A New Paradigm for Server-Aided MPC

Alessandra Scafuro and Tanner Verber*

North Carolina State University, Raleigh, NC

January 8, 2025

Abstract

The server-aided model for multiparty computation (MPC) was introduced to capture a real-world scenario where clients wish to off-load the heavy computation of MPC protocols to dedicated servers. A rich body of work has studied various trade-offs between security guarantees (e.g., semi-honest vs malicious), trust assumptions (e.g., the threshold on corrupted servers), and efficiency.

However, all existing works make the assumption that all clients must agree on employing the same servers, and accept the same corruption threshold. In this paper, we challenge this assumption and introduce a *new paradigm for server-aided MPC*, where each client can choose their own set of servers and their own threshold of corrupted servers. In this new model, the privacy of *each* client is guaranteed as long as their own threshold is satisfied, *regardless* of the other servers/clients. We call this paradigm *per-party private* server-aided MPC to highlight both a security and efficiency guarantee: (1) *per-party privacy*, which means that each party gets their *own privacy guarantees* that depend on their *own* choice of the servers; (2) *per-party complexity*, which means that each party only needs to communicate with *their chosen servers*. Our primary contribution is a new theoretical framework for server-aided MPC. We provide two protocols to show feasibility, but leave it as a future work to investigate protocols that focus on concrete efficiency.

1 Introduction

Multiparty Computation: Secure multiparty computation (MPC) is a cryptographic technique that allows a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, where each party P_i holds secret input x_i, to compute some agreed upon function $f(x_1, ..., x_n)$ without revealing their inputs to each other. This problem was first addressed by Yao [Yao86] for the two party case, using what are now called garbled circuits, and Goldreich, Micali, and Wigderson with the GMW protocol [GMW87] for the multiparty case. In these works, the communication and computation complexity of each party is proportional to the complexity of the circuit. In the case of Yao's garbled circuits, one party must garble the entire circuit and one must evaluate the resulting garbled circuit which allows for computation in constant rounds. GMW is based on secret sharing that allows for local addition, but requires parties to interact for multiplication,

*Alessandra Scafuro and Tanner Verber are supported by a research grant from Horizen Labs

meaning the number of rounds of communication is dependent on the number of multiplication gates in the circuit. Beaver, Micali, and Rogaway [BMR90] built on Yao's garbled circuits to extend it to the multiparty setting. This allowed multiple parties to compute a circuit in constant rounds, whereas Yao's approach only worked for two parties. However, the communication complexity still depends on the circuit.

Because the communication required depends on the complexity of the circuit, complicated functions could become impractical to compute if parties have modest processing power and bandwidth. The natural next step was to develop protocols that allow multiple parties to securely compute a circuit without requiring communication proportional to the circuit. To reduce this burden on the parties who hold the input, server-aided MPC was introduced.

Server-aided Multiparty Computation: Server-aided MPC is a term coined by Kamara, Mohassel, and Raykova [KMR11] and refers to protocols that make use of a designated set of servers to perform most of the communication and computation on behalf of the input holders. The model was introduced to reduce the work of the input holders, who are often referred to as the clients.

Server-aided MPC has been shown using any number of designated servers, from a single server (e.g., [FKN94, HLP11, KMR11, KMR12, LTV12]) to multiple servers (e.g., [DI05, MGBF14, JNO14, GLO+21]), with varying levels of savings for client computation and/or communication, and varying levels of security guarantees (we describe these works at greater length in Related Work (Section 1.2)). Very recently, server-aided MPC has gained renewed attention for its realistic use case of privacy-preserving machine learning (PPML). This is because machine learning (ML) often improves with larger training data sets; hence clients may want to pool their, potentially large, data sets to build a shared and more robust model. Server-aided MPC protocols that are highly optimized for ML computations have been shown with two [MZ17, MTZC21], three [MR18, CCPS19, PS20, TKTW21, KPPS21], and four [BCPS20, KPPS21] servers, while tolerating at most one corrupt server.

Drawbacks of Current Server-Aided Paradigm. All existing work that leverages server-aided MPC operates under the assumption that *all clients agree on the same set of servers*[1]. Further, clients must have the same corruption threshold. By corruption threshold, we mean the number of corrupt servers the protocol can tolerate without the clients losing their input privacy. For example, the clients might pick three servers and tolerate one being corrupt. If two of the three servers are corrupt, then the adversary would be able to learn the input of all honest clients. The three chosen servers, and the threshold of one, would be something that the clients need to agree on before they can begin executing the protocol.

Forcing clients to agree on a set of servers and a corruption threshold has two primary drawbacks:

- **Diminished Practicality:** All clients must agree on the *same* set of dedicated servers. This may be difficult to achieve in the real world as clients may have existing contracts with different sets of servers. For some clients, this would mean sending their (encoded) data to a new server in order to participate in the protocol rather than hiring

[1]Some recent models [CGG+21, GHK+21] allow the set to change over time, but all clients still use the same sets of servers throughout computation.

the server that already holds their data. Requiring the client to move their data to a new server may dissuade them from participating in the MPC protocol altogether.

- **Everyone or no-one privacy:** If the number of corrupt servers surpasses the fixed threshold, then *all* clients lose their input privacy. If most clients trust the chosen servers, they may set the corruption threshold low. A client who has little trust in the servers then would have to either agree with this low threshold and risk losing their input privacy or opt out of participating in the protocol. This again can dissuade a particularly distrustful client, or a client with particularly sensitive data, from participating in the MPC protocol.

This paper aims to overcome these drawbacks with a new approach for server-aided MPC. In our new paradigm, clients can choose *their own* set of servers and *their own* threshold of how many servers can misbehave, and have the guarantee that as long as their own assumption is correct, their input is *private*, *regardless of what other clients chose and what other servers did*. We stress that our new paradigm focuses on per-party *privacy* and *efficiency*. Correctness of the result might not guaranteed if other servers misbehave (since they can change the input of the clients they represent).

1.1 A New Paradigm: Per-party Private Server-aided MPC

We propose a new paradigm for server-aided MPC called per-party private server-aided MPC. We consider a setting in which each of the n clients $C_i \in \mathcal{C}$ chooses their own set of m_i servers $\mathcal{S}_{C_i} \in \mathcal{S}$ and their own threshold $1 \leq \tau_i < m_i$ of how many corrupt servers to tolerate. C_i will interact *only* with their own chosen servers $\mathcal{S}_{C_i}$. Further, as long as the number of malicious servers in $\mathcal{S}_{C_i}$ is at most τ_i, then C_i's input remains private. If the malicious servers in $\mathcal{S}_{C_i}$ surpass this threshold, however, they will learn C_i's private input and will be able to maliciously replace it with an input of their choosing. We do note that if a single client's corruption threshold is surpassed, the correctness of the output could be compromised for all clients, but the privacy of clients who did not surpass their own threshold will hold, beyond what can be inferred through the output of the function.

Our new paradigm targets two specific properties: per-party efficiency and per-party privacy.

- **Per-Party Efficiency:** We require that each client communicate *only* with their chosen set of servers. A client should be able to learn the output of the MPC protocol without talking to any other client, or any other server that they have not specifically chosen to work on their behalf [2].

- **Per-party Privacy for Clients:** The privacy of a client's input is guaranteed as long as the number of corrupt servers in their chosen set is below their chosen threshold, aside from what can be learned through the function output. This means that even if all other clients chose exclusively malicious servers, an honest client who was able to stay below their threshold of corrupt servers will retain their input privacy.

[2] As we explain in the related work section (Section 1.2), this efficiency requirement rules out the possibility of hiring one server only.

For example, a *privacy-oriented* client may choose to hire m servers with a threshold of $m-1$, trusting the minimum number of servers possible (concretely, the most conservative might choose to hire only 2 servers and trust only 1). While a different client, who is more focused on *robustness* (i.e., does not want to lose their data) may choose many servers m and set a more relaxed threshold of $m/3$ (concretely, a client might choose 5 servers and have their tolerated threshold to be at most 2).

To better understand the per-party private server-aided setting, consider the following example. Suppose there are three clients, Alice, Bob, and Charlie, who want to perform MPC but cannot agree on a set of servers or a threshold, so they use our model. Alice chooses 2 servers, AWS and Azure, with a threshold of 1, Bob chooses Google, IBM, and Red Hat with threshold 2, and Charlie chooses Google, Azure, and Oracle with a threshold of 1. Each client will communicate only with their own servers (i.e., Alice only communicates with AWS and Azure) and only sends input and receives output. Even if Google, IBM, Red Hat, Oracle, and Azure are all corrupt, Alice's input will *still* remain private, aside from what Google, IBM, Red Hat, Oracle, and Azure together could learn through the output of the function, as she has not surpassed her threshold. This is true even though both Bob and Charlie will lose their input privacy.

Contributions Our contributions can be summarized as follows:

- *Per-party privacy paradigm:* We formally define our new paradigm in the Universally Composable Framework [Can01], through an ideal functionality $\mathcal{F}_{\text{Per-Party}}$, presented in Section 3 (Figure 3). In this paradigm, we capture a setting where a client's privacy depends only on their own choices. As long as a client has chosen less than their threshold of corrupt servers, their input remains private. Further, a client only communicates with their chosen servers in our protocol.

  Our new paradigm generalizes standard server-aided MPC. If each client chooses the same set of servers and the same threshold, then this becomes standard server-aided; a single set of dedicated servers performing computation on behalf of the clients. However, the clients will still only communicate with the servers and never the other clients.

- *Feasibility result:* We provide two natural approaches to building a per-party private and efficient server-aided MPC protocol in Section 4. The first protocol is based on general MPC, and the second is a slightly more specific construction based on FHE. Both protocols are built based off of the same idea: clients share their input with their chosen servers via verifiable secret sharing, and the servers perform secure computation among themselves.

  Both approaches make use of verifiable secret sharing (VSS) and signatures. These primitives allow a client to share their input securely with their chosen set of servers, in a publicly verifiable manner. VSS allows a client to share their input securely with their chosen set of servers, and the signatures allow servers to confirm that a particular share has not been altered. Hence, in both our protocols the function f is augmented with an input validation phase, where the signatures on the VSS shares are validated and the shares are then reconstructed, and an output sharing phase, where the output of f is secret shared again. We denote this augmented function by g.

The first protocol $\Pi_{\mathsf{Per\text{-}Party}}$ (Figure 5) uses an underlying $(m, m-1)$ UC-secure MPC protocol Π^g, which is executed by the m servers only. The servers, upon receiving their shares, use Π^g to securely reconstruct the input of the clients using the received shares. Then, the reconstructed inputs are used to compute the function $f(x_1, \ldots, x_n)$ and share the output. At the end of the execution of Π^g, each server receives a share of their client's output. These output shares are then sent to the clients for reconstruction.

Note that we need the MPC protocol to be secure for $(m, m-1)$ corruptions. A client's privacy must depend only on their own choices, therefore we need that the computation performed by the servers is secure even if a client has chosen only one honest server (if they have chosen the maximum threshold) and all other servers chosen by all other clients are corrupt.

In the second protocol $\Pi^{FHE}_{\mathsf{Per\text{-}Party}}$ (Figure 14), instead of running a general MPC protocol, the servers use FHE. Therefore, $\Pi^{FHE}_{\mathsf{Per\text{-}Party}}$ can be thought of as an instantiation of $\Pi_{\mathsf{Per\text{-}Party}}$ where the general MPC protocol is replaced by FHE. The servers first collaboratively execute the key generation algorithm of a fully homomorphic encryption (FHE) scheme to receive the public key and a share of the secret key. This is done using an $(m, m-1)$ UC-secure MPC protocol Π^{KG}. They are then able to encrypt their shares and broadcast the ciphertext to all other servers. Upon receiving all ciphertexts, a computing server then homomorphically evaluates the function g. After the homomorphic computation, the computing server sends the output ciphertexts to all servers, along with a succinct non-interactive argument (SNARG) that the computation was performed honestly. The servers then use another $(m, m-1)$ UC-secure MPC protocol Π^{Dec} to decrypt the ciphertexts and obtain output shares, which are sent to their respective clients.

In both protocols, a client is able to choose their own threshold using the parameters of the VSS scheme. Further, a client only sends shares to their servers and receives shares. Therefore, they need only communicate with their own servers.

Further, in both of our provided protocols, the communication complexity of the clients is independent of the function. The computation of a client is limited to only sharing their input and reconstructing their output. Therefore, a client who wishes to further reduce their workload may opt for a lower threshold.

The two protocols offer trade-offs of communication and computation complexity. Protocol $\Pi_{\mathsf{Per\text{-}Party}}$, has high communication overhead for all servers but since it is based on an MPC-protocol, involves cryptographic primitives that are much faster than FHE evaluations (e.g. MPC protocols such as "SPDZ" [DPSZ12] involves mainly symmetric-key primitives). $\Pi_{\mathsf{Per\text{-}Party}}$ should be preferred when all servers can be online and actively participate in the protocol.

Protocol $\Pi^{FHE}_{\mathsf{Per\text{-}Party}}$ off-loads the burden of the computation on one computing server only, who evaluates the function homomorphically. The other servers only engage in the MPC for key generation and ciphertext decryption, which are independent of the function. The evaluator, however, must additionally compute a SNARG showing correct evaluation. This SNARG is computed on the input and output ciphertexts, not within the FHE, but is dependent on the function. Because the majority of the work is offloaded to an evaluat-

ing server, $\Pi^{FHE}_{\text{Per-Party}}$ should be preferred when not all servers can be online for the entire computation and/or have varying levels of computational ability.

Both protocols, however, are costly. In both cases, we have input that is shared under VSS being reconstructed, a function that is being computed, and output to be shared under VSS, all performed either in MPC or FHE. The primary challenge in constructing efficient Per-Party private protocols is that a client must assume that all servers chosen by any other client may be malicious. This, along with tolerating an all but one threshold for the client's own set, leads to the requirement that the inner protocol assume all but one servers are malicious.

1.2 Related Work

Server-Aided MPC Server-aided MPC was introduced to reduce the work done by the clients by offloading the work to a set of designated servers. There are two primary approaches to server-aided MPC: single server (e.g., [FKN94,KMR11,HLP11,KMR12,CKKC13, CMTB16, BPP+17, MOR16]), and multi-servers (e.g., [MZ17, MR18, CCPS19, BCPS20, PS20, WGC19]).

Most existing single-server protocols either require clients to do work proportional to the circuit [FKN94, HLP11], require the clients to interact with one another [KMR11, CMTB16, BPP+17], or require communication between the clients and server during execution [CKKC13]. In contrast, we provide a protocol in which the clients connect only to their servers and only to provide input and to learn output, do not need to interact with each other, and the input of the clients remains private as long as the number of corrupt servers they chose is below their chosen threshold.

The server-aided model has been heavily employed for performing machine learning training and inference in a privacy-preserving manner. In Privacy-Preserving Machine Learning (PPML) clients with potentially very large datasets outsource the training and/or inference of ML models to a designated set of servers. There are server-aided protocols for private logistic and linear regression [MZ17,MR18,CCPS19,BCPS20,PS20,KPPS21], support vector machines [CCPS19], neural networks [MZ17,MR18,BLCW19,BCPS20,PS20,TKTW21, KPPS21], decision trees [MTZC21]. The protocols to build these models are built on a variety of cryptographic techniques. ABY^3 [MR18] provided a general framework for ML in the three-server setting built on secret sharing and garbled circuits. BLAZE [PS20] introduced a three-server protocol built on secret sharing that makes use of an input-dependent pre-processing phase to support an efficient online phase. Lastly, Ma, Zhao, and Chow [MTZC21] present a protocol in the two-server model, that allows clients to outsource decision tree inference to two servers using secret sharing and garbled circuits.

In all such works, the clients must agree on the same group of servers and accept the corruption threshold fixed by the protocol.

MPC with Dynamic Parties A recent line of research in MPC is in allowing dynamic sets of parties to participate in the execution of the protocol. Damgård, Escudero, and Polychroniadou model this in Phoenix [DEP23], where the adversary can force honest parties offline for some amount of the computation. These parties are later able to rejoin the protocol without being considered malicious, and the protocol is able to continue execution without

them. With Phoenix, Damgård, Escudero, and Polychroniadou model the realistic setting where honest parties may fail and be put offline for some amount of time.

In Fluid MPC [CGG+21] the servers are allowed to participate in subsets of execution rather than the entire protocol. This is taken even further with maximal fluidity, where parties need only stay for one round of communication. Rachuri and Scholl [RS22] combined Fluid MPC and SPDZ to create a fluid MPC protocol that is secure against a malicious adversary controlling all but one of the parties involved. Bienstock, Escudero, and Polychroniadou [BEP23] presented Fluid MPC protocols that require only $O(n)$ communication per-gate, improving over the existing protocols requiring $O(n^2)$ where n is the number of parties online in a given round. David et al. [DDG+23] consider a layered MPC model, where parties can be organized as a layered graph and parties communicate only with the next layer. David et al. extend the results of BGW [BGW88] and achieve perfect full security in the layered setting with a corruption threshold of $n/3$.

Finally, YOSO models dynamic participation in a slightly different way. YOSO ("you only speak once") [GHK+21] protocols are protocols where parties only speak once during their participation in the protocol. A maximally fluid MPC protocol [CGG+21] is then considered a YOSO protocol if each set of parties in each round is independent.

In each of these works, clients still use the same set of servers throughout computation, differing from our protocols.

Differing Opinions on Set Ups The work of [GGJS11] and [GO14] also investigate the question of what happens if parties do not have the same belief about trustworthy servers.

Garg, Jain, and Sahai [GGJS11] show how to do MPC when the different parties involved in the protocol have different beliefs about which set up is to be trusted (e.g., common reference string (CRS), random oracle [BR93], or token hardware [Kat07]), and a party's privacy is guaranteed if their beliefs hold true. Groth and Ostrovsky [GO14] pose a similar question, but for the cases in which there are multiple CRSs and not all of them are trusted by all the participants, and show how MPC can be done when only a majority of them were computed honestly. Lastly, recent works have studied the setting where parties are assigned a trust grade [SH21] or weight [GJM+23]. That is, certain parties are deemed more likely to be honest than others, and privacy depends on the sum of the weight of the corrupted parties being below some level. As with MPC with dynamic parties, these works are not directly related to our work but are worth considering when discussing how the setting of MPC has grown.

MPC via FHE. Many works explored the using homomorphic encryption to reduce the computation overhead of clients in secure computation [AJL+12, LTV12, CKL21]. FHE allows the clients to encrypt their inputs and send them to a server, who performs the computation on their behalf. The main challenge here is to establish the key for FHE in a secure manner. Multi-key FHE was introduced in [LTV12, AJJM20] to allow servers to operate on ciphertexts computed under different keys. Such approaches are not applicable in our setting where we require that clients only speak with their own set of servers (and not other clients/servers).

2 Preliminaries

In the following subsections, we provide background on the building blocks used to construct our protocols.

2.1 Secure Multiparty Computation (MPC)

Secure multiparty computation (MPC) protocols allow a group of n parties $\mathcal{C} = \{C_1, \ldots, C_n\}$, each with their own private inputs x_i, to compute some function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ such that each party learns only their output y_i. An (n, t) MPC protocol is one in which there are n parties providing inputs and t corrupt parties are tolerated.

Security of an MPC protocol is shown through the real world-ideal world paradigm [Can01, Lin17]. In this paradigm, there exists an environment $\mathcal{Z}$ controlling the inputs of the corrupt parties. In the real world, the MPC protocol Π is performed as it would be in practice; there are clients and servers, some of them corrupted by a PPT adversary $\mathcal{A}$.

In the ideal world, there is an ideal functionality $\mathcal{F}_f$ that is a trusted third party for computing $f(x_1, \ldots, x_n)$. There is an ideal PPT adversary, $\mathcal{SIM}$, simulating the protocol for the corrupt parties. The goal of the simulator is to extract the inputs of the corrupt parties to provide to the ideal functionality, ensure the corrupt parties receive the correct output, and simulate the protocol such that the environment cannot distinguish between a real and ideal execution.

Let Π be an MPC protocol, $\mathcal{F}_f$ be an ideal functionality for computing $f : (\{0,1\}^\lambda)^n \to (\{0,1\}^\lambda)^n$, and $\mathsf{negl}(\lambda)$ be a negligible function in the security parameter λ. Define the security of an MPC protocol as follows:

Definition 1 *Let $n \in \mathbb{N}$, $\mathcal{F}_f$ be an ideal functionality, and Π be an n-party protocol. We say that Π securely realizes $\mathcal{F}_f$ if for every real-world adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{SIM}$ in the ideal world, controlling the same parties, such that for any environment $\mathcal{Z}$*

$$\texttt{REAL}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \texttt{IDEAL}_{\mathcal{F}_f,\mathcal{SIM},\mathcal{Z}}$$

2.2 Verifiable Secret Sharing (VSS)

Verifiable secret sharing (VSS) allows a dealer C to choose a threshold τ and send shares of a secret value s to a set of m parties $\mathcal{S} = \{S_1, \ldots, S_m\}$ such that no subset of parties $\mathcal{P} \subseteq \mathcal{S} \cup C$ with $|\mathcal{P}| \leq \tau < m$ can recover the secret. Secret sharing becomes verifiable when we can guarantee that the shares do encode some secret, regardless of whether the dealer was malicious. That is, given a set of more than τ shares, reconstruction will not fail.

VSS is used to distribute secrets in the presence of a malicious dealer. Verifying the shares allows the parties in $\mathcal{S}$ to ensure that they were not sent an invalid share by the malicious dealer. A VSS scheme consists of two algorithms:

- $\mathsf{Share}(s, \tau, m)$: On input a secret s, a threshold τ, and the number of parties to receive shares m, this algorithm outputs m shares $(s_1, \ldots, s_m)$
- $\mathsf{Reconst}(s_1, \ldots, s_{\tau+1})$: On input a set of $\tau + 1$ shares, this algorithm outputs the secret s

$\mathsf{Sig\text{-}Forge}_{\mathcal{A},\Sigma}(\lambda)$

1. $(pk^{\Sigma}, sk^{\Sigma}) = \Sigma.\mathsf{Gen}(1^{\lambda})$.
2. $(m, \sigma) \leftarrow \mathcal{A}^{\Sigma.\mathsf{Sign}_{sk}(\cdot)}(pk)$ let $\mathcal{Q}$ be the set of queries $\mathcal{A}$ asked to the oracle.
3. If $1 = \Sigma.\mathsf{Verify}(pk^{\Sigma}, m, \sigma)$ and $m \notin \mathcal{Q}$ output 1; otherwise output 0.

Figure 1: EUF-CMA Game for Σ

We give a definition of VSS [CCP22] in Definition 2.

Definition 2 *Let* $\mathsf{VSS} = (\mathsf{Share}, \mathsf{Reconst})$ *be a set of protocols for a dealer* C *to distribute a secret* s *to* m *parties* $\mathcal{S}$ *such that no subset of parties holding at most* $\tau < m$ *shares can recover the original secret. We say* VSS *is a* Verifiable Secret Sharing Scheme *if it satisfies the following properties in the presence of malicious adversary* $\mathcal{A}$ *controlling less than* $\tau + 1$ *parties:*

- ***Privacy:*** *If* C *is honest, then the view of* $\mathcal{A}$ *after* Share *contains no a posteriori information about the original secret* s
- ***Correctness:*** *If* C *is honest, then at the end of* Share *the parties* $\mathcal{S}$ *each hold shares of* s*. With these shares, any subset of parties holding more than* τ *valid shares can use* Reconst *to retrieve the original secret* s*.*
- ***Committed:*** *If* C *is corrupt, then at the end of* Share *the parties* $\mathcal{S}$ *each hold shares of some value* s'*, potentially different from* s*. With these shares, any subset of parties holding more than* τ *valid shares can use* Reconst *to retrieve the secret* s'*.*

2.3 Digital Signatures

Digital signatures allow a party C to sign a message, such that any other party, given the signer's verification key, can verify that the message came from the client. A digital signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ for a message space M is a tuple of three PPT algorithms:

- $\mathsf{Gen}(1^{\lambda})$: The key generation algorithm that takes 1^{λ} as input and outputs a pair of public and private keys $(pk^{\Sigma}, sk^{\Sigma})$.
- $\mathsf{Sign}(sk^{\Sigma}, m)$: The signing algorithm that takes a private key sk^{Σ} and a message m from the message space M as input and outputs a signature σ.
- $\mathsf{Verify}(pk^{\Sigma}, \sigma, m)$: The verification algorithm that takes a public key pk^{Σ}, a message m, and a signature σ as input and outputs 1 if a valid signature and 0 if an invalid signature.

We present the existential unforgeability under chosen message attacks game $\mathsf{Sig\text{-}Forge}_{\mathcal{A},\Sigma}$ in Figure 1 to capture unforgeability.

We give the definition of unforgeable signatures [KL14] in Definition 3.

Definition 3 *A signature scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is* **existentially unforgeable under adaptive chosen-message attack** *if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$\Pr[\mathsf{Sig\text{-}Forge}_{\mathcal{A},\Sigma}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

Further, Σ *is correct if for* $(pk^{\Sigma}, sk^{\Sigma}) = \Sigma.\mathsf{Gen}(1^{\lambda})$, $m \in M$, *it holds that* $1 = \Sigma.\mathsf{Verify}(pk^{\Sigma}, m, \Sigma.\mathsf{Sign}(sk^{\Sigma}, m))$ *except with a negligible probability.*

2.4 Succinct Non-Interactive Arguments (SNARG)

A succinct non-interactive argument (SNARG) [Mic94, BCC+17, KPY19, CCH+19, JKKZ21, CJJ21, HJKS22] allows a prover P to generate a proof π for a statement $stmt$ using a witness w such that any party given $(\pi, stmt)$ can verify that $(stmt, w) \in \mathcal{R}$ for some relation $\mathcal{R}$. In this work, we consider SNARGs for relations in P. Further, to be succinct, the size of the proof is sublinear in the size of w. A SNARG consists of a pair of algorithms:

- $\mathsf{SetUp}(1^{\lambda})$: On input the security parameter λ, this algorithm outputs a common reference string crs
- $\mathsf{Prove}(crs, stmt, w)$: On input a statement $stmt$ and a witness w, this algorithm outputs a proof π
- $\mathsf{Verify}(crs, \pi, stmt)$: On input a statement $stmt$ and a proof π, this algorithm outputs 0 or 1

We give the formal definition of a SNARG in Definition 4 [HJKS22].

Definition 4 *Let* $\mathcal{R}$ *be an NP relation.* $\mathsf{SNARG} = (\mathsf{SetUp}, \mathsf{Prove}, \mathsf{Verify})$ *is a* **succinct non-interactive argument** *if*

- **Completeness:** *For any statement stmt and witness w such that* $(stmt, w) \in \mathcal{R}$ *it holds that*

$$Pr\left[\mathsf{SNARG.Verify}(crs, \pi, stmt) = 1 \;\middle|\; \begin{array}{c} crs \leftarrow \mathsf{SNARG.SetUp}(1^{\lambda}) \\ \mathsf{SNARG.Prove}(crs, stmt, w) = \pi \end{array}\right] = 1$$

- **Soundness:** *There exists a negligible function* negl *such that for any PPT adversary* $\mathcal{A}$ *it holds that*

$$Pr\left[(stmt, w) \notin \mathcal{R} \;\middle|\; \begin{array}{c} crs \leftarrow \mathsf{SNARG.SetUp}(1^{\lambda}) \\ (stmt, \pi) \leftarrow \mathcal{A}(crs) \\ \mathsf{SNARG.Verify}(crs, stmt, \pi) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda)$$

for any witness w

$\mathsf{PubK}^{cpa}_{\mathcal{A},\mathsf{FHE}}(\lambda)$

1. $(pk, evk, sk) = \mathsf{FHE.KeyGen}(1^\lambda)$.
2. Send (pk, evk) to $\mathcal{A}$.
3. Receive equal length messages m_0, m_1 from $\mathcal{A}$
4. Choose a uniform bit $b \xleftarrow{\$} \{0,1\}$
   4.1 Compute $\mathbf{ct}^* = \mathsf{FHE.Enc}(pk, m_b)$
   4.2 Send $\mathbf{ct}^*$ to $\mathcal{A}$
5. Upon receipt of bit b' from $\mathcal{A}$, if $b' = b$ output 1, else output 0

Figure 2: IND-CPA game for FHE

2.5 Fully Homomorphic Encryption (FHE)

A fully homomorphic encryption (FHE) scheme [Gen09] is an encryption scheme that allows for any circuit of any depth to be evaluated on the ciphertexts. An FHE scheme $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ is a tuple of algorithms defined as follows:

- $\mathsf{KeyGen}(1^\lambda\ ;\ r)$: On input the security parameter λ, this algorithm outputs a public key pk, an evaluation key evk, and a secret key sk. r is the implicit randomness used in key generation
- $\mathsf{Enc}(pk, m)$: On input the public key pk and a message m, this algorithm outputs a ciphertext $\mathbf{ct}$
- $\mathsf{Eval}(evk, f, \{\mathbf{ct}_1, \ldots, \mathbf{ct}_n\})$: On input the evaluation key evk, a function f, and a set of ciphertexts $\{\mathbf{ct}_1, \ldots, \mathbf{ct}_n\}$, this function outputs a ciphertext $\mathbf{ct}$. Execution implicitly results in $trans$, the transcript of computation, consisting of the intermediate states of computation
- $\mathsf{Dec}(sk, \mathbf{ct})$: On input the secret key sk and a ciphertext $\mathbf{ct}$, this algorithm outputs the value m^*

First, we give the CPA indistinguishability expirement [KL14] in Figure 2.

We give the formal definition of CPA security in Definition 5 [KL14] and of full homomorphism in Definition 6 [BV14].

Definition 5 *An encryption scheme* $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is* **IND-CPA secure** *if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

$$Pr[\mathsf{PubK}^{cpa}_{\mathcal{A},\mathsf{FHE}}(\lambda) = 1] \leq 1/2 + \mathsf{negl}(\lambda)$$

Definition 6 *An encryption scheme* $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is* **Fully Homomorphic** *if for any function* $f : \{\{0,1\}^\lambda\}^n \to \{0,1\}^*$ *for all* $n \in \mathbb{N}$

$$Pr[\mathsf{FHE.Dec}(sk, \mathsf{FHE.Eval}(evk, f, \{\mathbf{ct}_1, \ldots, \mathbf{ct}_n\})) \neq f(x_1, \ldots, x_n)] \leq \mathsf{negl}(\lambda)$$

Where $(pk, evk, sk) = \mathsf{FHE.KeyGen}(1^\lambda)$ *and* $\mathbf{ct}_i = \mathsf{FHE.Enc}(pk, x_i)$*. Further, there exists a polynomial* $p(\lambda)$ *such that the output of* $\mathsf{FHE.Eval}(evk, f, \{\mathbf{ct}_1, \ldots, \mathbf{ct}_n\})$ *is at most* $p(\lambda)$ *bits long.*

3 Definition of Per-Party Private Server-Aided MPC

The following section defines our execution environment followed by a definition of our new paradigm through an ideal functionality. In this paradigm, the privacy of a client's input depends only on the servers chosen by that client and is not impacted by the other clients, aside from what can be learned through the output of the function. Further, we require that clients speak only to the servers they have chosen. While we allow clients to choose their own servers, we do require that any server that wishes to participate (and therefore is eligible to be chosen by a client) is part of a public registry $\mathcal{S}_{reg}$.

We refer to a single client as C_i and the set of n clients as $\mathcal{C}$. A single server is referred to as both S_k and $S_{i,j}$. We use the latter to identify the server by their client and their identity related to that client (i.e. $S_{i,j}$ is the jth server in client i's set $\mathcal{S}_{C_i}$). The former is used when referring to a server in a setting in which the client or clients who chose it are either not known or not relevant. The set of m servers is denoted by $\mathcal{S} \subseteq \mathcal{S}_{reg}$, where $\mathcal{S}_{reg}$ is the set of servers that can be chosen by a client, and the set of m_i servers chosen by a specific client C_i is referred to by $\mathcal{S}_{C_i}$.

When referring to a client or server that is known to be malicious, we use a $*$. For example, a malicious client and server would be referred to C_i^* and S_k^* respectively. Similarly, we use $*$ to mark a value that could have been maliciously modified. For example, a server S_k^* that receives a share $s_{i,j}$ could modify it. So when this server sends the share to another party, we denote it as $s_{i,j}^*$.

Real World Execution. We have a set of n clients $\mathcal{C}$ and a set of m servers $\mathcal{S}$. Each client $C_i \in \mathcal{C}$ picks its own set of m_i servers $\mathcal{S}_{C_i} \in \mathcal{S} \subseteq \mathcal{S}_{reg}$, such that $1 \leq i \leq n$ and $1 < m_i \leq m$. The servers in $\mathcal{S}_{C_i}$ are the *only* parties (meaning clients or servers) that C_i will communicate with. C_i also chooses its own threshold $1 \leq \tau_i < m_i$, the maximum number of corrupted servers tolerated in $\mathcal{S}_{C_i}$.

Let $\mathcal{A}$ be the adversary in the real world controlling a subset of the clients $\mathcal{C}_M \subset \mathcal{C}$ and servers $\mathcal{S}_M \subset \mathcal{S}$, where the clients in $\mathcal{C}_M$ and servers in $\mathcal{S}_M$ are chosen by the environment $\mathcal{Z}$.

Ideal World Execution. In the ideal world, we have an ideal functionality $\mathcal{F}_{\mathsf{Per\text{-}Party}}$ (Figure 3) for computing the function $f(x_1, ..., x_n) = (y_1, ..., y_n)$ with per-party privacy. Let $\mathcal{SIM}$ be the ideal adversary controlling a subset of clients $\mathcal{C}_M \subset \mathcal{C}$ and servers $\mathcal{S}_M \subset \mathcal{S}$, again with these subsets chosen by the environment $\mathcal{Z}$.

In this functionality, the ideal world adversary $\mathcal{SIM}$ notifies $\mathcal{F}_{\mathsf{Per\text{-}Party}}$ which clients and servers are corrupt. Then, each client C_i informs $\mathcal{F}_{\mathsf{Per\text{-}Party}}$ of which servers $\mathcal{S}_{C_i}$ they have chosen, and the threshold τ_i of corruption that they will tolerate. Next, there is an input phase where each client C_i sends their input x_i to the ideal functionality. For any client C_i

Functionality $\mathcal{F}_{\text{Per-Party}}$

This functionality has access to the public server registry $\mathcal{S}_{reg}$, and initializes the mapping of clients to servers $\textbf{Map} = \varnothing$

Set Up:

1. $\mathcal{SIM} \xrightarrow{(\texttt{CORRUPTIONS}, \mathcal{C}_M, \mathcal{S}_M)} \mathcal{F}_{\text{Per-Party}}$
2. $\mathcal{SIM} \xrightarrow{(\texttt{CLIENT-SET}, C_i^*, \mathcal{S}_{C_i^*}, \tau_i)} \mathcal{F}_{\text{Per-Party}}$ for each malicious client C_i^*
   2.1 Set $\textbf{Map}[(C_i^*, \tau_i)] = \mathcal{S}_{C_i^*}$
3. $C_i \xrightarrow{(\texttt{CLIENT-SET}, C_i, \mathcal{S}_{C_i}, \tau_i)} \mathcal{F}_{\text{Per-Party}}$ for each honest client C_i
   3.1 Set $\textbf{Map}[(C_i, \tau_i)] = \mathcal{S}_{C_i}$
4. If $\mathcal{S}_{C_i} \nsubseteq \mathcal{S}_{reg}$, for any C_i output $\perp$ and abort. Else continue
5. $\mathcal{F}_{\text{Per-Party}} \xrightarrow{(\texttt{MAPPING}, \textbf{Map})} \mathcal{SIM}$

Input:

1. $C_i \xrightarrow{(\texttt{CLIENT-INPUT}, C_i, x_i)} \mathcal{F}_{\text{Per-Party}}$ For each honest C_i
2. $\mathcal{F}_{\text{Per-Party}} \xrightarrow{((\texttt{CLIENT-INPUT}, C_i, x_i), ...)} \mathcal{SIM}$ For each C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$

Computation:

1. $\mathcal{SIM} \xrightarrow{\substack{(\texttt{RUN},(\texttt{CLIENT-INPUT},C_i^*,x_i),\\ ..., (\texttt{MAL-INPUT},C_j,x_i^*), ...)}} \mathcal{F}_{\text{Per-Party}}$ for each malicious C_i^* and honest C_j such that $|\mathcal{S}_{C_j} \cap \mathcal{S}_M| > \tau_j$, compute $f(x_1, ..., x_n) = (y_1, ..., y_n)$
2. Else if $\mathcal{SIM} \xrightarrow{(\texttt{RUN}, \textit{Abort})} \mathcal{F}_{\text{Per-Party}}$, output $\perp$ and abort

Output:

1. $\mathcal{F}_{\text{Per-Party}} \xrightarrow{((\texttt{CLIENT-OUTPUT}, C_i, y_i), ...)} \mathcal{SIM}$ for each C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$
2. $\mathcal{F}_{\text{Per-Party}} \xrightarrow{((\texttt{CLIENT-OUTPUT}, C_i^*, y_i), ...)} \mathcal{SIM}$ For each malicious C_i^*
3. $\mathcal{SIM} \xrightarrow{((\texttt{MAL-OUTPUT}, C_i, y_i^*), ...)} \mathcal{F}_{\text{Per-Party}}$ for each C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$
   3.1 Set $y_i = y_i^*$ for each y_i^* received
4. $\mathcal{F}_{\text{Per-Party}} \xrightarrow{(\texttt{CLIENT-OUTPUT}, C_i, y_i)} C_i$ for each honest C_i

Figure 3: Ideal Functionality for Per-Party Server-Aided MPC of Function $f(x_1, ..., x_n)$

who has chosen more malicious servers than their threshold can tolerate, $\mathcal{F}_{\text{Per-Party}}$ sends the client input x_i to $\mathcal{SIM}$, who replies with a potentially maliciously altered input x_i^* to be used on C_i's behalf during computation.

A similar process occurs in the output phase, where the output y_i of a client C_i who has chosen more than their threshold of corrupt servers is sent to $\mathcal{SIM}$ and replaced with a maliciously altered output y_i^*, which may equal $\perp$ if the corrupt servers decide to not provide output.

This is how $\mathcal{F}_{\text{Per-Party}}$ captures the per-party private paradigm. The execution of the protocol continues in the event that an honest client loses their input privacy, and, further, the adversary is now given control over that client's input and output. Specifically, if a client chooses too many malicious servers, the adversary not only learns the input and output of this client, but they may maliciously change the input and output, or refuse to provide input or output all together.

The formal description of $\mathcal{F}_{\text{Per-Party}}$ can be found in Figure 3.

4 Protocols for Per-Party Private Server-Aided MPC

In this section, we present our two protocols: $\Pi_{\text{Per-Party}}$ (Figure 5), where the servers use an $(m, m-1)$ UC-secure MPC protocol to compute the function and $\Pi^{FHE}_{\text{Per-Party}}$ (Figure 14), where the servers compute the function using FHE, and use $(m, m-1)$ UC-secure MPC to generate keys for FHE and decrypt the ciphertexts.

Before describing our protocols, we introduce the function $g(\mathsf{VSS}, \Sigma, \mathcal{I}_1, ..., \mathcal{I}_m, \mathbf{Map}) = (\mathcal{O}_1, ..., \mathcal{O}_m)$ (Figure 4) where $\mathcal{I}_j$ and $\mathcal{O}_j$ contain the input/output shares of server S_j respectively. g securely reconstructs shares, computes the function $f(x_1, ..., x_n) = (y_1, ..., y_n)$, and outputs shares to the servers.

4.1 Per-Party Private Server-Aided based on MPC

In our MPC-based protocol, clients first compute shares of their input according to VSS, then sign the shares along with their unique public identity and the unique public identity of the server to receive their shares. The shares, signatures, and C_i's verification key are then sent to the designated servers.

After this, the servers use a $(m,\ m-1)$ UC-secure MPC protocol Π^g (where m is the total number of servers participating) to compute the function $g(\mathsf{VSS}, \Sigma, \mathcal{I}_1, ..., \mathcal{I}_m, \mathbf{Map}) = (\mathcal{O}_1, ..., \mathcal{O}_m)$ (Figure 4). Each server provides their set of shares as input to the protocol and receives shares of the output after execution. The shares are then returned to the client.

We give the formal description of $\Pi_{\text{Per-Party}}$ in Figure 5. We remind the reader that we assume the existence of authenticated point-to-point channels between all clients and their chosen servers, and authenticated point-to-point channels and an authenticated broadcast channel between all servers.

4.2 Security of MPC-Based Protocol

We give our first main theorem in Theorem 1.

Function g

- **Input:** $\mathsf{VSS} = (\mathsf{Share}, \mathsf{Reconst})$, a verifiable secret sharing scheme. $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, a signature scheme. $\mathcal{I}_k = \{(s_{1,1}\sigma_{1,1}, pk^{\Sigma}_{1,1}), ..., (s_{n,m_n}, \sigma_{n,m_n}, pk^{\Sigma}_{n,m_n})\}$, the set of shares and signatures where $(s_{i,j}, \sigma_{i,j}, pk^{\Sigma}_{i})$ is held by S_k iff $S_k = S_{i,j} \in \mathcal{S}_{C_i}$. **Map** the mapping of clients to servers
- **Output:** $\mathcal{O}_k = \{s'_{1,1}, ...s'_{n,m_n}\}$, the set of shares where $s'_{i,j}$ is held by S_k iff $S_k = S_{i,j} \in \mathcal{S}_{C_i}$

1. For $1 \leq i \leq n$
   - 1.1 Let pk^{Σ}_{i} be the verification key submitted by more than τ_i of the servers in $\mathcal{S}_{C_i}$
     - 1.1.1 If no such set exists, set $x_i = 0$ and skip to next iteration
   - 1.2 Set $R_i = \varnothing$
   - 1.3 If $\Sigma.\mathsf{Verify}(pk^{\Sigma}_{i}, \sigma_{i,j}, (s_{i,j}, C_i, S_{i,j}) = 1$, set $R_i = R_i \cup s_{i,j}$ for all $j \in [m_i]$
   - 1.4 If $|R_i| \leq \tau_i$, set $x_i = 0$
   - 1.5 Else set $x_i = \mathsf{VSS.Reconst}(R_i)$
2. $(y_1, ..., y_n) = f(x_1, ..., x_n)$
3. For $1 \leq i \leq n$
   - 3.1 $(s'_{i,1}, \ldots, s'_{i,m_i}) = \mathsf{VSS.Share}(y_i, \tau_i, m_i)$
4. For $1 \leq k \leq m$
   - 4.1 $\mathcal{O}_k = (s'_{i,j}, ...)$ for i, j such that $S_k = S_{i,j} \in \mathcal{S}_{C_i}$

Figure 4: g The Function for Computing f on shares

Protocol $\Pi_{\text{Per-Party}}$

Inputs: x_i for each client C_i, $i \in [1, ..., n]$. Π^g, a $(m, m-1)$ UC-secure MPC protocol for ideal functionality $\mathcal{F}_g$. $\mathsf{VSS} = (\mathsf{Share}, \mathsf{Reconst})$, a verifiable secret sharing scheme. $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, a signature scheme.
Outputs: y_i for each client C_i, $i \in [1, ..., n]$

1. **Set Up:** For each client C_i
   - 1.1 Choose a set of servers $\mathcal{S}_{C_i}$ from registry $\mathcal{S}_{reg}$
   - 1.2 Choose threshold $\tau_i < |\mathcal{S}_{C_i}|$
   - 1.3 Compute $(pk_i^\Sigma, sk_i^\Sigma) = \Sigma.\mathsf{Gen}(1^\lambda)$
   - 1.4 Announce $(C_i, \mathcal{S}_{C_i}, \tau_i)$
2. **Input Phase:** For each client C_i
   - 2.1 Compute $\mathsf{VSS.Share}(x_i, \tau_i, m_i) = (s_{i,1}, ..., s_{i,m_i})$
   - 2.2 For each $j \in [1, \ldots, m_i]$, compute $\sigma_{i,j} = \Sigma.\mathsf{Sign}(sk_i^\Sigma, (s_{i,j}, C_i, S_{i,j}))$
   - 2.3 Send $(s_{i,j}, \sigma_{i,j}, pk_i^\Sigma)$ to each $S_{i,j} \in \mathcal{S}_{C_i}$
   - 2.4 For each server $S_{i,j} \in \mathcal{S}_{C_i}$, if $\Sigma.\mathsf{Verify}(pk_i^\Sigma, (s_{i,j}, C_i, S_{i,j})) = 0$ abort, else continue
3. **Computation Phase:** Servers cooperatively execute Π^g
   - 3.1 **Input Phase:** Let $\mathcal{I}_k = \{(s_{i,j}, \sigma_{i,j}, pk_i^\Sigma)\}_{S_k = S_{i,j} \in \mathcal{S}_{C_i}}$
     Each server S_k provides $\mathcal{I}_k$ as input to Π^g.
   - 3.2 **Computation Phase:** $\mathcal{S}$ run Π^g to compute $g(\mathsf{VSS}, \Sigma, \mathcal{I}_1, ..., \mathcal{I}_m, \mathbf{Map}) = (\mathcal{O}_1, ..., \mathcal{O}_m)$ (4). If g aborts, then Π^g aborts and returns the same thing that g did.
   - 3.3 **Output Phase:** S_k receives $\mathcal{O}_k = \{s'_{i,j}\}_{S_k = S_{i,j} \in \mathcal{S}_{C_i}}$ as output from Π^g
4. **Output Phase:** Each S_k parses $\mathcal{O}_k = \{s'_{i,j}\}_{S_k = S_{i,j} \in \mathcal{S}_{C_i}}$ and sends $s'_{i,j}$ to C_i
   For each C_i, if $\mathsf{VSS.Reconst}(s'_{i,1}, \ldots, s'_{i,m_i}) = y_i$ fails, output $\perp$ and abort. Else output y_i

Figure 5: Per-Party Private MPC Protocol $\Pi_{\text{Per-Party}}$

Algorithm $\mathcal{SIM}$ - Set Up

1. $C_i^* \xrightarrow{(\texttt{CLIENT-SET}, C_i^*, \mathcal{S}_{C_i^*}, \tau_i)} \mathcal{SIM}$ for malicious C_i^*
2. $\mathcal{SIM} \xrightarrow{(\texttt{CORRUPTIONS}, \mathcal{C}_M, \mathcal{S}_M)} \mathcal{F}_{\text{Per-Party}}$
3. $\mathcal{SIM} \xrightarrow{(\texttt{CLIENT-SET}, C_i^*, \mathcal{S}_{C_i^*}, \tau_i)} \mathcal{F}_{\text{Per-Party}}$ for each malicious C_i^*
4. $\mathcal{F}_{\text{Per-Party}} \xrightarrow{(\texttt{MAPPING}, \textbf{Map})} \mathcal{SIM}$
5. For each honest C_i
   5.1 Compute $(pk_i^\Sigma, sk_i^\Sigma) = \Sigma.\text{Gen}(1^\lambda)$

Figure 6: Set Up Phase of $\mathcal{SIM}$

Theorem 1 *If Π^g is an $(m, m-1)$ UC-secure MPC protocol for function g (Figure 4),* VSS *is an information theoretic private (τ_i, m_i)-threshold verified secret sharing scheme for $1 \leq \tau_i < m_i$ and $1 \leq i \leq n$, and $\Sigma = (\text{Gen}, \text{Sign}, \text{Verify})$ is an EUF-CMA signature scheme, where n is the number of clients and m_i is the number of servers chosen by client C_i then Protocol $\Pi_{\text{Per-Party}}$ (Figure 5) UC-realizes $\mathcal{F}_{\text{Per-Party}}$ (Figure 3).*

Correctness Our proof of correctness is straightforward, based on the correctness of VSS (Definition 2) and the correctness of the signature scheme (Definition 3), and the fact that Π^g must also be correct (Definition 1).

Simulator In this section, we define our simulator $\mathcal{SIM}$. $\mathcal{SIM}$ can be seen in Figures 6, 7, 8, and 9. We divide the simulator into 4 algorithms, one for each phase of the protocol, for readability purposes only. These algorithms are to be run consecutively.

In the setup phase (Figure 6), $\mathcal{SIM}$ simply notifies the ideal functionality $\mathcal{F}_{\text{Per-Party}}$ of which clients and servers are corrupt, and the set of servers chosen by each corrupt client by using the $(\texttt{CORRUPTIONS}, \mathcal{C}_M, \mathcal{S}_M)$ and $(\texttt{CLIENT-SET}, C_i^*, \mathcal{S}_{C_i^*}, \tau_i)$ commands respectively.

During the input phase (Figure 7), $\mathcal{SIM}$ receives the input of honest clients who chose too many corrupt servers. $\mathcal{SIM}$ then simulates the sharing of input for the malicious servers. If $\mathcal{SIM}$ knows the input x_i being shared it computes an honest share. Otherwise, $\mathcal{SIM}$ sends shares of zero. During this phase, $\mathcal{SIM}$ also learns shares sent by a malicious client to honest servers.

$\mathcal{SIM}$ makes use of $\mathcal{SIM}_{\Pi^g}$, the simulator for Π^g, to learn the remaining shares of the malicious clients during the computation phase (Figure 8). $\mathcal{SIM}$ plays the role of $\mathcal{F}_g$, the ideal functionality for computing the function g (Figure 4), for $\mathcal{SIM}_{\Pi^g}$. In doing so, $\mathcal{SIM}$ learns all shares held by the malicious servers. This means $\mathcal{SIM}$ learns the shares sent by a malicious client C_i^* to the honest servers in $\mathcal{S}_{C_i^*}$ during the input phase (Figure 7) and learns the shares sent to the malicious servers in $\mathcal{S}_{C_i^*}$ during the computation phase (Figure 8).

Algorithm $\mathcal{SIM}$ - Input

1. For each C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$

   1.1 $\mathcal{F}_{\text{Per-Party}} \xrightarrow{(\texttt{CLIENT-INPUT}, C_i, x_i)} \mathcal{SIM}$

2. For honest C_i and honest S_k

   2.1 $\mathcal{SIM}$ sends and receives nothing

3. For malicious C_i^* and honest S_k

   3.1 $C_i^* \xrightarrow{(s_{i,j}, \sigma_{i,j}, pk_i^\Sigma)} \mathcal{SIM}$ such that $S_k = S_{i,j} \in \mathcal{S}_{C_i}$

4. For honest C_i and malicious S_k^*

   4.1 If $\mathcal{SIM}$ knows x_i

   4.1.1 $\mathcal{SIM}$ computes $\text{VSS.Share}(x_i, \tau_i, m_i) = (s_{i,1}, \ldots, s_{i,m_i})$

   4.1.2 $\mathcal{SIM}$ computes $\sigma_{i,j} = \Sigma.\text{Sign}(sk_i^\Sigma, (s_{i,j}, C_i, S_{i,j}))$

   4.1.3 $\mathcal{SIM} \xrightarrow{(s_{i,j}, \sigma_{i,j}, pk_i^\Sigma)} S_k^*$ such that $S_k^* = S_{i,j} \in \mathcal{S}_{C_i}$

   4.2 Else if $\mathcal{SIM}$ does not know x_i

   4.2.1 $\mathcal{SIM}$ computes $\text{VSS.Share}(0, \tau_i, m_i) = (\widetilde{s}_{i,1}, \ldots, \widetilde{s}_{i,m_i})$

   4.2.2 $\mathcal{SIM}$ computes $\widetilde{\sigma_{i,j}} = \Sigma.\text{Sign}(sk_i^\Sigma, (s_{i,j}, C_i, S_{i,j}))$

   4.2.3 $\mathcal{SIM} \xrightarrow{(\widetilde{s}_{i,j}, \widetilde{\sigma_{i,j}}, pk_i^\Sigma)} S_k^*$ such that $S_k^* = S_{i,j} \in \mathcal{S}_{C_i}$

5. For malicious C_i^* and malicious S_k^*

   5.1 $\mathcal{SIM}$ sends and receives nothing

Figure 7: Input Phase of $\mathcal{SIM}$

$\mathcal{SIM}$ will then attempt to reconstruct C_i^*'s input and provide it to $\mathcal{F}_{\text{Per-Party}}$.

Further, during the computation phase (Figure 8) $\mathcal{SIM}$ learns the shares held by malicious servers chosen by the honest clients. The shares sent to these malicious servers were initially computed by $\mathcal{SIM}$, but could have been modified by the malicious servers since. If an honest client C_i has chosen too many corrupt servers, $\mathcal{SIM}$ uses these shares to attempt to reconstruct the potentially maliciously altered input x_i^* that is used on behalf of C_i in computation and sends x_i^* to $\mathcal{F}_{\text{Per-Party}}$.

In both cases above, signatures on the shares are verified, and only shares with verified signatures are used to reconstruct. If there are not enough verified shares (that is, τ_i or less) then the input is set to 0. Else, if reconstruction fails, $\mathcal{SIM}$ outputs *ReconstAbort* and aborts.

Finally, during the output phase (Figure 9), $\mathcal{SIM}$ finishes the simulation of Π^g for $\mathcal{SIM}_{\Pi^g}$. This involves computing honest shares when the output of a client C_i is known, as is the case when C_i chose too many malicious servers or C_i themself is malicious, or shares of zero when the output is unknown.

For an honest client, C_i who has chosen more than τ_i malicious servers, $\mathcal{SIM}$ intercepts the shares these malicious servers attempt to send to C_i. After intercepting these shares, $\mathcal{SIM}$ reconstructs the maliciously altered output y_i^* and sends it to $\mathcal{F}_{\text{Per-Party}}$ as the output that C_i is to receive. If reconstruction fails in this case, we take this as the malicious servers refusing to provide output. Thus, the client receives $\perp$ as output.

Simulator Runtime Next, we will prove that $\mathcal{SIM}$ runs in probabilistic polynomial time (PPT).

In the set up phase (Figure 6), $\mathcal{SIM}$ simply sends the set of servers for each corrupt client (determined by the environment) to the ideal functionality $\mathcal{F}_{\text{Per-Party}}$, receives the mapping **Map** from $\mathcal{F}_{\text{Per-Party}}$, then sends the set of corrupt clients $\mathcal{C}_M$ and servers $\mathcal{S}_M$ to the ideal functionality $\mathcal{F}_{\text{Per-Party}}$. Lastly, the simulator generates key pairs for the signature scheme Σ for all honest clients. This can all be done in polynomial time.

In the input phase (Figure 7), aside from sending messages, $\mathcal{SIM}$ computes VSS.Share and Σ.Sign for potentially every client. Both VSS.Share and Σ.Sign are polynomial time algorithms.

Then, in the computation phase (Figure 8), $\mathcal{SIM}$ makes use of $\mathcal{SIM}_{\Pi^g}$, which is a PPT simulator for the UC-secure MPC protocol, and forwards messages on behalf of this simulator. Aside from the messages sent, $\mathcal{SIM}$ runs Σ.Verify and VSS.Reconst. Both of these are polynomial time algorithms.

Lastly, in the output phase (Figure 9), $\mathcal{SIM}$ continues to make use of $\mathcal{SIM}_{\Pi^g}$, which is a PPT simulator for the UC-secure MPC protocol, and forwards messages on behalf of this simulator. Aside from the messages sent, $\mathcal{SIM}$ computes VSS.Share, which as we have stated is a polynomial time algorithm.

Since each phase can be computed in polynomial time, we know that $\mathcal{SIM}$ runs in probabilistic polynomial time.

Indistinguishability Finally, we prove that the real and ideal worlds are indistinguishable through a series of hybrids. We will start from the real-world protocol $\Pi_{\text{Per-Party}}$ and build hybrids, proving indistinguishability between each, until we reach the ideal world.

Consider the following hybrids:

Algorithm $\mathcal{SIM}$ - Computation

1. $\mathcal{F}_{\text{Per-Party}} \xrightarrow{(\texttt{RUN})} \mathcal{SIM}$
2. Set ABORT $=$ *False*
3. $\mathcal{SIM}$ activates $\mathcal{SIM}_{\Pi^g}$, the simulator for Π^g
4. $\mathcal{SIM} \xrightarrow{(\mathcal{S}_M)} \mathcal{SIM}_{\Pi^g}$
5. $\mathcal{SIM}_{\Pi^g}$ simulates Π^g
   - 5.1 For any message ρ sent by $S_k^* \in \mathcal{S}_M$ to honest S_j, $S_k^* \xrightarrow{(\rho)} \mathcal{SIM} \xrightarrow{(\rho)} \mathcal{SIM}_{\Pi^g}$
   - 5.2 For any message α sent by honest S_j to $S_k^* \in \mathcal{S}_M$, $\mathcal{SIM}_{\Pi^g} \xrightarrow{(\alpha)} \mathcal{SIM} \xrightarrow{(\alpha)} S_k^*$
6. For corrupt $S_k^* \in \mathcal{S}_M$, $\mathcal{SIM}$ intercepts messages intended for $\mathcal{F}_g$, $\mathcal{SIM}_{\Pi^g} \xrightarrow{(\texttt{SERVER-INPUT}, S_k^*, \mathcal{I}_k)} \mathcal{SIM}$
7. For honest C_i and corrupt $S_{i,j}^*$ such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| \leq \tau_i$
   - 7.1 For all $(s_{i,j}^*, \sigma_{i,j}^*, pk_i^\Sigma) \in \mathcal{I}_k$, if $s_{i,j}^* \neq s_{i,j}$ and $\Sigma.\text{Verify}(pk_i^\Sigma, \sigma_{i,j}^*, (s_{i,j}^*, C_i, S_{i,j}^*)) = 1$ output *SigFail* and abort
8. For corrupt C_i^* and all $S_{i,j}^* \in \mathcal{S}_{C_i^*}$, let pk_i^Σ be the key submitted by more than τ_i of the servers in $\mathcal{S}_{C_i^*}$ and set $R_i = \varnothing$
   - 8.1 If no such set exists, set $x_i^* = 0$ and skip to next iteration
   - 8.2 If $\Sigma.\text{Verify}(pk_i^\Sigma, (s_{i,j}, C_i^*, S_{i,j}^*)) = 1$, set $R_i = R_i \cup s_{i,j}$
   - 8.3 If $|R_i| \leq \tau_i$, set $x_i = 0$
   - 8.4 Else compute $x_i^* = \text{VSS.Reconst}(R_i)$
     - 8.4.1 If $\text{VSS.Reconst}(R_i)$ fails, set ABORT $=$ *True* and go to 10
9. For honest C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$, let pk_i^Σ be the key submitted by more than τ_i of the servers in $\mathcal{S}_{C_i}$ and set $R_i = \varnothing$
   - 9.1 If no such set exists, set $x_i^* = 0$ and skip to next iteration
   - 9.2 If $\Sigma.\text{Verify}(pk_i^\Sigma, (s_{i,j}, C_i, S_{i,j})) = 1$, set $R_i = R_i \cup s_{i,j}$
   - 9.3 If $|R_i| \leq \tau_i$, set $x_i = 0$
   - 9.4 Else compute $x_i^* = \text{VSS.Reconst}(R_i)$
     - 9.4.1 If $\text{VSS.Reconst}(R_i)$ fails, set ABORT $=$ *True* and go to 10
10. If ABORT $=$ *False*, $\mathcal{SIM} \xrightarrow{\substack{(\texttt{RUN}, (\texttt{CLIENT-INPUT}, C_i^*, x_i), \ldots, \\ (\texttt{MAL-INPUT}, C_j, x_j^*), \ldots)}} \mathcal{F}_{\text{Per-Party}}$ for each malicious C_i^* and honest C_j such that $|\mathcal{S}_{C_j} \cap \mathcal{S}_M| > \tau_j$
11. Else, $\mathcal{SIM} \xrightarrow{(\texttt{RUN}, \textit{Abort})} \mathcal{F}_{\text{Per-Party}}$, output *ReconstAbort* and abort

Figure 8: Computation Phase of $\mathcal{SIM}$

Algorithm $\mathcal{SIM}$ - Output

1. $\mathcal{F}_{\text{Per-Party}} \xrightarrow{((\texttt{CLIENT-OUTPUT}, C_i, y_i), \ldots)} \mathcal{SIM}$ For each C_i such that $C_i \notin \mathcal{C}_M$ and $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$
2. $\mathcal{F}_{\text{Per-Party}} \xrightarrow{((\texttt{CLIENT-OUTPUT}, C_i^*, y_i), \ldots)} \mathcal{SIM}$ for each malicious C_i^*
3. For all $S_k^* \in \mathcal{S}_M$
   - 3.1 For all i, j such that $S_k^* = S_{i,j}^* \in \mathcal{S}_{C_i}$
     - 3.1.1 If $\mathcal{SIM}$ knows y_i
       - 3.1.1.1 $\mathcal{SIM}$ computes $\mathsf{VSS.Share}(y_i, \tau_i, m_i) = (s'_{i,1}, \ldots, s'_{i,m_i})$ if not already computed
       - 3.1.1.2 Set $\mathcal{O}_k = \mathcal{O}_k \cup \{s'_{i,j}\}$
     - 3.1.2 Else
       - 3.1.2.1 $\mathcal{SIM}$ computes $\mathsf{VSS.Share}(0, \tau_i, m_i) = (\widetilde{s}'_{i,1}, \ldots, \widetilde{s}_{i,m_i})$ if not already computed
       - 3.1.2.2 Set $\mathcal{O}_k = \mathcal{O}_k \cup \{\widetilde{s}'_{i,j}\}$
4. $\mathcal{SIM} \xrightarrow{((\texttt{SERVER-OUTPUT}, S_k^*, \mathcal{O}_k), \ldots)} \mathcal{SIM}_{\Pi^g}$ for each malicious S_k^*
5. $\mathcal{SIM}_{\Pi^g}$ continues to simulate Π^g
   - 5.1 For any message ρ' sent by $S_k^* \in \mathcal{S}_M$ to honest S_j
     - 5.1.1 $S_k^* \xrightarrow{(\rho')} \mathcal{SIM} \xrightarrow{(\rho')} \mathcal{SIM}_{\Pi^g}$
   - 5.2 For any message α' sent by honest S_j to $S_k^* \in \mathcal{S}_M$
     - 5.2.1 $\mathcal{SIM}_{\Pi^g} \xrightarrow{(\alpha')} \mathcal{SIM} \xrightarrow{(\alpha')} S_k^*$
6. $\mathcal{SIM}_{\Pi^g}$ ends the simulation of Π^g
7. For each $S_k^* \in \mathcal{S}_M$ such that $S_k^* = S_{i,j}^*$ for honest C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$
   - 7.1 $S_k^* \xrightarrow{(s_{i,j}^*)} \mathcal{SIM}$
   - 7.2 If $\mathsf{VSS.Reconst}(s_{i,0}^*, \ldots, s_{i,\ell}^*)$ succeeds
     - 7.2.1 Compute $\mathsf{VSS.Reconst}(s_{i,0}^*, \ldots, s_{i,\ell}^*) = y_i^*$ such that $\tau_i < \ell = |\mathcal{S}_{C_i} \cap \mathcal{S}_M|$
   - 7.3 Else
     - 7.3.1 Set $y_i^* = \perp$
8. $\mathcal{SIM} \xrightarrow{((\texttt{MAL-OUTPUT}, C_i, y_i^*), \ldots)} \mathcal{F}_{\text{Per-Party}}$ for each honest C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$
9. For malicious C_i^* and honest $S_k = S_{i,j} \in \mathcal{S}_{C_i^*}$
   - 9.1 $\mathcal{SIM}$ computes $\mathsf{VSS.Share}(y_i, \tau_i, m_i) = (s'_{i,1}, \ldots, s'_{i,m_i})$ if not already computed
   - 9.2 $\mathcal{SIM} \xrightarrow{(s'_{i,j})} C_i^*$

Figure 9: Output Phase of $\mathcal{SIM}$

- $\mathbf{Hyb}_0$: This is the real world execution of the protocol $\Pi_{\mathsf{Per\text{-}Party}}$
- $\mathbf{Hyb}_1$: This is the same as $\mathbf{Hyb}_0$, except that the adversary outputs *ReconstAbort* and aborts if the reconstruction of input shares fails
- $\mathbf{Hyb}_2$: This is the same as $\mathbf{Hyb}_1$, except that the adversary outputs *SigFail* if malicious servers submit a signature that was not computed by the adversary, but still verifies.
- $\mathbf{Hyb}_3$: This is the same as $\mathbf{Hyb}_2$, except instead of the servers $\mathcal{S}$ computing Π^g, it is simulated by $\mathcal{SIM}_{\Pi^g}$
- $\mathbf{Hyb}_4$: This is the same as $\mathbf{Hyb}_3$, except instead of honest clients sending shares of their input x_i, they send shares of 0 and commitments to these shares

Lemma 1 *If* $\mathsf{VSS} = (\mathsf{Share}, \mathsf{Reconst})$ *is a verifiable secret sharing scheme, then* $\mathbf{Hyb}_1$ *is indistinguishable from* $\mathbf{Hyb}_0$

Proof In $\mathbf{Hyb}_1$, the signatures on shares are verified, and only those that pass verification are used to reconstruct. If there are less than $\tau_i + 1$ shares that pass, x_i is set to be 0. Therefore, the only way $\mathbf{Hyb}_1$ aborts is if there are $\tau_i + 1$ shares that do not reconstruct to any value, but have signatures that verify. However, by the committed property of verifiable secret sharing (Definition 2), we know that a set of $\tau_i + 1$ shares must reconstruct to some value. Therefore, $\mathbf{Hyb}_1$ aborts with *ReconstAbort* with negligible probability. ■

Lemma 2 *If* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is an EUF-CMA signature scheme,* $\mathbf{Hyb}_2$ *is indistinguishable from* $\mathbf{Hyb}_1$

Proof Proceed by contradiction. That is, assume that there exists an adversary $\mathcal{A}$ that can provide a transcript allowing the environment $\mathcal{Z}$ to distinguish between $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ with non-negligible probability. We can then construct a reduction $\mathcal{B}$ to the unforgeability of signatures. We define $\mathcal{B}$ in Figure 10.

The only difference between $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$ is that the simulator aborts with *SigFail* if a malicious server chosen by an honest client submits a signature that verifies on a share not computed by the client. Since $\mathcal{A}$ can distinguish between $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_1$, we know that the probability that $\mathcal{A}$ submits $s^*_{i,j} \neq s_{i,j}$ such that $\Sigma.\mathsf{Verify}(pk^\Sigma, (s^*_{i,j}, C_i, S^*_{i,j})) = 1$ is non negligible. Therefore, $\mathcal{B}$ submits a forgery with the same non-negligible probability. ■

Lemma 3 *If* Π^g *is a UC-secure MPC protocol,* $\mathbf{Hyb}_3$ *is indistinguishable from* $\mathbf{Hyb}_2$

Proof Proceed by contradiction. That is, assume that there exists an environment $\mathcal{Z}$ that can distinguish between $\mathbf{Hyb}_3$ and $\mathbf{Hyb}_2$ with some non-negligible probability $p(\lambda)$. We can then use $\mathcal{Z}$ to construct a reduction $\mathcal{Z}_{\Pi^g}$, which is an environment with the goal of distinguishing between the real and ideal execution of Π^g. $\mathcal{Z}_{\Pi^g}$ will act as the adversary for $\mathcal{Z}$, using the transcript received from the challenger as the execution of Π^g.

$\mathcal{Z}_{\Pi^g}$ controls an adversary $\mathcal{B}$ that has corrupted some subset of the servers executing Π^g in either the real or the ideal world and let these worlds be Π^g and Π^g_{IDEAL} respectively. In order to properly simulate the hybrid worlds for $\mathcal{Z}$, $\mathcal{Z}_{\Pi^g}$ will play the role of the clients. We define $\mathcal{Z}_{\Pi^g}$ in Figure 11.

There are two possible cases here: either $\mathcal{Z}_{\Pi^g}$ is in the real world (i.e. Π^g) and holds or $\mathcal{Z}_{\Pi^g}$ is in the ideal world (i.e. Π^g_{IDEAL}).

$\mathcal{B}(\lambda)$:

1. Receive pk^{Σ} from the challenger
2. Activate $\mathcal{A}$
3. Send $\mathbf{Map}, \mathcal{C}_M, \mathcal{S}_M$ to $\mathcal{A}$
4. Send x_i for $C_i \in \mathcal{C}_M$ to $\mathcal{A}$
5. Simulate as in $\mathbf{Hyb}_1$, using pk^{Σ} as the verification key of some $C_i \notin \mathcal{S}_M$ that has chosen corrupt servers
6. In the input phase, query the oracle with $(s_{i,j}, C_i, S^*_{i,j})$ to receive $\sigma_{i,j}$ for all $S^*_{i,j} \in \mathcal{S}_{C_i}$
7. Continue simulating $\mathbf{Hyb}_1$
8. In the computation phase, upon receipt of $\mathcal{I}_k$ for all $S^*_k \in \mathcal{S}_{C_i}$, for $(s^*_{i,j}, \sigma^*_{i,j}, pk^{\Sigma}) \in \mathcal{I}_k$
   8.1 If $s^*_{i,j} \neq s_{i,j}$ and $\Sigma.\mathsf{Verify}(pk^{\Sigma}, (s^*_{i,j}, C_i, S^*_{i,j})) = 1$, submit $(s^*_{i,j}, C_i, S^*_{i,j})$ to the challenger as a forgery
9. Abort

Figure 10: $\mathcal{B}$ An Adversary for Computing Forged Signatures against Σ

- **Case Π^g:**

  In this case, $\mathcal{Z}_{\Pi^g}$ perfectly emulates $\mathbf{Hyb}_2$, as in line 9 of Figure 11 the execution of Π^g happens in the real world as expected. Recall that the execution of Π^g is the only difference between $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_3$. In all other steps, $\mathcal{Z}_{\Pi^g}$ emulates $\mathbf{Hyb}_2$ by sharing according to the protocol $\Pi_{\mathsf{Per\text{-}Party}}$, and forwarding all messages expected by $\mathcal{Z}$. So $\mathcal{Z}_{\Pi^g}$ outputs 1 with the same probability as $\mathcal{Z}$, therefore

$$Pr[\mathcal{Z}_{\Pi^g}(\lambda) \to 1 | \Pi^g] = Pr[\mathcal{Z}(\lambda) \to 1 | \mathbf{Hyb}_2]$$

- **Case Π^g_{IDEAL}:**

  In this case, $\mathcal{Z}_{\Pi^g}$ perfectly emulates $\mathbf{Hyb}_3$, as in line 9 of Figure 11 the execution of Π^g is simulated by $\mathcal{SIM}_{\Pi^g}$ as expected in $\mathbf{Hyb}_3$. In all other steps, $\mathcal{Z}_{\Pi^g}$ emulates $\mathbf{Hyb}_3$ by sharing according to the protocol $\Pi_{\mathsf{Per\text{-}Party}}$, and forwarding all messages expected by $\mathcal{Z}$. So $\mathcal{Z}_{\Pi^g}$ outputs 1 with the same probability as $\mathcal{Z}$, therefore

$$Pr[\mathcal{Z}_{\Pi^g}(\lambda) \to 1 | \Pi^g_{IDEAL}] = Pr[\mathcal{Z}(\lambda) \to 1 | \mathbf{Hyb}_3]$$

In both cases we see that $\mathcal{Z}_{\Pi^g}$ outputs 1 with the same probability as $\mathcal{Z}$, which gives us

$$|Pr[\mathcal{Z}_{\Pi^g}(\lambda) \to 1 | \Pi^g] - Pr[\mathcal{Z}_{\Pi^g}(\lambda) \to 1 | \Pi^g_{IDEAL}]|$$

$\underline{\mathcal{Z}_{\Pi^g}(\lambda)}$

1. Activate $\mathcal{Z}$.
2. Receive **Map**, $\mathcal{C}_M$, $\mathcal{S}_M$, $x_1, ..., x_n$ and $(\tau_1, \ldots, \tau_n)$ from $\mathcal{Z}$
3. For each client $C_i^* \in \mathcal{C}_M$, send the information $\mathcal{Z}$ expects after a client is corrupted
4. For each server $S_k^* \in \mathcal{S}_M$, instruct $\mathcal{B}$ to corrupt S_k^* and forward the information $\mathcal{Z}$ expects to receive after a server is corrupted
5. For each client C_i, share x_i and sign honestly
6. For each server S_k forward the share/s intended for the server to $\mathcal{Z}$
7. For each server S_k, construct $\mathcal{I}_k$ according to the mapping
8. Provide each $\mathcal{I}_k$ to S_k as input to the execution of Π^g
9. Instruct $\mathcal{B}$ to execute Π^g, interacting with $\mathcal{Z}$ as needed
10. Receive $\mathcal{O}_k$ as output from Π^g and forward to $\mathcal{Z}$ for corrupt servers
11. For each client C_i, forward the expected shares to $\mathcal{Z}$ as if they were sent by the servers in $\mathcal{S}_{C_i}$
12. Output whatever $\mathcal{Z}$ outputs

Figure 11: $\mathcal{Z}_{\Pi^g}$ A Distinguishing Environment for UC-Secure MPC Protocol Π^g

$$= |Pr[\mathcal{Z}(\lambda) \to 1|\mathbf{Hyb}_2] - Pr[\mathcal{Z}(\lambda) \to 1|\mathbf{Hyb}_3]| = p(\lambda)$$

This is a contradiction, as we have found an environment that can distinguish between Π^g and Π^g_{IDEAL} with non-negligible probability $p(\lambda)$, but Π^g was assumed to be UC-secure.

Therefore, $\mathbf{Hyb}_3$ is indistinguishable from $\mathbf{Hyb}_2$. ■

Lemma 4 *If* $\mathsf{VSS} = (\mathsf{Share}, \mathsf{Reconst})$ *is an information theoretic private verifiable secret sharing scheme,* $\mathbf{Hyb}_4$ *is indistinguishable from* $\mathbf{Hyb}_3$

Proof The proof of Lemma 4 follows from the privacy property of VSS (Definition 2). Because the VSS scheme that we use is information-theoretic private, even an adversary with unlimited computational power cannot distinguish between a share of 0 and a share x_i. Therefore, we know that $\mathbf{Hyb}_4$ is indistinguishable from $\mathbf{Hyb}_3$. ■

4.3 Per-Party Private Server-aided MPC from FHE

Here we present a second protocol based on fully homomorphic encryption. This protocol is very similar to $\Pi_{\mathsf{Per\text{-}Party}}$, except that the general MPC protocol run by the servers is replaced by a slightly more specific instantiation based on FHE. We provide a full formal treatment here for completeness.

In this protocol, the clients' procedure is the same. The servers use an MPC protocol Π^{KG} to run the key generation algorithm of an FHE scheme [BV14]. Each server contributes randomness to the key generation algorithm and receives the public key pk, the evaluation key evk, and a share of the secret key sk_k for S_k as output. The servers then encrypt the shares and signatures they received from their clients and send the ciphertexts to one computing server. Without loss of generality, let this server be S_1.

The computing server homomorphically evaluates the function $g(\mathsf{VSS}, \Sigma, \mathcal{I}_1, \ldots, \mathcal{I}_m, \mathbf{Map})$ (Figure 4), computes a SNARG π_{out} proving correct output given the input ciphertexts, and returns the ciphertexts of output shares and proof to the respective servers. Finally, the servers use another MPC protocol Π^{Dec} to run the decryption algorithm of the FHE scheme on the output ciphertexts to obtain their output shares and return these shares to their clients. Π^{Dec} takes as input the set of ciphertexts from the computing server S_1, along with the proof of correct computation, and every server's share of the secret key sk_k. Π^{Dec} then verifies the proof π_{out} and decrypts the output ciphertexts. Each server then receives their respective share of their client's output.

Before giving the formal description of our protocol, we introduce the ideal functionalities $\mathcal{F}_{\mathsf{KG}}$ (Figure 12) and $\mathcal{F}_{\mathsf{Dec}}$ (Figure 13). Let Π^{KG} and Π^{Dec} be two $(m, m-1)$ UC-secure MPC protocols realizing each functionality respectively. Note that we assume for an FHE scheme, given the public key pk and secret key sk, it is easy to verify that $(pk, sk) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$.

We give the formal description of $\Pi^{FHE}_{\mathsf{Per\text{-}Party}}$ in Figure 14.

4.4 Security of FHE-Based Protocol

Here we present our second main theorem, Theorem 2.

Functionality $\mathcal{F}_{\mathsf{KG}}$

This functionality is parameterized by $\mathsf{FHE.KeyGen}(\cdot)$

KeyGen

1. For honest party P_i
   - 1.1 $P_i \xrightarrow{(\texttt{KG-RAND}, P_i, r_i)} \mathcal{F}_{\mathsf{KG}}$
   - 1.2 $\mathcal{F}_{\mathsf{KG}} \xrightarrow{(\texttt{KG-RAND}, P_i)} \mathcal{SIM}$
2. For corrupt party P_i^*
   - 2.1 $\mathcal{SIM} \xrightarrow{(\texttt{KG-RAND}, P_i^*, r_i)} \mathcal{F}_{\mathsf{KG}}$
3. Compute $(pk, evk, sk) = \mathsf{FHE.KeyGen}(1^\lambda \; ; \; r_1 \oplus \ldots \oplus r_m)$
4. Choose $sk_1, \ldots, sk_{m-1} \xleftarrow{\$} \mathbb{K}$ for key space $\mathbb{K}$
5. Set $sk_m = sk \oplus sk_1 \oplus \ldots sk_{m-1}$
6. For honest party P_i
   - 6.1 $\mathcal{F}_{\mathsf{KG}} \xrightarrow{(\texttt{KEY-SHARE}, P_i, pk, evk, sk_i)} P_i$
   - 6.2 $\mathcal{F}_{\mathsf{KG}} \xrightarrow{(\texttt{KEY-SHARE}, P_i, pk, evk)} \mathcal{SIM}$
7. For corrupt party P_i^*
   - 7.1 $\mathcal{F}_{\mathsf{KG}} \xrightarrow{(\texttt{KEY-SHARE}, P_i^*, pk, evk, sk_i)} \mathcal{SIM}$

Figure 12: $\mathcal{F}_{\mathsf{KG}}$ The Ideal Functionality for FHE Key Generation

Functionality $\mathcal{F}_{\mathsf{Dec}}$

This functionality is parameterized by $\mathsf{FHE.Dec}(\cdot)$

Dec

1. For honest P_i

   1.1 $P_1 \xrightarrow{(\texttt{DEC-INPUT}, P_i, (\mathbf{ct}_1^i, \ldots, \mathbf{ct}_m^i), pk_i, sk_i)} \mathcal{F}_{\mathsf{Dec}}$

   1.2 $\mathcal{F}_{\mathsf{Dec}} \xrightarrow{(\texttt{DEC-INPUT}, P_i)} \mathcal{SIM}$

2. Else

   2.1 $\mathcal{SIM} \xrightarrow{(\texttt{DEC-INPUT}, P_i, (\mathbf{ct}_1^i, \ldots, \mathbf{ct}_m^i), pk_i, sk_i)} \mathcal{F}_{\mathsf{Dec}}$

3. If $pk_i \neq pk_j$ for any $i, j \in [m]$, output $\perp$, else set $pk = pk_i$

4. If $\mathbf{ct}_k^i \neq \mathbf{ct}_k^j$ for any $i, j, k \in [m]$, output $\perp$

5. Compute $sk = sk_1 \oplus \ldots \oplus sk_m$

6. If sk is not the secret key to pk, output $\perp$

7. Else compute $s_i = \mathsf{FHE.Dec}(sk, \mathbf{ct}_i)$ for $i \in [m]$

8. For honest P_i

   8.1 $\mathcal{F}_{\mathsf{Dec}} \xrightarrow{(\texttt{DEC-OUTPUT}, P_i, s_i)} P_i$

   8.2 $\mathcal{F}_{\mathsf{Dec}} \xrightarrow{(\texttt{DEC-OUTPUT}, P_i)} \mathcal{SIM}$

9. For corrupt P_i^*

   9.1 $\mathcal{F}_{\mathsf{Dec}} \xrightarrow{(\texttt{DEC-OUTPUT}, P_i^*, s_i)} \mathcal{SIM}$

Figure 13: $\mathcal{F}_{\mathsf{Dec}}$ The Ideal Functionality for FHE Decryption

Protocol $\Pi^{FHE}_{\mathsf{Per\text{-}Party}}$

Inputs: x_i for each client C_i, $i \in [1, ..., n]$. $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$, a fully homomorphic encryption scheme. $\mathsf{VSS} = (\mathsf{Share}, \mathsf{Reconst})$, a verifiable secret sharing scheme. $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, a signature scheme. Π^{KG}, Π^{Dec} two $(m, m-1)$ UC-secure MPC protocols for ideal functionalities $\mathcal{F}_{\mathsf{KG}}$ and $\mathcal{F}_{\mathsf{Dec}}$ respectively. $\mathsf{SNARG} = (\mathsf{SetUp}, \mathsf{Prove}, \mathsf{Verify})$, a SNARG. crs, a common reference string generated by a trusted set up.
Outputs: y_i for each client C_i, $i \in [1, ..., n]$

1. **Set Up:**
   - 1.1 Each client C_i
     - 1.1.1 Choose a set of servers $\mathcal{S}_{C_i}$ from registry $\mathcal{S}_{reg}$ and choose a threshold $\tau_i < |\mathcal{S}_{C_i}|$
     - 1.1.2 Compute $(pk^{\Sigma}, sk^{\Sigma}) = \Sigma.\mathsf{Gen}(1^{\lambda})$ and announce $(C_i, \mathcal{S}_{C_i}, \tau_i)$
   - 1.2 All servers collaboratively run Π^{KG} to execute $\mathsf{FHE.KeyGen}(1^{\lambda})$
     - 1.2.1 Server S_k receives pk, evk, and sk_k as output, where sk_k is a share of the secret key
2. **Input Phase:** Each client C_i
   - 2.1 Compute $\mathsf{VSS.Share}(x_i, \tau_i, m_i) = (s_{i,1}, ..., s_{i,m_i})$
   - 2.2 For each $j \in [1, \ldots, m_i]$ compute $\sigma_{i,j} = \Sigma.\mathsf{Sign}(sk_i^{\Sigma}, (s_{i,j}, C_i, S_{i,j}))$
   - 2.3 Send $(s_{i,j}, \sigma_{i,j}, pk_i^{\Sigma})$ to $S_{i,j}$
   - 2.4 For each server $S_{i,j} \in \mathcal{S}_{C_i}$, if $\Sigma.\mathsf{Verify}(pk_i^{\Sigma}, (s_{i,j}, C_i, S_{i,j})) = 0$ abort, else continue
3. **Computation Phase:** Let $(s_{i,j}, \sigma_{i,j}, pk_i^{\Sigma}) \in \mathcal{I}_k$ iff $S_k = S_{i,j} \in \mathcal{S}_{C_i}$
   - 3.1 For $S_k \in \mathcal{S}$
     - 3.1.1 Compute $\mathbf{ct}^{in}_{i,j} = \mathsf{FHE.Enc}(pk, (s_{i,j}, \sigma_{i,j}, pk_i^{\Sigma}))$ for each $(s_{i,j}, \sigma_{i,j}) \in \mathcal{I}_k$

       Broadcast each $\mathbf{ct}^{in}_{i,j}$ to all servers
   - 3.2 S_1, upon receiving all ciphertexts $\mathbf{ct}^{in}_{i,j}$,
     - 3.2.1 Compute $\mathsf{FHE.Eval}(evk, g, \mathbf{ct}^{in}_{1,1}, \ldots, \mathbf{ct}^{in}_{n,m_n}) = (\mathbf{ct}^{out}_{1,1}, \ldots, \mathbf{ct}^{out}_{n,m_n})$. Let $trans$ be the transcript of the computation
     - 3.2.2 Compute $\mathsf{SNARG.Prove}(crs, (g, pk, evk, \mathbf{ct}^{in}_{1,1}, \ldots, \mathbf{ct}^{in}_{n,m_n}, \mathbf{ct}^{out}_{1,1}, \ldots, \mathbf{ct}^{out}_{n,m_n}), trans) = \pi_{out}$, a proof that $\mathsf{FHE.Eval}(evk, g, \mathbf{ct}^{in}_{1,1}, \ldots, \mathbf{ct}^{in}_{n,m_n} \; ; \; r) = (\mathbf{ct}^{out}_{1,1}, \ldots, \mathbf{ct}^{out}_{n,m_i})$
     - 3.2.3 Broadcast $(\mathbf{ct}^{out}_{1,1}, \ldots, \mathbf{ct}^{out}_{n,m_n}, \pi_{out})$ to all servers
   - 3.3 All servers run Π^{Dec} to execute $\mathsf{FHE.Dec}((sk_1 \oplus \ldots \oplus sk_m), (\mathbf{ct}^{out}_{1,1}, \ldots, \mathbf{ct}^{out}_{n,m_n})) = (\mathcal{O}_1, \ldots, \mathcal{O}_m)$ where each $S_k = S_{i,j}$ has as input $(\mathbf{ct}^{out}_{1,1}, \ldots, \mathbf{ct}^{out}_{n,m_n}, pk, sk_k)$ and receives $s'_{i,j}$ as output
4. **Output Phase:** For all $s'_{i,j} \in \mathcal{O}_k$, server S_k returns $s'_{i,j}$ to C_i.

   If $\mathsf{VSS.Reconst}(s'_{i,1}, \ldots, s'_{i,m_i}) = y_i$ fails, output $\perp$ and abort. Else output y_i

Figure 14: Per-Party Private MPC Protocol $\Pi^{FHE}_{\mathsf{Per\text{-}Party}}$

Theorem 2 *If protocols* Π^{KG} *and* Π^{Dec} *UC-realize* $\mathcal{F}_{\mathsf{KG}}$ *and* $\mathcal{F}_{\mathsf{Dec}}$ *respectively in presence of* $(m-1)$ *corruptions, if* FHE *is a secure fully homomorphic encryption scheme, and* VSS *is an information theoretic private* (τ_i, m_i)*-threshold verifiable secret sharing scheme for* $1 \leq \tau_i < m_i$ *and* $1 \leq i \leq n$*, and* $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is an EUF-CMA signature scheme, where* n *is the number of clients and* m_i *is the number of servers chosen by client* C_i *then Protocol* $\Pi^{FHE}_{\mathsf{Per\text{-}Party}}$ *(Figure 14) UC-realizes* $\mathcal{F}_{\mathsf{Per\text{-}Party}}$ *(Figure 3).*

Correctness Our proof of correctness is mostly straightforward, based on the correctness of Π^{KG}, Π^{Dec}, FHE and VSS.

Simulator We give our simulator $\mathcal{SIM}^{\mathsf{FHE}}$ in Figures 15, 16, 17, and 18. We divide the simulator into four algorithms for readability purposes only. These algorithms are to be run consecutively.

In the set up phase (Figure 15), $\mathcal{SIM}^{\mathsf{FHE}}$ notifies the ideal functionality of the set of corrupt servers and clients, as well as the set of chosen servers and chosen threshold of the malicious clients. Then, $\mathcal{SIM}^{\mathsf{FHE}}$ simulates the execution of Π^{KG} using the underlying simulator $\mathcal{SIM}^{\mathsf{FHE}}_{\Pi^{\mathsf{KG}}}$. As a result of this simulation, $\mathcal{SIM}^{\mathsf{FHE}}$ learns the shares of the secret key held by each malicious server. Lastly, for all honest clients C_i, $\mathcal{SIM}^{\mathsf{FHE}}$ chooses a key pair for the signature scheme Σ.

During the input phase (Figure 16), $\mathcal{F}_{\mathsf{Per\text{-}Party}}$ notifies $\mathcal{SIM}^{\mathsf{FHE}}$ of the input of a client who chose greater than their threshold of malicious servers. This is because these malicious servers will not only learn the input of the client, but have the ability to replace the client's input with one of their choosing. Then, there are four cases for clients sending input to their servers. (1) If both the client and server are honest, there is nothing to simulate. (2) If the client is malicious but the server is honest, $\mathcal{SIM}^{\mathsf{FHE}}$ intercepts the share and signature intended for the honest server. (3) If the client is honest but the server is malicious, then $\mathcal{SIM}^{\mathsf{FHE}}$ must "fake" the share and signature. If $\mathcal{SIM}^{\mathsf{FHE}}$ knows the honest client's input x_i (i.e. C_i choose too many malicious servers), then $\mathcal{SIM}^{\mathsf{FHE}}$ computes shares and signatures honestly. If $\mathcal{SIM}^{\mathsf{FHE}}$ does not know x_i, then $\mathcal{SIM}^{\mathsf{FHE}}$ shares 0. (4) If both the client and the server are malicious, there is nothing to simulate.

Next, in the computation phase (Figure 17), the simulator receives all input ciphertexts of the malicious servers via broadcast. For an honest server $S_{i,j}$, if $\mathcal{SIM}^{\mathsf{FHE}}$ already knows a share for $S_{i,j}$ (i.e. received it from a malicious client C_i^* or already computed shares of 0 for $\mathcal{S}_{C_i}$), $\mathcal{SIM}^{\mathsf{FHE}}$ encrypts the share and broadcasts. Else $\mathcal{SIM}^{\mathsf{FHE}}$ encrypts 0.

Then, $\mathcal{SIM}^{\mathsf{FHE}}$ decrypts the input ciphertexts to obtain the input shares of the malicious servers. This is done both to reconstruct a malicious client's input and to determine if, for a client who chose too many malicious servers, the malicious clients have altered an honest client's input. When reconstructing inputs, $\mathcal{SIM}^{\mathsf{FHE}}$ verifies the signatures sent by each malicious server. If a malicious server submits a signature not computed by $\mathcal{SIM}^{\mathsf{FHE}}$ which still verifies under the client's verification key pk^{Σ}, then $\mathcal{SIM}^{\mathsf{FHE}}$ aborts. If there does not exist more than τ_i servers in the set $\mathcal{S}_{C_i}$ that agree on the client's verification key pk_i^{Σ}, the client's input is set to 0. Further, if there are not more than τ_i signatures that verify, the client's input is set to 0. Else $\mathcal{SIM}^{\mathsf{FHE}}$ reconstructs x_i using the shares that verify. If reconstruction fails, $\mathcal{SIM}^{\mathsf{FHE}}$ aborts.

Algorithm $\mathcal{SIM}^{\mathsf{FHE}}$ - Set Up

1. Choose $crs \leftarrow \mathsf{SNARG.SetUp}(1^\lambda)$ and publish
2. $C_i^* \xrightarrow{(\texttt{CLIENT-SET}, C_i^*, \mathcal{S}_{C_i^*}, \tau_i)} \mathcal{SIM}^{\mathsf{FHE}}$ for malicious C_i^*
3. $\mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{(\texttt{CORRUPTIONS}, \mathcal{C}_M, \mathcal{S}_M)} \mathcal{F}_{\mathsf{Per\text{-}Party}}$
4. $\mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{(\texttt{CLIENT-SET}, C_i^*, \mathcal{S}_{C_i^*}, \tau_i)} \mathcal{F}_{\mathsf{Per\text{-}Party}}$ for each malicious C_i^*
5. $\mathcal{F}_{\mathsf{Per\text{-}Party}} \xrightarrow{(\texttt{MAPPING}, \textbf{Map})} \mathcal{SIM}^{\mathsf{FHE}}$
6. $\mathcal{SIM}^{\mathsf{FHE}}$ activates $\mathcal{SIM}^{\mathsf{FHE}}_{\Pi^{\mathsf{KG}}}$, the simulator for Π^{KG}
7. $\mathcal{SIM}^{\mathsf{FHE}}_{\Pi^{\mathsf{KG}}}$ simulates Π^{KG}
   - 7.1 For any message ρ sent by $S_k^* \in \mathcal{S}_M$
     - 7.1.1 $S_k^* \xrightarrow{(\rho)} \mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{(\rho)} \mathcal{SIM}^{\mathsf{FHE}}_{\Pi^{\mathsf{KG}}}$
   - 7.2 For any message α sent by honest S_j to $S_k^* \in \mathcal{S}_M$
     - 7.2.1 $\mathcal{SIM}^{\mathsf{FHE}}_{\Pi^{\mathsf{KG}}} \xrightarrow{(\alpha)} \mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{(\alpha)} S_k^*$
   - 7.3 For each corrupt $S_k^* \in \mathcal{S}_M$, $\mathcal{SIM}^{\mathsf{FHE}}$ intercepts messages intended for $\mathcal{F}_{KG}$
     - 7.3.1 $\mathcal{SIM}^{\mathsf{FHE}}_{\Pi^{\mathsf{KG}}} \xrightarrow{(\texttt{KG-RAND}, S_k^*, r_k)} \mathcal{SIM}$
     - 7.3.2 Sample $r_i \xleftarrow{\$} \{0,1\}^\lambda$ such that $S_i \in \mathcal{S} \setminus \mathcal{S}_M$
     - 7.3.3 Run $(pk, evk, sk) = \mathsf{FHE.KeyGen}(1^\lambda\ ;\ r_0 \oplus \ldots \oplus r_m)$
     - 7.3.4 Choose $\{sk_1, \ldots, sk_{m-1}\} \xleftarrow{\$} \mathbb{K}$ where $\mathbb{K}$ is the key space of FHE, and set $sk_m = sk_1 \oplus \ldots \oplus sk_{m-1} \oplus sk$
   - 7.4 For each corrupt server $S_k^* \in \mathcal{S}_M$
     - 7.4.1 $\mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{(\texttt{KEY-SHARE}, S_k^*, pk, evk, sk_k)} \mathcal{SIM}^{\mathsf{FHE}}_{\Pi^{\mathsf{KG}}}$
8. For each honest C_i
   - 8.1 Compute $(pk_i^\Sigma, sk_i^\Sigma) = \Sigma.\mathsf{Gen}(1^\lambda)$

Figure 15: Set Up Phase of $\mathcal{SIM}^{\mathsf{FHE}}$

Algorithm $\mathcal{SIM}^{\mathsf{FHE}}$ - Input Phase

1. For each C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$

   1.1 $\mathcal{F}_{\mathsf{Per\text{-}Party}} \xrightarrow{(\texttt{CLIENT-INPUT}, C_i, x_i)} \mathcal{SIM}^{\mathsf{FHE}}$

2. For honest C_i and honest S_k

   2.1 $\mathcal{SIM}^{\mathsf{FHE}}$ sends and receives nothing

3. For malicious C_i^* and honest S_k

   3.1 $C_i^* \xrightarrow{(s_{i,j}^{in}, \sigma_{i,j}, pk_i^{\Sigma})} \mathcal{SIM}^{\mathsf{FHE}}$ such that $S_k = S_{i,j} \in \mathcal{S}_{C_i}$

4. For honest C_i and malicious $S_k^* = S_{i,j}^*$

   4.1 If $\mathcal{SIM}^{\mathsf{FHE}}$ knows x_i

   4.1.1 $\mathcal{SIM}^{\mathsf{FHE}}$ computes $\mathsf{VSS.Share}(x_i, \tau_i, m_i) = (s_{i,1}, \ldots, s_{i,m_i})$ if not already computed

   4.1.2 $\mathcal{SIM}^{\mathsf{FHE}}$ computes $\Sigma.\mathsf{Sign}(sk_i^{\Sigma}, (s_{i,j}, C_i, S_{i,j}^*))$

   4.1.3 $\mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{(s_{i,j}, \sigma_{i,j}, pk_i^{\Sigma})} S_k^*$

   4.2 Else if $\mathcal{SIM}^{\mathsf{FHE}}$ does not know x_i

   4.2.1 $\mathcal{SIM}^{\mathsf{FHE}}$ computes $\mathsf{VSS.Share}(0, \tau_i, m_i) = (\widetilde{s}_{i,1}, \ldots, \widetilde{s}_{i,m_i})$ if not already computed

   4.2.2 $\mathcal{SIM}^{\mathsf{FHE}}$ computes $\widetilde{\sigma_{i,j}} = \Sigma.\mathsf{Sign}(sk_i^{\Sigma}, (\widetilde{s}_{i,j}, C_i, S_{i,j}^*))$

   4.2.3 $\mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{(\widetilde{s}_{i,j}, \widetilde{\sigma_{i,j}}, pk_i^{\Sigma})} S_k^*$ such that $S_k^* = S_{i,j} \in \mathcal{S}_{C_i}$

5. For malicious C_i^* and malicious S_k^*

   5.1 $\mathcal{SIM}^{\mathsf{FHE}}$ sends and receives nothing

Figure 16: Input Phase of $\mathcal{SIM}^{\mathsf{FHE}}$

Algorithm $\mathcal{SIM}^{\mathsf{FHE}}$ - Computation Phase

1. $\mathcal{F}_{\text{Per-Party}} \xrightarrow{(\texttt{RUN})} \mathcal{SIM}^{\mathsf{FHE}}$, set ABORT $=$ *False*, receive $\mathbf{ct}_{i,j}^{in}$ via broadcast $\forall\ S_{i,j}^* \in \mathcal{S}_M$
2. $\forall\ S_{i,j} \notin \mathcal{S}_M$, if $(s_{i,j}, \sigma_{i,j})$ exists, compute $\mathbf{ct}_{i,j}^{in} = \mathsf{FHE.Enc}(pk, s_{i,j})$, else compute $\widetilde{\mathbf{ct}_{i,j}}^{in} = \mathsf{FHE.Enc}(pk, 0)$, and broadcast all ciphertexts to all servers. $\forall\ S_{i,j} \in \mathcal{S}_M$, receive $\mathbf{ct}_{i,j}^{in}$ and compute $(s_{i,j}^*, \sigma_{i,j}^*) = \mathsf{FHE.Dec}(sk, \mathbf{ct}_{i,j}^{in})$
3. For honest C_i and corrupt $S_{i,j}^*$ such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| \leq \tau_i$, $\forall\ (s_{i,j}^*, \sigma_{i,j}^*)$ if $s_{i,j}^* \neq s_{i,j}$ and $\Sigma.\mathsf{Verify}(pk_i^{\Sigma}, \sigma_{i,j}^*, (s_{i,j}^*, C_i, S_{i,j}^*)) = 1$ output *SigFail* and abort
4. For corrupt C_i^* and all $S_{i,j}^* \in \mathcal{S}_{C_i}$, let pk_i^{Σ} be submitted by more than τ_i servers in $\mathcal{S}_{C_i^*}$ and set $R_i = \varnothing$
   - 4.1 If $\nexists\ pk_i^{\Sigma}$, set $x_i^* = 0$. If $\Sigma.\mathsf{Verify}(pk_i^{\Sigma}, \sigma_{i,j}^*, (s_{i,j}^*, C_i^*, S_{i,j}^*))$ set $R_i = R_i \cap s_{i,j}\ \forall\ S_{i,j} \in \mathcal{S}_{C_i}$
   - 4.2 If $|R_i| \leq \tau_i$, set $x_i = 0$, else compute $x_i = \mathsf{VSS.Reconst}(R_i)$. If reconstruction fails, set ABORT $=$ *True* and go to 9
5. For $C_i \notin \mathcal{C}_M$ such that $|\mathcal{S}_M \cap \mathcal{S}_{C_i}| > \tau_i$, let pk_i^{Σ} be the key submitted by more than τ_i of the servers in $\mathcal{S}_{C_i}$ and set $R_i = \varnothing$
   - 5.1 If $\nexists\ pk_i^{\Sigma}$, set $x_i^* = 0$. If $\Sigma.\mathsf{Verify}(pk_i^{\Sigma}, \sigma_{i,j}^*, (s_{i,j}^*, C_i^*, S_{i,j}^*))$ set $R_i = R_i \cap s_{i,j}$ for all $S_{i,j} \in \mathcal{S}_{C_i}$
   - 5.2 If $|R_i| \leq \tau_i$ set $x_i = 0$, else compute $x_i = \mathsf{VSS.Reconst}(R_i)$. If reconstruction fails, set ABORT $=$ *True* and go to 9
6. If ABORT $=$ *False*, $\mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{(\texttt{RUN}, (\texttt{CLIENT-INPUT}, C_i^*, x_i), \ldots, (\texttt{MAL-INPUT}, C_j, x_j^*), \ldots)} \mathcal{F}_{\text{Per-Party}}$ for each malicious C_i^* and honest C_j such that $|\mathcal{S}_{C_j} \cap \mathcal{S}_M| > \tau_j$, else $\mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{(\texttt{RUN}, \textit{Abort})} \mathcal{F}_{\text{Per-Party}}$, output *ReconstAbort* and abort
7. If $S_1^* \in \mathcal{S}_M$, receive $(\mathbf{ct}_{1,1}^{out}, \ldots, \mathbf{ct}_{n,m_n}^{out}, \pi^{out})$ from S_1^* and compute $\mathsf{SNARG.Verify}(crs, \pi_{out}, (g, pk, evk, \mathbf{ct}_{1,1}^{in}, \ldots, \mathbf{ct}_{n,m_n}^{in}, \mathbf{ct}_{1,1}^{out}, \ldots, \mathbf{ct}_{n,m_n}^{out})) = b$, if $b = 0$ abort
   - 7.1 $\forall\ \mathbf{ct}_{i,j}^{out}$ compute $s'_{i,j} = \mathsf{FHE.Dec}(sk, \mathbf{ct}_{i,j}^{out})$, if $s'_{i,j} \notin \mathcal{O}_k$ for $g(\mathsf{VSS}, \Sigma, \mathcal{I}_1, \ldots, \mathcal{I}_m) = (\mathcal{O}_1, \ldots, \mathcal{O}_m)$ and any $S_k = S_{i,j}$, output *SoundFail* and abort
8. If $S_1 \notin \mathcal{S}_M$, compute $\mathsf{FHE.Eval}(evk, g, \mathbf{ct}_{1,1}^{in}, \ldots, \mathbf{ct}_{n,m_n}^{in}) = (\mathbf{ct}_{1,1}^{out}, \ldots, \mathbf{ct}_{n,m_n}^{out})$ and let *trans* be the transcript resulting from the computation
   - 8.1 Compute $\mathsf{SNARG.Prove}(crs, (g, pk, evk, \mathbf{ct}_{1,1}^{in}, \ldots, \mathbf{ct}_{n,m_n}^{in}, \mathbf{ct}_{1,1}^{out}, \ldots, \mathbf{ct}_{n,m_n}^{out}), trans) = \pi_{out}$, a proof that $\mathsf{FHE.Eval}(evk, g, \mathbf{ct}_{1,1}^{in}, \ldots, \mathbf{ct}_{n,m_n}^{in}) = (\mathbf{ct}_{1,1}^{out}, \ldots, \mathbf{ct}_{n,m_i}^{out})$ and broadcast $(\mathbf{ct}_{1,1}^{out}, \ldots, \mathbf{ct}_{n,m_n}^{out}, \pi_{out})$ to all malicious servers
9. Activate $\mathcal{SIM}_{\Pi^{\mathsf{Dec}}}^{\mathsf{FHE}}$, the simulator for Π^{Dec}
   - 9.1 Simulate messaging for necessary parties
   - 9.2 For $S_k^* \in \mathcal{S}_M$, $\mathcal{SIM}_{\Pi^{\mathsf{Dec}}}^{\mathsf{FHE}} \xrightarrow{(\texttt{DEC-INPUT}, S_k^*, (\mathbf{ct}_{1,1}^{out,k}, \ldots, \mathbf{ct}_{n,m_n}^{out,k}), pk_k, sk_k)} \mathcal{SIM}^{\mathsf{FHE}}$
     - 9.2.1 If $\mathbf{ct}_{i,j}^{out,a} \neq \mathbf{ct}_{i,j}^{out,b}$ or $pk_a \neq pk_b$ for any $i \in [n], j \in [m]$, and $S_a^*, S_b^* \in \mathcal{S}_M$, abort

Figure 17: Computation Phase of $\mathcal{SIM}^{\mathsf{FHE}}$

Algorithm $\mathcal{SIM}^{\mathsf{FHE}}$ - Output Phase

1. $\mathcal{F}_{\mathsf{Per\text{-}Party}} \xrightarrow{((\texttt{CLIENT-OUTPUT}, C_i, y_i), \ldots)} \mathcal{SIM}^{\mathsf{FHE}}$ For each C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$
2. $\mathcal{F}_{\mathsf{Per\text{-}Party}} \xrightarrow{((\texttt{CLIENT-OUTPUT}, C_i^*, y_i), \ldots)} \mathcal{SIM}^{\mathsf{FHE}}$ for each malicious C_i^*
3. $\mathcal{SIM}^{\mathsf{FHE}}$ computes $\mathsf{VSS.Share}(y_i, \tau_i, m_i) = (s'_{i,1}, \ldots, s'_{i,m_i})$ for all $C_i \in \mathcal{C}_M$ and C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$
4. $\mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{(\texttt{DEC-OUTPUT}, S^*_{i,j}, s'_{i,j})} \mathcal{SIM}^{\mathsf{FHE}}_{\Pi^{\mathsf{Dec}}}$ for all $S^*_{i,j} \in \mathcal{S}_M$
5. Finish the simulation of Π^{Dec}
6. For each honest C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$
   - 6.1 $S^*_{i,j} \xrightarrow{(s^*_{i,j})} \mathcal{SIM}^{\mathsf{FHE}}$ for all $S^*_{i,j} \in \mathcal{S}_{C_i}$
   - 6.2 Compute $\mathsf{VSS.Reconst}(s^*_{i,0}, \ldots, s^*_{i,\ell}) = y^*_i$ such that $\tau_i < \ell = |\mathcal{S}_{C_i} \cap \mathcal{S}_M|$. If reconstruction fails, set $y^*_i = \perp$
7. $\mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{((\texttt{MAL-OUTPUT}, C_i, y^*_i), \ldots)} \mathcal{F}_{\mathsf{Per\text{-}Party}}$ for each honest C_i such that $|\mathcal{S}_{C_i} \cap \mathcal{S}_M| > \tau_i$
8. For malicious C_i^* and honest $S_k = S_{i,j} \in \mathcal{S}_{C_i^*}$
   - 8.1 $\mathcal{SIM}^{\mathsf{FHE}} \xrightarrow{(s'_{i,j})} C_i^*$

Figure 18: Output Phase of $\mathcal{SIM}^{\mathsf{FHE}}$

$\mathcal{SIM}^{\mathsf{FHE}}$ then sends the reconstructed inputs to $\mathcal{F}_{\mathsf{Per\text{-}Party}}$. If the computing server S_1 is corrupted, then S_1 performs the homomorphic computation and proof π_{out}. Else $\mathcal{SIM}^{\mathsf{FHE}}$ performs this computation and proof. $\mathcal{SIM}^{\mathsf{FHE}}$ then begins the simulation of Π^{Dec}. Through this simulation, $\mathcal{SIM}^{\mathsf{FHE}}$ receives the shares of the secret key of each malicious server.

Finally, in the output phase (Figure 18), $\mathcal{SIM}^{\mathsf{FHE}}$ uses the outputs y_i provided by $\mathcal{F}_{\mathsf{Per\text{-}Party}}$ in the simulation of Π^{Dec} to ensure that corrupt clients receive the correct output. Further, $\mathcal{SIM}^{\mathsf{FHE}}$ intercepts the shares sent by malicious servers to an honest client who chose too many corrupt servers. These shares are used to reconstruct the client's maliciously altered output, which is then provided to $\mathcal{F}_{\mathsf{Per\text{-}Party}}$. In the final step, $\mathcal{SIM}^{\mathsf{FHE}}$ sends shares of a malicious client's output on behalf of the honest servers they chose.

Simulator Run Time The proof of polynomial runtime is straightforward, based on the fact that VSS, FHE, SNARG, Π^{KG}, and Π^{Dec} are all polynomial time.

Indistinguishability We prove the indistinguishability of the real and ideal worlds via a series of hybrids. Consider the following hybrids:

- $\mathbf{Hyb}_0$: The real world execution of $\Pi^{FHE}_{\mathsf{Per\text{-}Party}}$
- $\mathbf{Hyb}_1$: This is the same as $\mathbf{Hyb}_0$, except that the adversary outputs *ReconstAbort* if the reconstruction of input shares fails
- $\mathbf{Hyb}_2$: This is the same as $\mathbf{Hyb}_1$, except that FHE.KeyGen is simulated by $\mathcal{SIM}^{\mathsf{FHE}}_{\Pi^{\mathsf{KG}}}$ instead of run via the protocol Π^{KG}
- $\mathbf{Hyb}_3$: This is the same as $\mathbf{Hyb}_2$, except the adversary aborts with *SoundFail* if the computing server uses the incorrect output in the proof π_{out}
- $\mathbf{Hyb}_4$: This is the same as $\mathbf{Hyb}_3$, except FHE.Dec is simulated by $\mathcal{SIM}^{\mathsf{FHE}}_{\Pi^{\mathsf{Dec}}}$ instead of run via the protocol Π^{Dec}
- $\mathbf{Hyb}_5$: This is the same as $\mathbf{Hyb}_4$, except that ciphertexts of 0 are sent as input for honest server $S_{i,j}$ chosen by honest client C_i instead of ciphertexts of $s_{i,j}$
- $\mathbf{Hyb}_6$: This is the same as $\mathbf{Hyb}_5$, except that the adversary outputs *SigFail* if a malicious server chosen by an honest client submits a signature that was not computed by the adversary
- $\mathbf{Hyb}_7$: This is the same as $\mathbf{Hyb}_6$, except that instead of honest clients sending shares of their input, shares of 0 are sent instead

Lemma 5 *If* $\mathsf{VSS} = (\mathsf{Share}, \mathsf{Reconst})$ *is a verifiable secret sharing scheme, then* $\mathbf{Hyb}_1$ *is indistinguishable from* $\mathbf{Hyb}_0$

Proof Follows from the proof of Lemma 1. ■

Lemma 6 *If* Π^{KG} *is a UC-secure MPC protocol,* $\mathbf{Hyb}_2$ *is indistinguishable from* $\mathbf{Hyb}_1$

Proof Follows from the proof of Lemma 3. ■

Lemma 7 *If* $\mathsf{SNARG} = (\mathsf{SetUp}, \mathsf{Prove}, \mathsf{Verify})$ *is a sound succinct non-interactive argument,* $\mathbf{Hyb}_3$ *is indistinguishable from* $\mathbf{Hyb}_2$

Proof Towards a contradiction, assume that $Pr[\mathcal{SIM}^{\mathsf{FHE}}(1^\lambda) \rightarrow \mathit{SoundFail}] = p(\lambda)$ for non-negligible $p(\lambda)$. Then, we can construct a reduction $\mathcal{B}$ that violates the soundness of SNARG, using $\mathcal{A}$ controlling the corrupt clients and servers in the ideal world. We define $\mathcal{B}$ in Figure 19.

As we can see, $\mathcal{A}$ submits a proof π_{out} that verifies but $\mathsf{FHE.Eval}(evk, g, \mathbf{ct}^{in}_{1,1}, \ldots, \mathbf{ct}^{in}_{n,m_n}) \neq (\mathbf{ct}^{out}_{1,1}, \ldots, \mathbf{ct}^{out}_{n,m_i})$. The conditions checked by $\mathcal{B}$ are exactly the conditions required for $\mathcal{SIM}^{\mathsf{FHE}}$ to abort with *SoundFail*. Therefore, $\mathcal{B}$ violates the soundness of SNARG with the same probability as $\mathcal{SIM}^{\mathsf{FHE}}$ has of aborting with *SoundFail*. This is a contradiction, thus $\mathcal{SIM}^{\mathsf{FHE}}$ aborts with *SoundFail* with negligible probability. ■

$\underline{\mathcal{B}(crs)}$

1. Activate $\mathcal{A}(1^\lambda)$
2. Simulate as in $\mathbf{Hyb}_2$, using crs as the common reference string, until step 11 of the computation phase
3. If S_1 is not corrupt, abort
4. Else
   - 4.1 Receive $(\mathbf{ct}^{out}_{1,1}, \ldots, \mathbf{ct}^{out}_{n,m_n}, \pi^{out})$ from $\mathcal{A}$
   - 4.2 Compute $\mathsf{SNARG.Verify}(crs, \pi_{out}, (\mathbf{ct}^{in}_{1,1}, \ldots, \mathbf{ct}^{in}_{n,m_n}, \mathbf{ct}^{out}_{1,1}, \ldots, \mathbf{ct}^{out}_{n,m_n})) = b$, if $b = 0$ abort
     - 4.2.1 For all $\mathbf{ct}^{out}_{i,j}$ compute $s'_{i,j} = \mathsf{FHE.Dec}(sk, \mathbf{ct}^{out}_{i,j})$
     - 4.2.2 If $s'_{i,j} \notin \mathcal{O}_k$ for $g(\mathsf{VSS}, \Sigma, \mathcal{I}_1, \ldots, \mathcal{I}_m) = (\mathcal{O}_1, \ldots, \mathcal{O}_m)$ and any $S_k = S_{i,j}$, submit $(\pi_{out}, (\mathbf{ct}^{in}_{1,1}, \ldots, \mathbf{ct}^{in}_{n,m_n}, \mathbf{ct}^{out}_{1,1}, \ldots, \mathbf{ct}^{out}_{n,m_n}))$ to the challenger

Figure 19: $\mathcal{B}$ - The Adversary for Soundness of SNARG

Lemma 8 *If* Π^{Dec} *is a UC-secure MPC protocol,* $\mathbf{Hyb}_3$ *is indistinguishable from* $\mathbf{Hyb}_4$

Proof Follows from the proof of Lemma 3 ■

Lemma 9 *If* $\mathsf{FHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is an IND-CPA secure encryption scheme,* $\mathbf{Hyb}_4$ *is indistinguishable from* $\mathbf{Hyb}_5$

Proof We prove indistinguishability through a series of sub-hybrids. Let $\{\mathbf{ct}^{in}_{i,j}\}_{S_{i,j} \notin \mathcal{S}_M}$ be the set of ciphertexts sent by the honest servers at the beginning of the computation phase. Then, let $\mathbf{Hyb}_{3,i}$ be such that the first $i-1$ ciphertexts in $\{\mathbf{ct}^{in}_{i,j}\}_{S_{i,j} \notin \mathcal{S}_M}$ are encryptions of zero $\widetilde{\mathbf{ct}_{i',j'}}$, and all following ciphertexts are encryptions of input shares.

Note that $\mathbf{Hyb}_{4,1}$ is exactly $\mathbf{Hyb}_4$, as all ciphertexts are honest encryptions of input shares. Likewise, $\mathbf{Hyb}_{4,|S_{i,j} \notin \mathcal{S}_M|}$ is exactly $\mathbf{Hyb}_5$, as all ciphertexts are encryptions of 0. It suffices to prove that $\mathbf{Hyb}_{4,i}$ is indistinguishable from $\mathbf{Hyb}_{4,i+1}$.

Towards a contradiction assume that there exists an adversary $\mathcal{Z}$ that can distinguish between $\mathbf{Hyb}_{4,i}$ and $\mathbf{Hyb}_{4,i+1}$. We can then construct an adversary $\mathcal{A}$ that violates the CPA security of FHE. We define $\mathcal{A}$ in Figure 20.

$\underline{\mathcal{A}(1^\lambda)}$

1. Receive (pk, evk) from the challenger
2. Activate $\mathcal{Z}$
3. Receive $\mathbf{Map}, \mathcal{C}_M, \mathcal{S}_M$ x_i for $C_i \in \mathcal{C}_M$ and $\tau_1, \ldots, \tau_n$ from $\mathcal{Z}$
4. For each $C_i^* \in \mathcal{C}_M$ send the information $\mathcal{Z}$ expects after a client is corrupted
5. For each $S_k^* \in \mathcal{S}_M$, send the information $\mathcal{Z}$ expects after a server is corrupted
6. Upon simulating Π^{KG}, use (pk, evk) as the public key and evaluation key
7. Simulate as in $\mathbf{Hyb}_{4,i}$ until each server $S_{i,j}$ has their share $s_{i,j}$
   - 7.1 Without loss of generality, let $S_{i',j'}$ hold the ith share in the set $\{s_{i,j}^{in}\}_{S_{i,j} \notin \mathcal{S}_M}$
   - 7.2 Let $m_0 = s_{i',j'}$ and $m_1 = 0$
   - 7.3 Send m_0, m_1 to the challenger and receive $\mathbf{ct}^*$
   - 7.4 Use $\mathbf{ct}^*$ as the ciphertext for server $S_{i',j'}$
8. Finish simulating as in $\mathbf{Hyb}_{4,i}$
9. Output whatever $\mathcal{Z}$ outputs

Figure 20: $\mathcal{A}$ - The Adversary for the IND-CPA Game

There are two cases here, either $\mathbf{ct}^* = \mathsf{FHE.Enc}(pk, m_0)$ or $\mathbf{ct}^* = \mathsf{FHE.Enc}(pk, m_1)$

- **Case $\mathbf{ct}^* = \mathsf{FHE.Enc}(pk, m_0)$:** In this case, the ciphertext used as input by server $S_{i',j'}$ is a ciphertext of an honestly computed share. This is exactly what $\mathcal{Z}$ expects when the transcript received is from $\mathbf{Hyb}_{4,i}$
- **Case $\mathbf{ct}^* = \mathsf{FHE.Enc}(pk, m_1)$:** In this case, the ciphertext used as input by server $S_{i',j'}$ is a ciphertext of 0. This is exactly what $\mathcal{Z}$ expects when the transcript received is from $\mathbf{Hyb}_{4,i+1}$

Therefore, $\mathcal{A}$ has the same probability of winning the CPA game as $\mathcal{Z}$ has of distinguishing between $\mathbf{Hyb}_{4,i}$ and $\mathbf{Hyb}_{4,i+1}$. Since $\mathbf{Hyb}_{4,1}$ is exactly $\mathbf{Hyb}_4$ and $\mathbf{Hyb}_{4,|S_{i,j} \notin \mathcal{S}_M|}$ is exactly $\mathbf{Hyb}_5$, this means that $\mathbf{Hyb}_4$ is indistinguishable from $\mathbf{Hyb}_5$. ■

Lemma 10 *If $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is an EUF-CMA signature scheme, $\mathbf{Hyb}_6$ is indistinguishable from $\mathbf{Hyb}_5$*

Proof Follows from the proof of Lemma 2. ■

Lemma 11 *If $\mathsf{VSS} = (\mathsf{Share}, \mathsf{Reconst})$ is an information theoretic private verifiable secret sharing scheme, $\mathbf{Hyb}_7$ is indistinguishable from $\mathbf{Hyb}_6$*

Proof Follows from the proof of Lemma 4. ■

References

[AJJM20] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 28–57. Springer, 2020.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2012.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4):989–1066, 2017.

[BCPS20] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. FLASH: fast and robust framework for privacy-preserving machine learning. *Proc. Priv. Enhancing Technol.*, 2020(2):459–480, 2020.

[BEP23] Alexander Bienstock, Daniel Escudero, and Antigoni Polychroniadou. On linear communication complexity for (maximally) fluid MPC. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 263–294. Springer, 2023.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

[BLCW19] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. ngraph-he: a graph compiler for deep learning on homomorphically encrypted data. In Francesca Palumbo, Michela Becchi, Martin Schulz, and Kento Sato, editors, *Proceedings of the 16th ACM International Conference on Computing Frontiers, CF 2019, Alghero, Italy, April 30 - May 2, 2019*, pages 3–13. ACM, 2019.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513. ACM, 1990.

[BPP+17] Foteini Baldimtsi, Dimitrios Papadopoulos, Stavros Papadopoulos, Alessandra Scafuro, and Nikos Triandopoulos. Server-aided secure computation with off-line parties. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*, volume 10492 of *Lecture Notes in Computer Science*, pages 103–123. Springer, 2017.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, pages 62–73. ACM, 1993.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1082–1090. ACM, 2019.

[CCP22] Anirudh Chandramouli, Ashish Choudhury, and Arpita Patra. A survey on perfectly secure verifiable secret-sharing. *ACM Comput. Surv.*, 54(11s):232:1–232:36, 2022.

[CCPS19] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. ASTRA: high throughput 3pc over rings with application to secure prediction. In Radu Sion and Charalampos Papamanthou, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, CCSW@CCS 2019, London, UK, November 11, 2019*, pages 81–92. ACM, 2019.

[CGG+21] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: secure multiparty computation with dynamic participants. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 94–123. Springer, 2021.

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Snargs for $\mathcal{P}$ from LWE. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 68–79. IEEE, 2021.

[CKKC13] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Carlos Cid. Multi-client non-interactive verifiable computation. In Amit Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 499–518. Springer, 2013.

[CKL21] Jung Hee Cheon, Dongwoo Kim, and Keewoo Lee. Mhz2k: MPC from HE over $\mathbb {Z}_{2^k}$ with new packing, simpler reshare, and better ZKP. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 426–456. Springer, 2021.

[CMTB16] Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin R. B. Butler. Secure outsourced garbled circuit evaluation for mobile devices. *J. Comput. Secur.*, 24(2):137–180, 2016.

[DDG+23] Bernardo David, Giovanni Deligios, Aarushi Goel, Yuval Ishai, Anders Konring, Eyal Kushilevitz, Chen-Da Liu-Zhang, and Varun Narayanan. Perfect MPC over layered graphs. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 360–392. Springer, 2023.

[DEP23] Ivan Damgård, Daniel Escudero, and Antigoni Polychroniadou. Phoenix: Secure computation in an unstable network with dropouts and comebacks. In Kai-Min Chung, editor, *4th Conference on Information-Theoretic Cryptography, ITC 2023, June 6-8, 2023, Aarhus University, Aarhus, Denmark*, volume 267 of *LIPIcs*, pages 7:1–7:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2005.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 554–563. ACM, 1994.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, USA, 2009.

[GGJS11] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Bringing people of different beliefs together to do UC. In Yuval Ishai, editor, *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, volume 6597 of *Lecture Notes in Computer Science*, pages 311–328. Springer, 2011.

[GHK+21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: you only speak once - secure MPC with stateless ephemeral roles. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 2021.

[GJM+23] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Cryptography with weights: Mpc, encryption and signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 295–327. Springer, 2023.

[GLO+21] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. ATLAS: efficient and scalable MPC in the honest majority setting. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 244–274. Springer, 2021.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[GO14] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. *J. Cryptol.*, 27(3):506–543, 2014.

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. Snargs for P from sub-exponential DDH and QR. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 520–549. Springer, 2022.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In 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*, pages 132–150. Springer, 2011.

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. Snargs for bounded depth computations and PPAD hardness from sub-exponential LWE. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 708–721. ACM, 2021.

[JNO14] Thomas P. Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A framework for outsourcing of secure computation. In Gail-Joon Ahn, Alina Oprea, and Reihaneh Safavi-Naini, editors, *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, CCSW '14, Scottsdale, Arizona, USA, November 7, 2014*, pages 81–92. ACM, 2014.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2007.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[KMR11] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. *IACR Cryptol. ePrint Arch.*, page 272, 2011.

[KMR12] Seny Kamara, Payman Mohassel, and Ben Riva. Salus: a system for server-aided secure function evaluation. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 797–808. ACM, 2012.

[KPPS21] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. SWIFT: super-fast and robust privacy-preserving machine learning. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 2651–2668. USENIX Association, 2021.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1115–1124. ACM, 2019.

[Lin17] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium*

on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012, pages 1219–1234. ACM, 2012.

[MGBF14] Benjamin Mood, Debayan Gupta, Kevin R. B. Butler, and Joan Feigenbaum. Reuse it or lose it: More efficient secure computation through reuse of encrypted values. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 582–596. ACM, 2014.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 436–453. IEEE Computer Society, 1994.

[MOR16] Payman Mohassel, Ostap Orobets, and Ben Riva. Efficient server-aided 2pc for mobile phones. *Proc. Priv. Enhancing Technol.*, 2016(2):82–99, 2016.

[MR18] Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52. ACM, 2018.

[MTZC21] Jack P. K. Ma, Raymond K. H. Tai, Yongjun Zhao, and Sherman S. M. Chow. Let's stride blindfolded in a forest: Sublinear multi-client decision trees evaluation. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

[MZ17] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38. IEEE Computer Society, 2017.

[PS20] Arpita Patra and Ajith Suresh. BLAZE: blazing fast privacy-preserving machine learning. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[RS22] Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 719–749. Springer, 2022.

[SH21] Jaskaran V. Singh and Nicholas Hopper. Grades of trust in multiparty computation. *IACR Cryptol. ePrint Arch.*, page 82, 2021.

[TKTW21] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. Cryptgpu: Fast privacy-preserving machine learning on the GPU. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1021–1038. IEEE, 2021.

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proc. Priv. Enhancing Technol.*, 2019(3):26–49, 2019.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986.