrow \mathsf{Enc}(0, 1\ ; r_j{}^{(0)})\right\}_{j\in\overline{T}}$$

$$\left\{\pi_j{}^{(0)} \leftarrow \mathsf{Sim}^{\mathsf{Zero}}(\rho_j{}^{(0)}, z_j, \mathsf{pk})\right\} \quad , \quad \hat{c} := c + \sum_{i=1}^{N} z_i$$

$\mathsf{View}_1 \overset{c}{\approx} \mathsf{View}_2$ by the *hiding* property of the trapdoor commitment scheme.

Hybrid 3: In this hybrid, for $i \in T$ we let

$$r_i{}^{(0)} := \mathsf{Ext}^{\mathsf{Com}}(\rho_i{}^{(0)}) \quad \text{and} \quad \widetilde{z}_i := \mathsf{Enc}_{\mathsf{pk}}(0, 1\ ; r_i{}^{(0)})$$

and halt if $\widetilde{z}_i \neq z_i$, where $\{z_i\}_{i\in T}$ are the ciphertexts sent by $\mathcal{A}$. We claim that $\mathsf{View}_2 \overset{c}{\approx} \mathsf{View}_3$. This is guaranteed by the *simulation-soundness* of $\Pi^{(zero)}$ and the *extraction* property of the trapdoor commitment.

Hybrid 4: Let $j^* \in \overline{T}$, and for $j \in \overline{T}$, let $\widetilde{z}_j = z_j$. We view $\widetilde{z}_{j^*}$ as a deterministic function of $\hat{c}$ and the rest of the $\widetilde{z}_i$'s: $z_{j^*} = \hat{c} - \sum_{i=1}^{N} \widetilde{z}_i$. We now change $\hat{c}$ to be a fresh encryption of y.

$$\left\{\rho_j{}^{(0)} := \mathsf{Com}_{\mathsf{ck}}(0)\right\}_{j\in\overline{T}} \quad , \quad \left\{z_j \leftarrow \mathsf{Enc}(0, 1\ ; r_j{}^{(0)})\right\}_{j\in\overline{T}, j\neq j^*} \quad , \quad z_{j^*} = \hat{c} - \sum_{i=1}^{N} \widetilde{z}_i$$

$$\left\{\pi_j{}^{(0)} \leftarrow \mathsf{Sim}^{\mathsf{Zero}}(\rho_j{}^{(0)}, z_j, \mathsf{pk})\right\} \quad , \quad \hat{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(y, 1)$$

We claim that $\mathsf{View}_3 \overset{s}{\approx} \mathsf{View}_4$. In both cases, z_{j^*} is the same deterministic function of $\hat{c}$ and the rest of the $\widetilde{z}_j$'s. The distribution of $\hat{c}$ is statistically close to $\mathsf{Enc}_{\mathsf{pk}}(y, 1)$ since the noise in each z_i is a super-polynomial factor larger than the noise in c.

We have proved that $\mathsf{View}_0 \overset{c}{\approx} \mathsf{View}_4$. Since View_4 is the view that $\mathcal{S}^{\mathrm{MAL}}_{\mathrm{RAND}}$ produces, and View_0 is the view produced in an execution of $\Pi^{\mathrm{MAL}}_{\mathrm{RAND}}$, we have that $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S}^{\mathrm{MAL}}_{\mathrm{RAND}}}(\vec{x}) \overset{c}{\approx} \mathrm{REAL}_{\Pi^{\mathrm{MAL}}_{\mathrm{RAND}},\mathcal{A}}(\vec{x})$.

□

A.3 Key Combination

Proof of Theorem 7.3: We describe the simulator $\mathcal{S}^{\text{SH}}_{\text{KEYS}}$ for a (static) semi-honest adversary $\mathcal{A}$ corrupting parties in $T \subsetneq [N]$ in Figure 18. We prove that $\text{IDEAL}_{\Pi^{\text{SH}}_{\text{KEYS}},\mathcal{S}^{\text{SH}}_{\text{KEYS}}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\mathcal{F}^{\mathcal{E}}_{\text{KEYS}},\mathcal{A}}(\vec{x})$ using a series of hybrids. The view of the $\mathcal{A}$ consists of

$$\mathsf{View} = \left(\{\mathsf{sk}_i\}_{i\in T} \ , \ \{\mathsf{pk}_i, \mathsf{ek}_i\}_{i\in[N]} \ , \ \left\{ w^{(j,k)}_{d,\tau} \ , \ \widetilde{\mathbf{c}}^{(k)}_{d,\tau} \ , \ \widetilde{\mathbf{z}}^{(j,k)}_{d,\tau} \right\}_{j\in\overline{T},k\in[N]} \right)$$

Simulator $\mathcal{S}^{\text{SH}}_{\text{KEYS}}$

Round 1:

- Runs $\mathsf{params} = \left(\mathbf{a}, \left\{a^{(i)}_{d,\tau}, a'^{(i)}_{d,\tau}\right\}_{d,\tau,i}\right) \leftarrow \mathsf{Setup}(1^\kappa, N)$. When a corrupted party P_i calls $\mathcal{F}^{\mathcal{D}}_{\text{CRS}}$ with sid, sends $(\mathsf{sid}, \mathsf{params})$.
- For $k \in [N]$,
  - Parses $\mathsf{ek}_k := \left\{\widetilde{\mathbf{c}}^{(k)}_{0,d}, \widetilde{\mathbf{c}}^{(k)}_{1,d}, \widetilde{\mathbf{z}}^{(k)}_{0,d}, \widetilde{\mathbf{z}}^{(k)}_{1,d}\right\}_{d\in[D]}$.
  - Verifies that $\widetilde{\mathbf{c}}^{(k)}_{1,d} = (a^{(k)}_{d,1}, \ldots, a^{(k)}_{d,\lfloor \log q_{d-1} \rfloor})$ and $\widetilde{\mathbf{z}}^{(k)}_{1,d} = (a'^{(k)}_{d,1}, \ldots, a'^{(k)}_{d,\lfloor \log q_{d-1} \rfloor})$
  - For all $j \in \overline{T}$ and $k \in [N]$, samples $w^{(j,k)}_{d,\tau} \leftarrow R_{q_d}$ and sends $(\mathsf{sid}, j, w^{(j,k)}_{d,\tau})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$.

Round 2: Upon receiving $(\mathsf{sid}, i, w^{(i,k)}_{d,\tau})$ from $\mathcal{F}_{\text{BC}}$ for all $i \in T, k \in [N]$, $d \in [D]$, $\tau \in \{0, \ldots, \lfloor \log q_{d-1} \rfloor\}$:

- For all $j \in \overline{T}$ and $k \in [N]$, samples $\widetilde{\mathbf{c}}^{(k)}_{d,\tau}, \widetilde{\mathbf{z}}^{(j,k)}_{d,\tau} \leftarrow R^2_{q_d}$ and sends $(\mathsf{sid}, j, \widetilde{\mathbf{c}}^{(j,k)}_{d,\tau}, \widetilde{\mathbf{z}}^{(j,k)}_{d,\tau})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$.

Output whatever $\mathcal{A}$ outputs.

Figure 18: Simulator $\mathcal{S}^{\text{SH}}_{\text{KEYS}}$ for (static) semi-honest adversary $\mathcal{A}$ corrupting parties in $T \subsetneq [N]$ in protocol $\Pi^{\text{SH}}_{\text{KEYS}}$

We simply prove that the joint distribution of these variables in the simulation is computationally indistinguishable from the joint distribution of the same variables in a real-world execution of protocol $\Pi^{\text{SH}}_{\text{KEYS}}$. We further prove that the distribution of ek as output by the simulator is computationally indistinguishable from the output of the algorithm $\mathsf{CombineKeys}$. We let View_i be the joint distribution of these variables in hybrid i.

Hybrid 0: This is a real-world execution of $\Pi^{\text{SH}}_{\text{KEYS}}$. We have

$$\left\{w^{(j,k)}_{d,\tau} := a^{(k)}_{d,\tau} s^{(j)}_d + 2e^{(j,k)}_{d,\tau}\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\widetilde{\mathbf{c}}^{(k)}_{d,\tau} \leftarrow \widetilde{\mathbf{x}}^{(k)}_{d,\tau} + \mathsf{Enc}_{\mathsf{pk}}(0)\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{z}}^{(j,k)}_{d,\tau} \leftarrow \widetilde{\mathbf{y}}^{(j,k)}_{d,\tau} + \mathsf{Enc}_{\mathsf{pk}}(0)\right\}_{j\in\overline{T},k\in[N]}$$

Hybrid 1: We change how we compute $\widetilde{\mathbf{z}}^{(j,k)}_{d,\tau}$. We now choose each pair uniformly at random from $R^2_{q_d}$.

$$\left\{w^{(j,k)}_{d,\tau} := a^{(k)}_{d,\tau} s^{(j)}_d + 2e^{(j,k)}_{d,\tau}\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\widetilde{\mathbf{c}}^{(k)}_{d,\tau} \leftarrow \widetilde{\mathbf{x}}^{(k)}_{d,\tau} + \mathsf{Enc}_{\mathsf{pk}}(0)\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)} \leftarrow R_{q_d}^2\right\}_{j\in\overline{T},k\in[N]}$$

$\mathsf{View}_0 \overset{c}{\approx} \mathsf{View}_1$ because the ciphertext $\mathsf{Enc}_{\mathsf{pk}}(0)$ added to $\widetilde{\mathbf{y}}_{d,\tau}^{(j,k)}$ in Hybrid 0 is *pseudorandom*, and therefore $\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)}$ in Hybrid 0 is pseudorandom as well.

Hybrid 2: We change how we compute $\widetilde{\mathbf{c}}_{d,\tau}^{(k)}$. We now choose each pair uniformly at random from $R_{q_d}^2$.

$$\left\{w_{d,\tau}^{(j,k)} := a_{d,\tau}^{(k)} s_d^{(j)} + 2e_{d,\tau}^{(j,k)}\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\widetilde{\mathbf{c}}_{d,\tau}^{(k)} \leftarrow R_{q_d}^2\right\} \quad , \quad \left\{\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)} \leftarrow R_{q_d}^2\right\}_{j\in\overline{T},k\in[N]}$$

$\mathsf{View}_1 \overset{c}{\approx} \mathsf{View}_2$ because the ciphertext $\mathsf{Enc}_{\mathsf{pk}}(0)$ added to $\widetilde{\mathbf{x}}_{d,\tau}^{(k)}$ in Hybrid 0 is *pseudorandom*, and therefore $\widetilde{\mathbf{c}}_{d,\tau}^{(k)}$ in Hybrid 0 is pseudorandom as well.

Hybrid 3: We change how we compute $w_{d,\tau}^{(j,k)}$. We now choose it uniformly at random from R_{q_d}.

$$\left\{w_{d,\tau}^{(j,k)} \leftarrow R_{q_d}\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\widetilde{\mathbf{c}}_{d,\tau}^{(k)} \leftarrow R_{q_d}^2\right\} \quad , \quad \left\{\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)} \leftarrow R_{q_d}^2\right\}_{j\in\overline{T},k\in[N]}$$

The fact that $\mathsf{View}_2 \overset{c}{\approx} \mathsf{View}_3$ follows from a series of hybrid arguments, one per each $s_d^{(j)}$, in which we use the fact that $w_{d,\tau}^{(j,k)}$ is a PLWE sample with secret $s_d^{(j)}$ and is therefore pseudorandom.

We have proved that $\mathsf{View}_0 \overset{c}{\approx} \mathsf{View}_3$. Since View_3 is the view that $\mathcal{S}_{\text{KEYS}}^{\text{SH}}$ produces, and View_0 is the view produced in an execution of $\Pi_{\text{KEYS}}^{\text{SH}}$, we have that $\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\text{KEYS}}^{\text{SH}}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\Pi_{\text{KEYS}}^{\text{SH}},\mathcal{A}}(\vec{x})$. Furthermore, the distribution of ek as output by the simulator is computationally indistinguishable from the output of $\mathsf{CombineKeys}$. This again follows from a series of hybrid arguments, all based on the PLWE assumption with secret s_d.

□

Proof of Theorem 7.4: We describe the simulator $\mathcal{S}_{\text{KEYS}}^{\text{MAL}}$ for a (static) semi-honest adversary $\mathcal{A}$ corrupting parties in $T \subsetneq [N]$ in Figure 19. We prove that $\text{IDEAL}_{\Pi_{\text{KEYS}}^{\text{MAL}},\mathcal{S}_{\text{KEYS}}^{\text{MAL}}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\mathcal{F}_{\text{KEYS}}^{\mathcal{E}},\mathcal{A}}(\vec{x})$ using a series of hybrids. The view of the $\mathcal{A}$ consists of

$$\mathsf{View} = \left(\{\mathsf{sk}_i\}_{i\in T} \ , \ \{\mathsf{pk}_i, \mathsf{ek}_i\}_{i\in[N]} \ , \ \left\{ w_{d,\tau}^{(j,k)} \ , \ \pi_{d,\tau}^{(j,k)} \ , \ \widetilde{\mathbf{c}}_{d,\tau}^{(k)} \ , \ \phi_{d,\tau}^{(j,k)} \ , \ \widetilde{\mathbf{z}}_{d,\tau}^{(j,k)} \ , \ \varphi_{d,\tau}^{(j,k)} \right\}_{j\in\overline{T},k\in[N]}\right)$$

We simply prove that the joint distribution of these variables in the simulation is computationally indistinguishable from the joint distribution of the same variables in a real-world execution of protocol $\Pi_{\text{KEYS}}^{\text{SH}}$. We further prove that the distribution of ek as output by the simulator is computationally indistinguishable from the output of the algorithm $\mathsf{CombineKeys}$. We let View_i be the joint distribution of these variables in hybrid i.

Hybrid 0: This is a real-world execution of $\Pi_{\text{KEYS}}^{\text{MAL}}$. We have

$$\left\{w_{d,\tau}^{(j,k)} := a_{d,\tau}^{(k)} s_d^{(j)} + 2e_{d,\tau}^{(j,k)}\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\pi_{d,\tau}^{(j,k)} \leftarrow \mathsf{Prove}^{\mathsf{Enc}}((w_{d,\tau}^{(j,k)}, a_{d,\tau}^{(j,k)}), (s_d^{(j)}, e_{d,\tau}^{(j,\tau)}))\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{c}}_{d,\tau}^{(k)} \leftarrow \widetilde{\mathbf{x}}_{d,\tau}^{(k)} + \mathsf{Enc}_{\mathsf{pk}}(0\ ; r_{d,\tau}^{(k)})\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\phi_{d,\tau}^{(k)} \leftarrow \mathsf{Prove}^{\mathsf{Rand}}((\widetilde{\mathbf{c}}_{d,\tau}^{(k)}, \widetilde{\mathbf{x}}_{d,\tau}^{(k)}), (1, r_{d,\tau}^{(k)}))\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)} \leftarrow \widetilde{\mathbf{y}}_{d,\tau}^{(j,k)} + \mathsf{Enc}_{\mathsf{pk}}(0\ ; r_{d,\tau}^{(j,k)})\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\varphi_{d,\tau}^{(j,k)} \leftarrow \mathsf{Prove}^{\mathsf{Rand}}((\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)}, \widetilde{\mathbf{x}}_{d,\tau}^{(k)}), (s_d^{(j)}, r_{d,\tau}^{(j,k)}))\right\}_{j\in\overline{T},k\in[N]}$$

Simulator $\mathcal{S}^{\text{MAL}}_{\text{KEYS}}$

Round 1:

- Runs $\mathsf{params} = \left(\mathbf{a}, \left\{a^{(i)}_{d,\tau}, a'^{(i)}_{d,\tau}\right\}_{d,\tau,i}\right) \leftarrow \mathsf{Setup}(1^\lambda, N)$. When a corrupted party P_i calls $\mathcal{F}^{\mathcal{D}}_{\text{CRS}}$ with sid, sends $(\mathsf{sid}, \mathsf{params})$.
- For $k \in [N]$,
  - Parses $\mathsf{ek}_k := \left\{\widetilde{\mathbf{c}}^{(k)}_{0,d}, \widetilde{\mathbf{c}}^{(k)}_{1,d}, \widetilde{\mathbf{z}}^{(k)}_{0,d}, \widetilde{\mathbf{z}}^{(k)}_{1,d}\right\}_{d\in[D]}$.
  - Verifies that $\widetilde{\mathbf{c}}^{(k)}_{1,d} = (a^{(k)}_{d,1}, \ldots, a^{(k)}_{d,\lfloor \log q_{d-1}\rfloor})$ and $\widetilde{\mathbf{z}}^{(k)}_{1,d} = (a'^{(k)}_{d,1}, \ldots, a'^{(k)}_{d,\lfloor \log q_{d-1}\rfloor})$
  - For all $j \in \overline{T}$ and $k \in [N]$, samples $w^{(j,k)}_{d,\tau} \leftarrow R_{q_d}$ and computes $\pi^{(j,k)}_{d,\tau} \leftarrow \mathsf{Sim}^{\mathsf{Enc}}(w^{(j,k)}_{d,\tau}, a^{(k)}_{d,\tau})$. Sends $(\mathsf{sid}, j, w^{(j,k)}_{d,\tau}, \pi^{(j,k)}_{d,\tau})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$.

Round 2: Upon receiving $(\mathsf{sid}, i, w^{(i,k)}_{d,\tau})$ from $\mathcal{F}_{\text{BC}}$ for all $i \in T, k \in [N]$, $d \in [D]$, $\tau \in \{0, \ldots, \lfloor \log q_{d-1}\rfloor\}$, for all $j \in \overline{T}$ and $k \in [N]$:

- Samples $\widetilde{\mathbf{c}}^{(k)}_{d,\tau}, \widetilde{\mathbf{z}}^{(j,k)}_{d,\tau} \leftarrow R^2_{q_d}$
- Computes $\phi^{(k)}_{d,\tau} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{c}}^{(k)}_{d,\tau}, \widetilde{\mathbf{x}}^{(k)}_{d,\tau})$ and $\varphi^{(j,k)}_{d,\tau} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{z}}^{(i,k)}_{d,\tau}, \widetilde{\mathbf{x}}^{(k)}_{d,\tau})$.
- Sends $(\mathsf{sid}, j, \widetilde{\mathbf{c}}^{(j,k)}_{d,\tau}, \widetilde{\mathbf{z}}^{(j,k)}_{d,\tau}, \phi^{(k)}_{d,\tau}, \varphi^{(j,k)}_{d,\tau})$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\text{BC}}$.

Output whatever $\mathcal{A}$ outputs.

Figure 19: Simulator $\mathcal{S}^{\text{MAL}}_{\text{KEYS}}$ for (static) malicious adversary $\mathcal{A}$ corrupting parties in $T \subsetneq [N]$ in protocol $\Pi^{\text{MAL}}_{\text{KEYS}}$

Hybrid 1: We change how the proofs $\phi^{(k)}_{d,\tau}, \varphi^{(j,k)}_{d,\tau}$ are created. Instead of computing them honestly using the $\mathsf{Prove}^{\mathsf{Rand}}$ algorithm, we compute simulated proofs using the $\mathsf{Sim}^{\mathsf{Rand}}$ algorithm.

$$\left\{w^{(j,k)}_{d,\tau} := a^{(k)}_{d,\tau} s^{(j)}_d + 2e^{(j,k)}_{d,\tau}\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\pi^{(j,k)}_{d,\tau} \leftarrow \mathsf{Prove}^{\mathsf{Enc}}((w^{(j,k)}_{d,\tau}, a^{(j,k)}_{d,\tau}), (s^{(j)}_d, e^{(j,\tau)}_{d,\tau}))\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{c}}^{(k)}_{d,\tau} \leftarrow \widetilde{\mathbf{x}}^{(k)}_{d,\tau} + \mathsf{Enc}_{\mathsf{pk}}(0\,; r^{(k)}_{d,\tau})\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\phi^{(k)}_{d,\tau} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{c}}^{(k)}_{d,\tau}, \widetilde{\mathbf{x}}^{(k)}_{d,\tau})\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{z}}^{(j,k)}_{d,\tau} \leftarrow \widetilde{\mathbf{y}}^{(j,k)}_{d,\tau} + \mathsf{Enc}_{\mathsf{pk}}(0\,; r^{(j,k)}_{d,\tau})\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\varphi^{(j,k)}_{d,\tau} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{z}}^{(j,k)}_{d,\tau}, \widetilde{\mathbf{x}}^{(k)}_{d,\tau})\right\}_{j\in\overline{T},k\in[N]}$$

$\mathsf{View}_0 \overset{c}{\approx} \mathsf{View}_1$ by the *zero-knowledge* property of $\Pi^{(rand)}$.

Hybrid 2: We change how we compute $\widetilde{\mathbf{z}}^{(j,k)}_{d,\tau}$. We now choose each pair uniformly at random from $R^2_{q_d}$.

$$\left\{w^{(j,k)}_{d,\tau} := a^{(k)}_{d,\tau} s^{(j)}_d + 2e^{(j,k)}_{d,\tau}\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\pi^{(j,k)}_{d,\tau} \leftarrow \mathsf{Prove}^{\mathsf{Enc}}((w^{(j,k)}_{d,\tau}, a^{(j,k)}_{d,\tau}), (s^{(j)}_d, e^{(j,\tau)}_{d,\tau}))\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{c}}^{(k)}_{d,\tau} \leftarrow \widetilde{\mathbf{x}}^{(k)}_{d,\tau} + \mathsf{Enc}_{\mathsf{pk}}(0\,; r^{(k)}_{d,\tau})\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\phi^{(k)}_{d,\tau} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{c}}^{(k)}_{d,\tau}, \widetilde{\mathbf{x}}^{(k)}_{d,\tau})\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{z}}^{(j,k)}_{d,\tau} \leftarrow R^2_{q_d}\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\varphi^{(j,k)}_{d,\tau} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{z}}^{(j,k)}_{d,\tau}, \widetilde{\mathbf{x}}^{(k)}_{d,\tau})\right\}_{j\in\overline{T},k\in[N]}$$

$\mathsf{View}_1 \overset{c}{\approx} \mathsf{View}_2$ because the ciphertext $\mathsf{Enc}_{\mathsf{pk}}(0)$ added to $\widetilde{\mathbf{y}}^{(j,k)}_{d,\tau}$ in Hybrid 1 is *pseudorandom*, and therefore $\widetilde{\mathbf{z}}^{(j,k)}_{d,\tau}$ in Hybrid 1 is pseudorandom as well.

Hybrid 3: We change how we compute $\widetilde{\mathbf{c}}_{d,\tau}^{(k)}$. We now choose each pair uniformly at random from $R_{q_d}^2$.

$$\left\{w_{d,\tau}^{(j,k)} := a_{d,\tau}^{(k)} s_d^{(j)} + 2e_{d,\tau}^{(j,k)}\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\pi_{d,\tau}^{(j,k)} \leftarrow \mathsf{Prove}^{\mathsf{Enc}}((w_{d,\tau}^{(j,k)}, a_{d,\tau}^{(j,k)}), (s_d^{(j)}, e_{d,\tau}^{(j,\tau)}))\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{c}}_{d,\tau}^{(k)} \leftarrow R_{q_d}^2\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\phi_{d,\tau}^{(k)} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{c}}_{d,\tau}^{(k)}, \widetilde{\mathbf{x}}_{d,\tau}^{(k)})\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)} \leftarrow R_{q_d}^2\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\varphi_{d,\tau}^{(j,k)} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)}, \widetilde{\mathbf{x}}_{d,\tau}^{(k)})\right\}_{j\in\overline{T},k\in[N]}$$

$\mathsf{View}_2 \overset{c}{\approx} \mathsf{View}_3$ because the ciphertext $\mathsf{Enc}_{\mathsf{pk}}(0)$ added to $\widetilde{\mathbf{x}}_{d,\tau}^{(k)}$ in Hybrid 2 is *pseudorandom*, and therefore $\widetilde{\mathbf{c}}_{d,\tau}^{(k)}$ in Hybrid 2 is pseudorandom as well.

Hybrid 4: We change how the proofs $\phi_{d,\tau}^{(k)}, \pi_{d,\tau}^{(j,k)}$ are created. Instead of computing them honestly using the $\mathsf{Prove}^{\mathsf{Enc}}$ algorithm, we compute simulated proofs using the $\mathsf{Sim}^{\mathsf{Enc}}$ algorithm.

$$\left\{w_{d,\tau}^{(j,k)} := a_{d,\tau}^{(k)} s_d^{(j)} + 2e_{d,\tau}^{(j,k)}\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\pi_{d,\tau}^{(j,k)} \leftarrow \mathsf{Sim}^{\mathsf{Enc}}(w_{d,\tau}^{(j,k)}, a_{d,\tau}^{(j,k)})\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{c}}_{d,\tau}^{(k)} \leftarrow R_{q_d}^2\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\phi_{d,\tau}^{(k)} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{c}}_{d,\tau}^{(k)}, \widetilde{\mathbf{x}}_{d,\tau}^{(k)})\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)} \leftarrow R_{q_d}^2\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\varphi_{d,\tau}^{(j,k)} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)}, \widetilde{\mathbf{x}}_{d,\tau}^{(k)})\right\}_{j\in\overline{T},k\in[N]}$$

$\mathsf{View}_3 \overset{c}{\approx} \mathsf{View}_4$ by the *zero-knowledge* property of $\Pi^{(enc)}$.

Hybrid 5: We change how we compute $w_{d,\tau}^{(j,k)}$. We now choose it uniformly at random from R_{q_d}.

$$\left\{w_{d,\tau}^{(j,k)} \leftarrow R_{q_d}\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\pi_{d,\tau}^{(j,k)} \leftarrow \mathsf{Sim}^{\mathsf{Enc}}(w_{d,\tau}^{(j,k)}, a_{d,\tau}^{(j,k)})\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{c}}_{d,\tau}^{(k)} \leftarrow R_{q_d}^2\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\phi_{d,\tau}^{(k)} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{c}}_{d,\tau}^{(k)}, \widetilde{\mathbf{x}}_{d,\tau}^{(k)})\right\}_{j\in\overline{T},k\in[N]}$$

$$\left\{\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)} \leftarrow R_{q_d}^2\right\}_{j\in\overline{T},k\in[N]} \quad , \quad \left\{\varphi_{d,\tau}^{(j,k)} \leftarrow \mathsf{Sim}^{\mathsf{Rand}}(\widetilde{\mathbf{z}}_{d,\tau}^{(j,k)}, \widetilde{\mathbf{x}}_{d,\tau}^{(k)})\right\}_{j\in\overline{T},k\in[N]}$$

The fact that $\mathsf{View}_4 \overset{c}{\approx} \mathsf{View}_5$ follows from a series of hybrid arguments, one per each $s_d^{(j)}$, in which we use the fact that $w_{d,\tau}^{(j,k)}$ is a PLWE sample with secret $s_d^{(j)}$ and is therefore pseudorandom.

We have proved that $\mathsf{View}_0 \overset{c}{\approx} \mathsf{View}_5$. Since View_5 is the view that $\mathcal{S}_{\text{KEYS}}^{\text{MAL}}$ produces, and View_0 is the view produced in an execution of $\Pi_{\text{KEYS}}^{\text{MAL}}$, we have that $\text{IDEAL}_{\mathcal{F},\mathcal{S}_{\text{KEYS}}^{\text{MAL}}}(\vec{x}) \overset{c}{\approx} \text{REAL}_{\Pi_{\text{KEYS}}^{\text{MAL}},\mathcal{A}}(\vec{x})$. Furthermore, the distribution of ek as output by the simulator is computationally indistinguishable from the output of $\mathsf{CombineKeys}$. This again follows from a series of hybrid arguments, all based on the PLWE assumption with secret s_d.

□