Client-Private Secure Aggregation for Privacy-Preserving Federated Learning

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Privacy-preserving federated learning (PPFL) is a paradigm of distributed privacy-preserving machine learning training in which a set of clients jointly compute a shared global model under the orchestration of an aggregation server. The system has the property that no party learns any information about any client's training data, besides what could be inferred from the global model. The core cryptographic component of a PPFL scheme is the secure aggregation protocol, a secure multi-party computation protocol in which the server securely aggregates the clients' locally trained models, and sends the aggregated model to the clients. However, in many applications the global model represents a trade secret of the consortium of clients, which they may not wish to reveal in the clear to the server. In this work, we propose a novel model of secure aggregation, called client-private secure aggregation, in which the server computes an encrypted global model that only the clients can decrypt. We provide an explicit construction of a client-private secure aggregation protocol, as well as a theoretical and empirical evaluation of our construction to demonstrate its practicality. Our experiments demonstrate that the client and server running time of our protocol are less than 19 s and 2 s, respectively, when scaled to support 250 clients.

1 Introduction

Federated learning (FL) [24] is a paradigm of distributed machine learning (ML) training in which a set of n clients jointly compute a shared global model under the orchestration of an aggregation server, without sharing their local training data in the clear. This is particularly applicable in industries where sensitive data is distributed across silos and centralizing such data for analysis is infeasible. For example, in the healthcare domain, a consortium of healthcare providers, each hosting patient-level sensitive data, can contribute towards building a shared machine learning model to improve patient care, while aiding in complying with regulatory guidelines [10, 11, 12]. Similarly, in the finance industry, FL has been applied for credit card fraud detection, leveraging data hosted across several banks [31]. In FL, the aggregation server maintains the current state of the global model so that when new clients join, the global model can be distributed to each new client. Additionally, the aggregation server facilitates the communication between the consortium of clients so that the clients need not setup and maintain a complete graph network infrastructure to directly communicate with each other. In this work, we consider cross-silo FL, in which the clients are typically fixed institutions (e.g. businesses, institutions, hospitals, etc.), and total in number on the order of 100.

FL begins with the server sending the initial global model to all clients. Each client then locally trains the model on their training data to compute a local model update, which they send to the aggregation server. The server aggregates the model updates from the clients to compute a global model update, which it applies to the initial model to compute a new global model. The new global model is then broadcast to the clients. This process can be repeated until a global optimum is reached.

Note that after an iteration of FL, each party receives the new aggregated global model without any client sharing their training data in the clear. However, several attacks [30, 32] demonstrate that an adversary can reconstruct the clients' training data from their local model updates. The breakthrough work of [3] constructed a secure multi-party computation (MPC) protocol in which the server computes the sum of the clients' local model updates, which it then broadcasts to the clients. The security of the protocol enforces that no party learns any information about any client's model update, except for what could be inferred from the sum of the clients' model updates. FL with secure aggregation is commonly called *privacy-preserving federated learning (PPFL)*.

In the standard model of secure aggregation the server computes the global model in the clear, which it then broadcasts to the clients. However, in many cases the global model may represent a trade secret of the consortium of clients. As such, the clients may not wish to reveal their global model to the server, which, potentially, could be run by a cloud service provider independent of the consortium. In this work, we propose a model of secure aggregation in which the server computes an encrypted global model that can only be decrypted the clients. The result is that while the clients all obtain the global model in the clear, the server only ever sees the encrypted global model. Since in the cross-silo FL setting, the network availability of the clients is typically not an issue, we don't seek to address client dropout in our model of secure aggregation.

Prior Work. Beginning with the foundational work of [3], secure aggregation protocols have been widely studied in the literature [2, 6, 7, 23, 29]. Various protocols have been constructed which offer trade-offs with respect to the security model and computational, communication, space, and round complexity. The original work of [3] constructed a four-round secure aggregation protocol with semi-honest security (and a five-round variant with malicious security). Their protocol works by each client choosing a random mask which locally encrypts their input as a one-time pad (OTP), but with the property that the clients' masks all together sum to zero. In this way, the server can sum over the OTP's from the clients to compute the sum of their inputs.

In [29], the authors construct a natural two-round computationally efficient secure aggregation protocol with semi-honest security using threshold additive homomorphic encryption (AHE). The work of [2] employs a simple additive secret sharing approach to achieve a one-round maliciously secure aggregation protocol. Their protocol is computationally and communication-efficient, but requires two independent non-colluding servers.

Recall that the security property of a secure aggregation protocol enforces that no party learns any information about any other party's input, except for what could be inferred from the protocol output. This begs the question if we can enforce a privacy guarantee against an adversary inferring information about a party's input from the protocol output. Differential privacy (DP) [14, 15] is a statistical model of privately releasing aggregate data which masks a single party's contribution to the aggregate data. That is, DP ensures that no adversary, given access to the differentially private aggregate data, can infer any information about *any particular party's* contribution to the aggregate data. Several secure aggregation protocols [6, 7, 20] employ DP to construct a protocol in which all parties compute a differentially private sum of the clients' inputs.

Our Contributions. While differentially private secure aggregation protocols enforce client privacy against all parties, the server still learns the plaintext global model, which in many applications represents a trade secret of the consortium of clients. We propose a novel model of differentially private secure aggregation, called *client-private secure aggregation*, in which the server computes an encrypted global model that only the clients can decrypt. Client-private secure aggregation protocols composed with differential privacy achieve complete input privacy for the clients, precluding model inversion and membership inference attacks against all parties. Additionally, this model enforces a stronger security guarantee against the server, namely that an adversarial server learns no information about any client's input, not even the global model. We construct a novel client-private secure aggregation protocol that is secure against a semi-honest adversary, and relies only on a trusted third-party (TTP) for homomorphic encryption key management. We also provide a theoretical and empirical evaluation of our protocol, and compare it to another protocol which can be constructed from a modification to the secure aggregation protocols of [3, 7]. If $m \in \mathbb{N}$ is the dimension of each client's input vector to the protocol, then we define $m' = m'(m, n) \in \mathbb{N}$ as the dimension of the server's output ciphertext vector. Our novel protocol achieves a constant-rate output ciphertext vector dimension of $m' = m$, while the protocol modified from [3, 7] has a quadratic-rate output ciphertext vector dimension of $m' = \mathcal{O}(mn^2)$. Additionally, our experimental results demonstrate the practicality of our novel protocol, achieving client and server running times of less than 19 s and 2 s, respectively, when scaled to $n = 250$. We remark that we limit the scope of this work to studying the client-private secure aggregation protocol itself. A plethora of prior works ([2, 6, 7, 29], and many more) have shown how to use PPFL with different secure aggregation protocols and DP to train high-quality models on popular standard datasets, and so it's clear that client-private secure aggregation protocols can be similarly applied in this manner.

2 Client-Private Secure Aggregation

In this section, we construct a novel client-private secure aggregation protocol. See Appendix C for proofs of correctness and security in the semi-honest model. We additionally provide a theoretical comparison of our protocol against another protocol which can be constructed from a modification to the secure aggregation protocols of [3, 7] (this modification of [3, 7] is detailed in Appendix D).

Preliminaries. If $k \in \mathbb{N}$, then we denote by $[k]$ the set $\{1, 2, \ldots, k\}$. If $q \in \mathbb{N}$, then we write $\mathbb{Z}_q$ for the ring of integers $(\bmod\ q)$. For $m \in \mathbb{N}$, we denote vectors in $\mathbb{Z}_q^m$ by bold lower-case characters $\mathbf{x}$. If $\mathbf{x} \in \mathbb{Z}_q^m$, then we denote the i^{th} component of $\mathbf{x}$ by $x_i \in \mathbb{Z}_q$. If $x_1, \ldots, x_m \in \mathbb{Z}_q$, then we write $(x_i)_{i \in [m]}$ for the vector in $\mathbb{Z}_q^m$ whose i^{th} component is x_i. Sets are written as upper-case bold characters $\mathbf{S}$, algorithms are written as $\mathcal{A}$, and probabilistic distributions are written as D. Throughout this work, we denote the security parameter by $\lambda \in \mathbb{N}$. A quantity $f(\lambda)$ is said to be negligible in λ, written $f(\lambda) = \mathrm{negl}(\lambda)$, if $f(\lambda)$ asymptotically tends to zero faster than any inverse polynomial in λ. A quantity $f(\lambda)$ is said to be polynomial in λ if $f(\lambda) = \mathcal{O}(\lambda^c)$, for some constant $c \in \mathbb{N}$. We say that two distributions X and Y are statistically indistinguishable, written $\mathsf{X} \equiv \mathsf{Y}$, if for every probabilistic algorithm $\mathcal{A}$ which gives output in $\{0, 1\}$, it holds that

$$\left| \Pr_{x \sim \mathsf{X}}\left[\mathcal{A}(1^\lambda, x) = 1\right] - \Pr_{y \sim \mathsf{Y}}\left[\mathcal{A}(1^\lambda, y) = 1\right] \right| = \mathrm{negl}(\lambda). \quad (1)$$

In the aforementioned definition, if we instead restrict $\mathcal{A}$ to be a probabilistic polynomial time (PPT) algorithm, then we say that X and Y are computationally indistinguishable, and write $\mathsf{X} \approx_c \mathsf{Y}$. See Appendix A for further cryptographic preliminaries.

Client-Private Secure Aggregation. Let $\lambda \in \mathbb{N}$ be the security parameter and $n = n(\lambda), q = q(\lambda), m = m(\lambda)$. A *secure aggregation protocol* is a secure multi-party computation (MPC) protocol executed among a set of parties $\mathbf{P} = \{C_1, \ldots, C_n, S\}$ consisting of n clients $C_1, \ldots, C_n$ and a server S. The protocol utilizes the star network graph in which each client C_i has an established secure communication channel with the server S. Each client C_i holds a private input $\mathbf{x}_i \in \mathbb{Z}_q^m$, the server has no input, and all parties securely compute $\mathbf{z} = \sum_i \mathbf{x}_i \in \mathbb{Z}_q^m$. In each round of the protocol, every client sends a message to the server, and the server responds with a message for each client.

Here we define a novel model of secure aggregation which we call *client-private secure aggregation*. The syntax of a client-private secure aggregation protocol Π is described in Figure 1. Intuitively, the security of the protocol enforces that no adversary that corrupts a subset of parties learns any information about any non-corrupted client's input, except for what could be inferred from the protocol output. Additionally, since the server outputs a vector of ciphertexts to each client which the clients decrypt to reconstruct $\mathbf{z}$, then the security of the associated encryption scheme implies that the server learns no information about any client's input, not even the sum of their inputs. See Appendix B for a formal definition of the security model.

Adding Differential Privacy. We briefly describe a generic method to integrate differential privacy into a client-private secure aggregation protocol. This technique follows the approach described in [29]. For an overview of DP, see Section A.0.6 in Appendix A. We employ the Gaussian mechanism, in which the degree of DP enforced is characterized by two parameters $\varepsilon, \delta > 0$. The Gaussian mechanism works by adding to the sum $\sum_i \mathbf{x}_i \in \mathbb{Z}_q^m$ of the clients' inputs a vector of independently generated Gaussian samples $\mathbf{e} \leftarrow \mathsf{N}_\sigma^m$, where N_σ denotes the Gaussian distribution centered at 0

Notation: Let $\lambda \in \mathbb{N}$ be the security parameter and $n = n(\lambda), q = q(\lambda), m = m(\lambda), m' = m'(\lambda) \in \mathbb{N}$. The protocol participants are n clients $C_1, \ldots, C_n$ and a server S. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme with ciphertext space $\mathbf{G}$.

Input: Each client C_i $(i \in [n])$ receives as input $\mathbf{x}_i \in \mathbb{Z}_q^m$; the server S has no input.

Output: The server S computes a vector $\mathbf{c} \in (\mathbf{G}^{m'})^n$ of ciphertexts and outputs to each client C_i the vector $\mathbf{c}_i \in \mathbf{G}^{m'}$; each client then outputs $\sum_{i=1}^{n} \mathbf{x}_i \in \mathbb{Z}_q^m$.

Figure 1: The syntax of a client-private secure aggregation protocol Π.

with variance $\sigma > 0$. It follows that when $\varepsilon \in (0, 1)$, $\delta > 0$ and $\sigma > \Delta_2 \sqrt{\log(25/(16\delta))}/\varepsilon$, where Δ_2 denotes the ℓ_2-sensitivity[1] of the sum function, then the Gaussian mechanism with variance σ achieves $(\varepsilon, \delta)-$DP [15]. It is a well-known fact that if $e_1 \leftarrow \mathsf{N}_{\sigma_1}$ and $e_2 \leftarrow \mathsf{N}_{\sigma_2}$ $(\sigma_1, \sigma_2 > 0)$, then $e_1 + e_2 \leftarrow \mathsf{N}_{\sigma_1+\sigma_2}$. Hence if each client C_i transforms its input vector $\mathbf{x}_i$ in the protocol to $\mathbf{x}'_i := \mathbf{x}_i + \mathbf{e}_i$, for $\mathbf{e}_i \leftarrow \mathsf{N}^m_{\sigma/n}$, then the protocol will output $\sum_i \mathbf{x}_i + \sum_i \mathbf{e}_i$, where $\sum_i \mathbf{e}_i \leftarrow \mathsf{N}^m_{\sigma}$, as desired.

An Explicit Construction. We now construct a novel client-private secure aggregation protocol Π_{A} which achieves semi-honest security. At a high level, the protocol works by each client C_i first receiving a public/secret key pair $(\mathsf{pk}, \mathsf{sk})$ for an additive homomorphic encryption (AHE) scheme from a TTP. Next, each client C_i begins by splitting their input $\mathbf{x}_i \in \mathbb{Z}_q^m$ into n additive secret shares $\{\mathbf{s}_{i,j}\}_{j\in[n]} \subseteq \mathbb{Z}_q^m$, one for every other client, and distributing each share $\mathbf{s}_{i,j}$ to Client C_j by way of the server. Now, each client C_i holds shares $\{\mathbf{s}_{j,i}\}_{j\in[n]} \subseteq \mathbb{Z}_q^m$, and sums over the shares to compute $\mathbf{t}_i = \sum_{j\in[n]} \mathbf{s}_{j,i} \in \mathbb{Z}_q^m$, which is a share of the sum $\mathbf{z} = \sum_{r\in[n]} \mathbf{x}_r \in \mathbb{Z}_q^m$. Each client C_i then uses the AHE scheme to encrypt their share $\mathbf{t}_i$ under pk, obtaining $\mathbf{c}_i$, and sends $\mathbf{c}_i$ to the server. The server homomorphically reconstructs $\mathbf{z}$ by homomorphically adding $\{\mathbf{c}_i\}_{i\in[n]}$, obtaining a ciphertext $\mathbf{c}'$ of $\mathbf{z}$. The server then outputs $\mathbf{c}'$ to each client, which uses sk to decrypt $\mathbf{c}'$ to $\mathbf{z}$. The full protocol description is detailed in Figure 2. See Appendix C for proofs of correctness and security.

Remark. Note that in our protocol Π_{A}, each client C_i holds the same AHE secret key sk. Although this is typically non-standard in homomorphic encryption solutions to $n-$party secure MPC protocols, in this application the AHE scheme is used to protect the protocol output, which is learned by all clients, from the server. If an adversary controlling the server corrupts a client, then the AHE secret key sk falls into the view of the adversary. Thus the adversary learns all of the clients' shares of the sum $\mathbf{z} \in \mathbb{Z}_q^m$, and hence the plaintext sum $\mathbf{z}$ falls into the adversary's view. But, since the adversary has corrupted the client, then $\mathbf{z}$ already falls into its view, and so no further information about any non-corrupted client's input is revealed to the adversary.

Theoretical Evaluation. A close inspection of the full protocol description of Π_{A}, detailed in Appendix C, reveals that the client and server computational complexities are $\mathcal{O}(mn)$ and $\mathcal{O}(mn^2)$, respectively. The client's computational complexity is dominated in Round 3 by summing across n shares, each of which is an $m-$dimensional vector. The server's computational complexity is dominated in Round 2 by distributing to each client n ciphertext vectors of dimension m. Similarly, we can see that the client and server communication complexity is $\mathcal{O}(mn)$ and $\mathcal{O}(mn^2)$, respectively. Each client needs to store a vector of $n - 1$ public keys (one from every other client), in addition to its input and output vector of dimension m, which yields a space complexity of $\mathcal{O}(m + n)$. The server only needs to store the encrypted $m-$dimensional protocol output, which requires $\mathcal{O}(m)$ space. By inspecting the full protocol description of the client-private secure aggregation protocol Π_{B} constructed from [3, 7] (Appendix D), we can see that Π_{B} is equivalent to Π_{A} with respect to each of these criteria, except that Π_{B} is a two-round protocol (while Π_{A} requires three rounds), but

[1] The ℓ_2-sensitivity of a function $f : \mathbf{X}^r \to \mathbf{Y}^s$ $(\mathbf{X}, \mathbf{Y} \subseteq \mathbb{R})$ is defined as $\max_{\substack{\mathbf{x}_1,\mathbf{x}_2\in\mathbf{X}^r s.t. \\ \mathrm{dist}(\mathbf{x}_1,\mathbf{x}_2)=1}} \left\{ ||f(\mathbf{x}_1) - f(\mathbf{x}_2)||_2 \right\}$, where $\mathrm{dist}(\cdot, \cdot)$ denotes hamming distance.

Setup: All parties have access to the protocol security parameter $\lambda \in \mathbb{N}$, a key agreement scheme $\mathsf{KA} = (\mathsf{Gen}, \mathsf{Agree})$ with key space $\mathbf{K}$, an authenticated encryption scheme $\mathsf{AE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with ciphertext space $\mathbf{H}$, and an additive homomorphic encryption scheme $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$ with plaintext space $\mathbb{Z}_q$ and ciphertext space $\mathbf{G}$. Assume that a TTP generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{AHE.Gen}(1^\lambda)$ and sends $(\mathsf{pk}, \mathsf{sk})$ to each client C_i $(i \in [n])$.

Input: Each client C_i has a private input $\mathbf{x}_i \in \mathbb{Z}_q^m$; the server S has no input.

Output: The server outputs a ciphertext $\mathbf{c}'' \in \mathbf{H}^m$ to each client C_i $(i \in [n])$; each client then outputs $\sum_{i=1}^{n} \mathbf{x}_i \in \mathbb{Z}_q^m$.

Round 1:

• $\mathrm{C}_i \to S$: Generate $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KA.Gen}(1^\lambda)$, and output pk_i.

• $\mathrm{S} \to C_i$: Output $(\mathsf{pk}_j)_{j=1}^n$.

Round 2:

• $\mathrm{C}_i \to S$: For each $j \in [n]\backslash\{i\}$, compute $\mathsf{k}_{i,j} \leftarrow \mathsf{KA.Agree}(\mathsf{sk}_i, \mathsf{pk}_j)$. For all $j \in [n-1]$, choose $\mathbf{s}_{i,j} \leftarrow \mathbb{Z}_q^m$, and let $\mathbf{s}_{i,n} = \mathbf{x}_i - \sum_{j \in [n-1]} \mathbf{s}_{i,j} \in \mathbb{Z}_q^m$. For each $j \in [n]\backslash\{i\}$, perform the following: for all $k \in [m]$, compute $c_{i,j,k} \leftarrow \mathsf{AE.Enc}(\mathsf{k}_{i,j}, s_{i,j,k})$, and let $\mathbf{c}_{i,j} = (c_{i,j,k})_{k \in [m]} \in \mathbf{H}^m$. Output $(\mathbf{c}_{i,j})_{j \in [n]\backslash\{i\}}$.

• $\mathrm{S} \to C_i$: Receive $(\mathbf{c}_{i,j})_{j \in [n]\backslash\{i\}}$ from each client C_i $(i \in [n])$. Output $(\mathbf{c}_{j,i})_{j \in [n]\backslash\{i\}}$ to each client C_i.

Round 3:

• $\mathrm{C}_i \to S$: For each $j \in [n]\backslash\{i\}$, perform the following: for all $k \in [m]$, compute $s^*_{j,i,k} \leftarrow \mathsf{AE.Dec}(\mathsf{k}_{i,j}, c_{i,j,k})$, and let $\mathbf{s}^*_{j,i} = (s^*_{j,i,k})_{k \in [m]}$. Compute $\mathbf{t}_i = \sum_{j \in [n]} \mathbf{s}^*_{j,i} \in \mathbb{Z}_q^m$. For each $k \in [m]$, compute $c'_{i,k} \leftarrow \mathsf{AHE.Enc}(\mathsf{pk}, t_{i,k})$. Let $\mathbf{c}'_i = (c'_{i,k})_{k \in [m]} \in \mathbf{G}^m$, and output $\mathbf{c}'_i$.

• $\mathrm{S} \to C_i$: Let $\mathbf{c}'' := \mathbf{c}'_1$. For all $i \in \{2, \ldots, n\}, k \in [m]$, update $c''_k \leftarrow \mathsf{AHE.Add}(c''_k, c'_{i,k})$. Output $\mathbf{c}''$ to each client C_i.

• C_i : Receive $\mathbf{c}''$. For all $k \in [m]$, compute $z_k \leftarrow \mathsf{AHE.Dec}(\mathsf{sk}, c''_k)$. Output $\mathbf{z} = (z_k)_{k \in [m]} \in \mathbb{Z}_q^m$.

Figure 2: Protocol Π_{A}

Π_{B} has a server space complexity of $\mathcal{O}(mn^2)$ (while that of Π_{A} is $\mathcal{O}(m)$). Note that in Π_{A}, the server outputs to each client a single $m-$dimensional vector of ciphertexts, while in Π_{B}, the server outputs to each client an $(n-1)-$sized set of $m-$dimensional ciphertext vectors.

3 Experimental Results

In this section, we empirically evaluate our client-private secure aggregation protocol Π_{A} with respect to its running time, and communication and space overhead. Additionally, we compare its performance Π_{B} across these criteria. We implemented both protocols in Python, using Elliptic Curve Diffie-Hellman for a key agreement scheme, AES-GCM for an authenticated encryption scheme, AES-CTR for a pseudorandom generator, and Paillier Encryption [26] for an additive homomorphic encryption scheme. For each protocol construction, we conducted the following experiments:

- Measure running time for client and server for number n of clients, dimension m of clients' input vector, and modulus q when $n \in \{10, 50, 100, 250\}$, $m = 100$, $q = 2^{128}$.
- Measure communication and space overhead for client and server when $n = 100, m = 100, q = 2^{128}$.

All experiments were run on a MacBook Pro with Intel Core i7 6-core 2.6 GHz CPU, and each party was simulated as a sub-process. Our experiments only measure the local performance of the

	Client		**Server**	
n	Π_{A}	Π_{B}	Π_{A}	Π_{B}
50	16.763	0.458	0.195	0.000690
100	16.990	0.934	0.415	0.00659
250	18.0570	2.334	1.0150	0.0342

Table 1: Client and server running times for $n \in \{50, 100, 250\}$ $(m = 100)$.

Client		
	Comm Overhead (KB)	**Space Overhead (KB)**
Π_{A}	925.239	21.391
Π_{B}	868.029	21.303
Server		
	Comm Overhead (MB)	**Space Overhead (MB)**
Π_{A}	92.996	5.720
Π_{B}	87.751	86.791

Table 2: Client and server communication and space overhead $(n = 100, m = 100)$.

protocol, and in particular ignore network latency. Table 1 compares the running times vs. number $n \in \{50, 100, 250\}$ of clients between the two protocols for the client and server, respectively. We can see that while our theoretical analysis of the computational complexity of protocols Π_{A} and Π_{B} indicates they are identical, in reality the running time of Π_{A} for both the client and server is noticeably higher than that of Π_{B}. This is due to the cost of the homomorphic operations of Paillier Encryption. However, note that the running times for Π_{A} are still practical, with the client and server obtaining running times of roughly 18 s and 1 s, respectively, even when scaled to 250 clients.

Table 2 displays the client and server communication and space overhead for each protocol construction when $n = 100$ and $m = 100$. We remark that for both the client and server, the communication and space overhead between the two protocol constructions is quite comparable, except that the server's space overhead of 5.72 MB in Π_{A} is significantly lower than for Π_{B} (86.791 MB). This is because in Π_{A}, the server outputs to each client a single $m-$dimensional vector of ciphertexts, while in Π_{B}, the server outputs to each client an $(n-1)-$sized set of $m-$dimensional ciphertext vectors.

4 Conclusions and Future Work

In this work, we propose a novel model of secure aggregation, called client-private secure aggregation, in which the server computes an encrypted global model that can only be decrypted by the clients. When composed with differential privacy, the security implication is that no party learns any information about any client's input. In particular, the server does not even learn the global model in the clear. We provided an explicit construction Π_{A} of a client-private secure aggregation protocol and proved its correctness and security in the semi-honest model. Finally, we empirically evaluated our construction to demonstrates its practicality, and compared it to a client-private secure aggregation protocol Π_{B} which can be obtained from a modification to [3, 7]. While the client and server running time of Π_{A} is noticeably higher than Π_{B}, and Π_{A} requires an extra round over Π_{B} and uses a TTP for key management, Π_{A} requires the server to only store a single $m-$dimensional encrypted global model. On the other hand, Π_{B} requires the server to store an $(n-1)-$sized set of $m-$dimensional ciphertext vectors for every client, yielding a storage complexity of $\mathcal{O}(mn^2)$.

There are several elements of future work which we seek to incorporate into the full version of this work. First, we believe it's possible to prove (a simple modification) of our protocol construction Π_{A} is secure against a malicious adversary. Also, we believe we can employ techniques to mitigate the client and server running time of our implementation of Π_{A}. For example, it may be possible to use the additive homomorphic version of ElGamal Encryption [17], which works over small input domains, and is more computationally efficient than Paillier Encryption. Alternatively, we wish to investigate packing Paillier ciphertexts, following [25], to improve the client and server running times.

References

[1] J. Alperin-Sheriff and C. Peikert. Faster bootstrapping with polynomial error. In J. A. Garay and R. Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 297–314. Springer, 2014.

[2] C. Beguier and E. W. Tramel. SAFER: sparse secure aggregation for federated learning. *CoRR*, abs/2007.14861, 2020.

[3] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1175–1191. ACM, 2017.

[4] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In R. Safavi-Naini and R. 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 868–886. Springer, 2012.

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electron. Colloquium Comput. Complex.*, page 111, 2011.

[6] D. Byrd, V. Mugunthan, A. Polychroniadou, and T. H. Balch. Collusion resistant federated learning with oblivious distributed differential privacy. *CoRR*, abs/2202.09897, 2022.

[7] D. Byrd and A. Polychroniadou. Differentially private secure multi-party computation for federated learning in financial applications. In T. Balch, editor, *ICAIF '20: The First ACM International Conference on AI in Finance, New York, NY, USA, October 15-16, 2020*, pages 16:1–16:9. ACM, 2020.

[8] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.

[9] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33, 2016.

[10] O. Choudhury, A. Gkoulalas-Divanis, T. Salonidis, I. Sylla, Y. Park, G. Hsu, and A. Das. Differential privacy-enabled federated learning for sensitive health data. *CoRR*, abs/1910.02578, 2019.

[11] O. Choudhury, A. Gkoulalas-Divanis, T. Salonidis, I. Sylla, Y. Park, G. Hsu, and A. Das. Anonymizing data for privacy-preserving federated learning. *CoRR*, abs/2002.09096, 2020.

[12] O. Choudhury, Y. Park, T. Salonidis, A. Gkoulalas-Divanis, I. Sylla, and A. Das. Predicting adverse drug reactions on distributed health data using federated learning. In *AMIA 2019, American Medical Informatics Association Annual Symposium, Washington, DC, USA, November 16-20, 2019*. AMIA, 2019.

[13] L. Ducas and D. Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, 2015.

[14] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith. Calibrating noise to sensitivity in private data analysis. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006.

[15] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[16] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

[17] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

[18] C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.

[19] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2013.

[20] R. C. Geyer, T. Klein, and M. Nabi. Differentially private federated learning: A client level perspective. *CoRR*, abs/1712.07557, 2017.

[21] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 464–479. IEEE Computer Society, 1984.

[22] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In D. S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 25–32. ACM, 1989.

[23] Y. Guo, A. Polychroniadou, E. Shi, D. Byrd, and T. Balch. Microfedml: Privacy preserving federated learning for small weights. *IACR Cryptol. ePrint Arch.*, page 714, 2022.

[24] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In A. Singh and X. J. Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017.

[25] T. D. T. Nguyen and M. T. Thai. Preserving privacy and security in federated learning. *CoRR*, abs/2202.03402, 2022.

[26] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[27] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, volume 4, pages 169–180, 1978.

[28] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[29] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou. A hybrid approach to privacy-preserving federated learning. In L. Cavallaro, J. Kinder, S. Afroz, B. Biggio, N. Carlini, Y. Elovici, and A. Shabtai, editors, *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2019, London, UK, November 15, 2019*, pages 1–11. ACM, 2019.

[30] B. Zhao, K. R. Mopuri, and H. Bilen. idlg: Improved deep leakage from gradients. *CoRR*, abs/2001.02610, 2020.

[31] W. Zheng, L. Yan, C. Gou, and F.-Y. Wang. Federated meta-learning for fraudulent credit card detection. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 4654–4660, 2021.

[32] L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14747–14756, 2019.

A Cryptographic Preliminaries

A.0.1 Key Agreement Scheme

Here, we define a key agreement scheme [8], which is typically used for two parties to agree on a shared key for a symmetric-key cryptosystem.

Definition 1. *A key agreement scheme is a pair of PPT algorithms* $\mathsf{KA} = (\mathsf{Gen}, \mathsf{Agree})$ *with the following syntax, correctness, and security.*

Syntax:

- $\mathsf{Gen}(1^\lambda)$ *takes as input the security parameter* λ *and outputs a public/secret key pair* $(\mathsf{pk}, \mathsf{sk})$ *for some user.*
- $\mathsf{Agree}(\mathsf{sk}_i, \mathsf{pk}_j)$ *takes as input a secret key* sk_i *corresponding to some user* i*, and a public key* pk_j*, corresponding to some user* $j \neq i$*, and outputs a key* $\mathsf{k}_{i,j}$ *from the key space* $\mathbf{K}$.

Correctness: *Let* λ *be the security parameter. If* $(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Gen}(1^\lambda), \mathsf{k}_{1,2} \leftarrow \mathsf{Agree}(\mathsf{sk}_1, \mathsf{pk}_2), \mathsf{k}_{2,1} \leftarrow \mathsf{Agree}(\mathsf{sk}_2, \mathsf{pk}_1)$*, then* $\mathsf{k}_{1,2} = \mathsf{k}_{2,1}$.

Security: *Let* λ *be the security parameter. Define the following distributions:*

- $\mathsf{D}_0(1^\lambda)$: *Compute* $(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Gen}(1^\lambda)$*,* $\mathsf{k} \leftarrow \mathsf{Agree}(\mathsf{sk}_1, \mathsf{pk}_2)$*, and output* $(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{k})$.
- $\mathsf{D}_1(1^\lambda)$: *Compute* $(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Gen}(1^\lambda)$*,* $\mathsf{k} \leftarrow \mathbf{K}$*, and output* $(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{k})$.

If $\mathcal{A}$ *is a PPT distinguishing algorithm, then* $\forall b \in \{0, 1\}$*, define*

$$\mathrm{P}_b^{\mathcal{A}}(\lambda) := \Pr_{(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{k}) \leftarrow \mathsf{D}_b(1^\lambda)}\left[\mathcal{A}(1^\lambda, \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{k}) = 1\right]. \tag{2}$$

Then, for all PPT distinguishing adversaries $\mathcal{A}$*,*

$$\left|P_0^{\mathcal{A}}(\lambda) - P_1^{\mathcal{A}}(\lambda)\right| = \mathrm{negl}(\lambda). \tag{3}$$

A.0.2 Authenticated Encryption

Authenticated encryption (AE) is a cryptographic primitive that provides confidentiality and integrity of messages exchanged between two parties which each hold a shared symmetric key.

Definition 2. *An authenticated encryption scheme is a triple of PPT algorithms* $\mathsf{AE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax, correctness, and security.*

Syntax:

- $\mathsf{Gen}(1^\lambda)$ *takes as input the security parameter* λ *and outputs a symmetric key* $\mathsf{k} \in \mathbf{K}$ *in the key space* $\mathbf{K}$.

- $\mathsf{Enc}(\mathsf{k}, m)$ *takes as input a symmetric key* $\mathsf{k} \in \mathbf{K}$*, a message* $m \in \mathbf{M}$ *in the message space* $\mathbf{M}$*, and outputs a ciphertext* $c = (c', t) \in \mathbf{H}$ *of* m *under* k *in ciphertext space* $\mathbf{H}$*.* c' *denotes the actual ciphertext of the message, while* t *denotes the message authentication code (MAC).*
- $\mathsf{Dec}(\mathsf{k}, c)$ *takes as input a symmetric key* $\mathsf{k} \in \mathbf{K}$ *and a ciphertext* $c = (c', t) \in \mathbf{H}$*, and outputs either a message* $m \in \mathbf{M}$ *or an error symbol* $\perp$*.*

Correctness: *Let* λ *be the security parameter. If* $\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda)$*,* $m \in \mathbf{M}$ *is a message,* $c \leftarrow \mathsf{Enc}(\mathsf{k}, m)$*, then* $\mathsf{Dec}(\mathsf{k}, c) = m$*.*

Semantic Security: *Let* λ *be the security parameter. If* $\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda)$*, then for every PPT distinguishing adversary* $\mathcal{A}$ *and distinct messages* $m_0, m_1 \in \mathbf{M}$*, it holds that*

$$\left|\Pr_{c \leftarrow \mathsf{Enc}(\mathsf{k}, m_0)}\left[\mathcal{A}(1^\lambda, m_0, m_1, c) = 1\right] - \Pr_{c \leftarrow \mathsf{Enc}(\mathsf{k}, m_1)}\left[\mathcal{A}(1^\lambda, m_0, m_1, c) = 1\right]\right| = \mathrm{negl}(\lambda). \quad (4)$$

Ciphertext Integrity: *Let* λ *be the security parameter. The AE scheme* $\mathsf{AE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to provide* ciphertext integrity *if every PPT adversary* $\mathcal{A}$ *can only win the following game against a computationally unbounded challenger* $\mathcal{C}$ *with probability* $\mathrm{negl}(\lambda)$*:*
Setup: $\mathcal{C}$ *computes* $\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda)$.
Query Phase: *For all* $i = 1, \ldots, r = \mathrm{poly}(\lambda)$*,* $\mathcal{A}$ *generates a message* $m_i \in \mathbf{M}$ *and send* m_i *to* $\mathcal{C}$*.* $\mathcal{C}$ *then computes and outputs to* $\mathcal{A}$ *the ciphertext* $c_i \leftarrow \mathsf{Enc}(\mathsf{k}, m_i)$*.*
Challenge Phase: $\mathcal{A}$ *produces and sends to* $\mathcal{C}$ *a ciphertext* $c' \in \mathbf{H}$*.* $\mathcal{A}$ *wins if* $c' \notin \{c_1, \ldots, c_r\}$ *and* $\mathsf{Dec}(\mathsf{k}, c') \neq \perp$.

A.0.3 Pseudorandom Generator

Definition 3. *Let* $r, s \in \mathbb{N}$ *such that* $r < s$*. A pseudorandom generator (PRG) [21, 22] is a PPT function* $G : \{0,1\}^r \rightarrow \{0,1\}^s$ *such that* $G\big(\mathsf{U}(\{0,1\}^r)\big) \approx_c \mathsf{U}(\{0,1\}^s)$*, where* $\mathsf{U}(\{0,1\}^r)$ *and* $\mathsf{U}(\{0,1\}^s)$ *denote the uniform distributions on* $\{0,1\}^r$ *and* $\{0,1\}^s$*, respectively.*

A PRG G can be used to strecth a random shared symmetric key in the following way. Let $\mathbf{K}$ be a symmetric key space and $q, m \in \mathbb{N}$ such that $\log_2(|\mathbf{K}|) < m \log_2(q)$. Then, it is easy to see that without loss of generality we can define a PRG $G : \mathbf{K} \rightarrow \mathbb{Z}_q^m$.

A.0.4 Additive Secret Sharing

Let $n, t, q \in \mathbb{N}$. A $(t, n)-$*secret sharing* scheme over $\mathbb{Z}_q$ is a pair of PPT algorithms $(\mathsf{Share}, \mathsf{Rec})$ with the following properties:

- $\mathsf{Share}(x)$ takes as input a secret $x \in \mathbb{Z}_q$ and outputs shares $\{s_i\}_{i \in [n]}$ for a set of n users, indexed by $[n]$.
- $\mathsf{Rec}(\{x_{i_j}\}_{j \in [t]})$ takes as input a subset $\{x_{i_j}\}_{j \in [t]} \subseteq \mathbb{Z}_q$ of t distinct shares of a secret $x \in \mathbb{Z}_q$, and reconstructs and outputs $x \in \mathbb{Z}_q$.
- Any subset of shares of size less than t is statistically independent of the underlying secret.

Additive secret sharing is a $(n, n)-$secret sharing scheme over $\mathbb{Z}_q$ in which $\mathsf{Share}(x)$ chooses random $s_1, \ldots, s_{n-1} \leftarrow \mathbb{Z}_q$, computes $s_n = x - \sum\limits_{i \in [n-1]} s_i \in \mathbb{Z}_q$, and outputs $\{s_i\}_{i \in [n]}$. $\mathsf{Rec}(\{s_i\}_{i \in [n]})$ simply works by outputting $\sum\limits_{i=1}^{n} s_i \in \mathbb{Z}_q$. Note that any subset of $\{s_i\}_{i=1}^n$ of size $k < n$ is distributed identically to k uniformly random elements of $\mathbb{Z}_q$, hence is statistically independent of the secret x.

A.0.5 Homomorphic Encryption

Homomorphic encryption (HE) [4, 5, 16, 18, 19, 26, 27] is a cryptographic primitive which enables computation directly on encrypted data. That is, HE is an encryption scheme which supports homomorphic addition or multiplication operations, so that a party, holding only ciphertexts of two messages m_1, m_2, can apply the homomorphic addition (resp., multiplication) operation to compute a ciphertext of m_1+m_2 (resp., $m_1 \cdot m_2$). Since an arbitrary computable function $f : \{0,1\}^* \to \{0,1\}^*$ can be expressed as an arithmetic circuit, then theoretically a HE scheme allows a client C, which holds a private input $\mathbf{x} \in \{0,1\}^*$, to outsource the computation of $f(\mathbf{x})$ to a server S without revealing any information about $\mathbf{x}$ to S. This works by C encrypting $\mathbf{x} \in \{0,1\}^*$ and sending the ciphertext to S, which can homomorphically compute a ciphertext of $f(\mathbf{x})$ which can be decrypted by C. The semantic security of the HE scheme ensures that S learns no information about $\mathbf{x}$ during the homomorphic evaluation of $f(\mathbf{x})$.

A partially homomorphic encryption (PHE) scheme is an HE scheme that supports either homomorphic addition or multiplication operations, but not both. PHE schemes are either additive homomorphic encryption (AHE) schemes or multiplicative homomorphic encryption (MHE) schemes. An example of an AHE scheme is Paillier Encryption [26], while examples of MHE schemes are RSA [28] and ElGamal Encryption [17].

A fully homomorphic encryption (FHE) scheme is a HE scheme that supports both homomorphic addition and multiplication operations. First constructed by Craig Gentry in [18], numerous follow-up works [4, 5, 16, 19] introduced improved constructions of FHE schemes. Most FHE constructions rely on a computationally expensive bootstrapping operation [1, 9, 13] to refresh the ciphertexts after a fixed-length consecutive sequence of homomorphic operations. Indeed, these bootstrapping algorithms continue to serve as the principal bottleneck in achieving FHE as a computationally practical general-purpose solution to privacy-preserving cloud-outsourced computation.

In this work, we use AHE, and so for completeness we provide a formal definition of AHE below.

Definition 4. *An additive homomorphic encryption (AHE) scheme is a quadruple of PPT algorithms* $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add})$ *with the following syntax, correctness, and security.*

Syntax:

- $\mathsf{Gen}(1^\lambda)$ *takes as input the security parameter* $\lambda \in \mathbb{N}$ *and outputs a public/secret key pair* $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Enc}(\mathsf{pk}, m)$ *takes as input a public key* pk *and message* $m \in \mathbf{M}$ *in the message space* $\mathbf{M}$*, and outputs a ciphertext* $c \in \mathbf{H}$ *in the ciphertext space* $\mathbf{H}$.
- $\mathsf{Dec}(\mathsf{sk}, c)$ *takes as input a secret key* sk *and ciphertext* $c \in \mathbf{H}$*, and outputs a message* $m \in \mathbf{M}$.
- $\mathsf{Add}(c_1, c_2)$ *takes as input two ciphertexts* $c_1, c_2 \in \mathbf{H}$ *and outputs a ciphertext* $c_3 \in \mathbf{H}$.

Correctness of Decryption: *Let* $\lambda \in \mathbb{N}$ *be the security parameter,* $m \in \mathbf{M}$ *be a message, and suppose* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$*. Then,* $\mathsf{Dec}(\mathsf{sk}, c) = m$.

Correctness of Homomorphic Addition: *Let* $\lambda \in \mathbb{N}$ *be the security parameter,* $m_1, m_2 \in \mathbf{M}$ *be a message, and suppose* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, m_i)\ \forall i \in \{1, 2\}$*, and* $c_3 \leftarrow \mathsf{Add}(c_1, c_2)$*. Then,* $\mathsf{Dec}(\mathsf{sk}, c_3) = m_1 + m_2$.

Semantic Security: *Let* $\lambda \in \mathbb{N}$ *be the security parameter, and suppose* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$*. If* $\mathcal{A}$ *is a PPT distinguishing algorithm and* $m_0, m_1 \in \mathbf{M}$ *are distinct messages, then* $\forall b \in \{0,1\}$ *define*

$$\mathrm{P}_b^{\mathcal{A}}(\lambda, \mathsf{pk}, m_0, m_1) := \mathrm{Pr}_{c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)}\big[\mathcal{A}(1^\lambda, \mathsf{pk}, m_0, m_1, c) = 1\big]. \tag{5}$$

Then, for every PPT distinguishing adversary $\mathcal{A}$ *and distinct messages* $m_0, m_1 \in \mathbf{M}$*, it holds that*

$$\big|\mathrm{P}_0^{\mathcal{A}}(\lambda, \mathsf{pk}, m_0, m_1) - \mathrm{P}_1^{\mathcal{A}}(\lambda, \mathsf{pk}, m_0, m_1)\big| = \mathrm{negl}(\lambda). \tag{6}$$

A.0.6 Differential Privacy

Differential privacy (DP) [14, 15] is a statistical model of privately releasing aggregate data which masks a single party's contribution to the aggregate data. That is, DP ensures that no adversary, given access to the differentially private aggregate data, can infer any information about *any particular party's* contribution to the aggregate data. While an adversary may infer be able to infer information about some client's contribution to the aggregate data, they can't associate that inference with a particular client.

Let $\mathbf{X}, \mathbf{Y} \subseteq \mathbb{R}$, $n, m \in \mathbb{N}$, and $f : \mathbf{X}^n \to \mathbf{Y}^m$ be a function. f is meant to model an aggregate function of data collected from n users that is represented as a database record $\mathbf{x} \in \mathbf{X}^n$. We define the function $\mathrm{dist}(\cdot,\cdot) : \mathbf{X}^n \times \mathbf{X}^n \to \mathbb{Z}$ as the hamming distance function (*i.e.,* $\mathrm{dist}(\mathbf{x}, \mathbf{x}') = \#\{i \in [n] : x_i \neq x_i'\}$). A differentially private mechanism for f is a PPT algorithm $\mathcal{M}^f$ that gets oracle access to f, takes input in $\mathbf{X}^n$, and provides output in $\mathbf{Y}^m$. We now provide a formal definition of a differentially private mechanism below.

Definition 5. *Let $\varepsilon, \delta > 0$. A PPT algorithm $\mathcal{M}^f$ is said to be an $(\varepsilon, \delta)-$differentially private mechanism for f if $\forall \mathbf{x}, \mathbf{x}' \in \mathbf{X}^n$ such that $\mathrm{dist}(\mathbf{x}, \mathbf{x}') = 1$, and $\forall \mathbf{S} \subseteq \mathrm{supp}(\mathcal{M})$,*

$$\Pr_{\mathbf{y} \leftarrow \mathcal{M}^f(\mathbf{x})}\big[\mathbf{y} \in \mathbf{S}\big] \leq e^{\varepsilon} \cdot \Pr_{\mathbf{y}' \leftarrow \mathcal{M}^f(\mathbf{x}')}\big[\mathbf{y}' \in \mathbf{S}\big] + \delta. \tag{7}$$

The parameters (ε, δ) in the definition above are said to be the privacy parameters. It is important to emphasize that a differentially private mechanism $\mathcal{M}^f$ *does not*:

- Guarantee that the output $\mathcal{M}^f(\mathbf{x})$ cryptographically hides either the aggregate data $f(\mathbf{x})$ or the input record $\mathbf{x}$.
- Guarantee that the values of $\mathcal{M}^f(\mathbf{x})$ and $\mathcal{M}^f(\mathbf{x}')$ are the same when $\mathrm{dist}(\mathbf{x}, \mathbf{x}') = 1$.

What a differentially private mechanism $\mathcal{M}^f$ *does* guarantee is if $\mathbf{x}, \mathbf{x}' \in \mathbf{X}^n$ are neighboring databases (*i.e.,* $\mathrm{dist}(\mathbf{x}, \mathbf{x}') = 1$), then the distributions $\mathcal{M}^f(\mathbf{x})$ and $\mathcal{M}^f(\mathbf{x}')$ are close. Consequently, no adversary can distinguish between the cases in which it sees $\mathcal{M}^f(\mathbf{x})$ and $\mathcal{M}^f(\mathbf{x}')$, thus masking a particular user's contribution to the aggregate data. It follows that any information the adversary could possibly infer from $\mathcal{M}^f(\mathbf{x})$ can't be associated with a particular user $i \in [n]$.

Two popular explicit constructions of differentially private mechanisms are the Laplace and Gaussian mechanisms. See [15] for details on these constructions, as well for a more complete treatment of DP.

B Security Model

Let Π be a client-private secure aggregation protocol. We define two notions of security:

S1: No information about any client's input, other than the protocol output, is revealed to any other party.

S2: No information about any client's input, not even the protocol output, is revealed to the server.

Additionally, we are concerned with the threats:

T1: A single client attempts to steal information about another client's input.

T2: The server attempts to steal information about some client's input.

T3: A subset of clients, possibly including the server, collude to attempt to steal information about another client's input.

Our security model of client-private secure aggregation enforces **S1** against against all threats, and **S2** against **T2**. Since by definition of client-private secure aggregation, each client computes the protocol output in the clear, then it is not possible to enforce **S2** against **T1** or **T3**.

We formally prove each notion of security using the standard real/ideal world paradigm. Let $\mathcal{A}$ be a PPT adversary controlling a corrupted subset $\mathbf{C} \subseteq \mathbf{P}$ of parties. We define the *view* of $\mathcal{A}$ in Π as the distribution that includes the input and random coins from each $P_i \in \mathbf{C}$, as well as the messages sent to each $P_i \in \mathbf{C}$ from the non-corrupted parties. Let $\mathcal{S}$ be a PPT simulation algorithm which simulates the view of $\mathcal{A}$ in an ideal execution of Π, without access to the non-corrupted parties' inputs. The ideal execution $\mathrm{Sim}_{\Pi,\mathbf{P},\mathbf{C},\mathcal{A},\mathcal{S}}$ of Π is defined in Figure 3. We define the following random variables:

- $\mathsf{Real}_{\Pi,\mathbf{P},\mathbf{C},\mathcal{A}}(1^\lambda, \{\mathbf{x}_i\}_{i\in[n]})$ is the view of $\mathcal{A}$ during a real execution of Π.
- $\mathsf{Ideal}_{\Pi,\mathbf{P},\mathbf{C},\mathcal{A},\mathcal{S}}\left(1^\lambda, \{\mathbf{x}_i\}_{\substack{i\in[n] s.t.\\ C_i\in\mathbf{C}}}\right)$ is the view of $\mathcal{A}$ during an ideal execution $\mathrm{Sim}_{\Pi,\mathbf{P},\mathbf{C},\mathcal{A},\mathcal{S}}$ of Π.

We say that Π is *secure against $\mathcal{A}$ controlling* $\mathbf{C}$ if there exists a PPT simulation algorithm $\mathcal{S}$ such that $\mathsf{Real}_{\Pi,\mathbf{P},\mathbf{C},\mathcal{A}}(1^\lambda, \{\mathbf{x}_i\}_{i\in[n]}) \approx_c \mathsf{Ideal}_{\Pi,\mathbf{P},\mathbf{C},\mathcal{A},\mathcal{S}}\left(1^\lambda, \{\mathbf{x}_i\}_{\substack{i\in[n] s.t.\\ C_i\in\mathbf{C}}}\right)$.

Notation: Let $\lambda \in \mathbb{N}$ be the security parameter and $n = n(\lambda), q = q(\lambda), m = m(\lambda), m' = m'(\lambda) \in \mathbb{N}$. The protocol participants are a set $\mathbf{P} = \{C_1, \ldots, C_n, S\}$ consisting of n clients $C_1, \ldots, C_n$ and a server S. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme with ciphertext space $\mathbf{G}$. $\mathcal{A}$ is a PPT adversary which controls a subset $\mathbf{C} \subseteq \mathbf{P}$ of compromised parties, and $\mathcal{S}$ is a PPT simulation algorithm which simulates the distribution of messages sent from the non-corrupted parties to the corrupted parties.

Input: Each client $C_i \in \mathbf{C}$ receives as input $\mathbf{x}_i \in \mathbb{Z}_q^m$; the server S has no input. The simulation algorithm $\mathcal{S}$ does not have access to the non-corrupted clients' inputs.

Output: The server S computes a vector $\mathbf{c} \in (\mathbf{G}^{m'})^n$ of ciphertexts and outputs to each client C_i the vector $\mathbf{c}_i \in \mathbf{G}^{m'}$; each client outputs $\sum_{i=1}^{n} \mathbf{x}_i \in \mathbb{Z}_q^m$.

Simulation: In each round of Π, every corrupted client $C_i \in \mathbf{C}$ computes its output message according to $\mathcal{A}$, and sends that message to S. Every non-corrupted client $C_j \in \{C_1, \ldots, C_n\}\backslash\mathbf{C}$ computes its output message according to $\mathcal{S}$, and sends that message to S. If S is corrupted (resp., non-corrupted), then S computes its output messages for each client according to $\mathcal{A}$ (resp., $\mathcal{S}$), and sends each client their corresponding message.

Figure 3: The ideal execution $\mathrm{Sim}_{\Pi,\mathbf{P},\mathbf{C},\mathcal{A},\mathcal{S}}$ of Π.

There are two types of adversaries we are concerned with: semi-honest and malicious adversaries. A semi-honest adversary instructs the corrupted parties to follow the protocol honestly, but attempts to infer information about the non-corrupted parties' inputs from its view. In contrast, a malicious adversary can instruct the corrupted parties to deviate from the protocol, sending arbitrary messages or dishonestly forwarding messages to parties, to attempt to infer information about the non-corrupted parties from its view. We say that Π is *secure in the semi-honest model* (resp., *secure in the malicious model*), if for every semi-honest (resp., malicious) adversary $\mathcal{A}$, and every subset $\mathbf{C} \subseteq \mathbf{P}$ of corrupted parties, Π is secure against $\mathcal{A}$ controlling $\mathbf{C}$.

C Correctness and Security of Π_{A}

Here we supply proofs of correctness and security for protocol Π_{A}.

Correctness. We begin by proving the correctness of Π_{A}, captured by Lemma 6 below.

Lemma 6. *After an execution of* Π_{A}*, the server outputs a vector of ciphertexts* $\mathbf{c}'' \in \mathbf{G}^m$ *to each client, and each client outputs* $\sum_i \mathbf{x}_i \in \mathbb{Z}_q^m$.

Proof. After the end of Round 1, each client C_i holds their secret key sk_i and a public key pk_j from every other client C_j $(j \neq i)$ for the key agreement scheme KA. So, in Round 2, each client C_i computes a shared symmetric key $\mathsf{k}_{i,j}$ with every other client C_j. C_i then splits its private input into additive secret shares $\{\mathbf{s}_{i,j}\}_{j\in[n]} \subseteq \mathbb{Z}_q^m$ for every client, encrypts each C_j's share $\mathbf{s}_{i,j}$ $(j \neq i)$ with the authenticated encryption scheme AE under the shared symmetric key $\mathsf{k}_{i,j}$, and sends the resulting ciphertexts to the server. The server then forwards to each client C_i ciphertexts of its shares $\mathbf{s}_{j,i}$ from every other client C_j, which it decrypts in Round 3 to obtain $\{\mathbf{s}_{j,i}\}_{j\in[n]}$. C_i then computes $\mathbf{t}_i = \sum_{j\in[n]} \mathbf{s}_{j,i} \in \mathbb{Z}_q^m$, which it follows is a share of $\mathbf{z} := \sum_{j\in[n]} \mathbf{x}_i \in \mathbb{Z}_q^m$. Finally, C_i uses the additive homomorphic encryption scheme AHE to encrypt $\mathbf{t}_i$ under pk, obtaining a ciphertext $\mathbf{c}'_i$, which it sends to the server. By definition of additive secret sharing, it follows that $\sum_{i\in[n]} \mathbf{t}_i = \mathbf{z} \in \mathbb{Z}_q^m$. So, the server, each holding AHE ciphertext vectors $\mathbf{c}'_1, \ldots, \mathbf{c}'_n$ of $\mathbf{t}_1, \ldots, \mathbf{t}_n$, respectively, component-wise homomorphically adds $\{\mathbf{c}'_1, \ldots, \mathbf{c}'_n\}$ to obtain a ciphertext $\mathbf{c}''$ of $\mathbf{z}$, which it outputs to each client. Each client then uses sk to decrypt $\mathbf{c}''$ to $\mathbf{z}$. $\square$

Security. We now prove that Π_{A} is secure with respect to **S1** and **S2** in the semi-honest model. Let $\mathcal{A}$ be a semi-honest adversary controlling a subset $\mathbf{C} \subseteq \mathbf{P}$ of corrupted parties. First, we note that the security **S2** of Π_{A} when $\mathbf{C} = \{S\}$ follows immediately from the semantic security of the authenticated encryption and additive homomorphic encryption schemes. It thus suffices to prove the security **S1** of Π_{A} when there exists some client $C_i \in \mathbf{C}$. Lemma 7 below completes the proof of security.

Lemma 7 (Security)**.** *Let $\mathcal{A}$ be a semi-honest adversary which corrupts a subset $\mathbf{C} \subset \mathbf{P}$ of parties containing at least one client. Then, Π_{A} is secure against $\mathcal{A}$ controlling $\mathbf{C}$.*

Proof. We may assume without loss of generality that $\mathbf{T} := \{i \in [n] : C_i \notin \mathbf{C}\} \neq \emptyset$. Let $\mathbf{z} = \sum_{i \in \mathbf{T}} \mathbf{x}_i \in \mathbb{Z}_q^m$. We'll actually make one small modification to the ideal execution of Π_{A} in this case. Although the simulation algorithm $\mathcal{S}$ is not given access to the non-corrupted parties' inputs, we will endow $\mathcal{S}$ with the sum $\mathbf{z} := \sum_{i \in \mathbf{T}} \mathbf{x}_i \in \mathbb{Z}_q^m$ of the non-corrupted parties' inputs. Note that this is without loss of generality since any adversary, given the protocol output $\sum_{i \in [n]} \mathbf{x}_i \in \mathbb{Z}_q^m$ and the corrupted parties inputs $\{\mathbf{x}_i\}_{i \notin \mathbf{T}} \subseteq \mathbb{Z}_q^m$ can efficiently compute $\mathbf{z}$. We thus replace $\mathsf{Ideal}_{\Pi_{\mathrm{A}}, \mathbf{P}, \mathbf{C}, \mathcal{A}, \mathcal{S}}\left(1^\lambda, \{\mathbf{x}_i\}_{i \notin \mathbf{T}}\right)$ with $\mathsf{Ideal}_{\Pi_{\mathrm{A}}, \mathbf{P}, \mathbf{C}, \mathcal{A}, \mathcal{S}}\left(1^\lambda, \{\mathbf{x}_i\}_{i \notin \mathbf{T}}, \mathbf{z}\right)$. We now proceed by a standard hybrid argument.

- $\mathcal{H}_0$: This hybrid is simply a real execution of Π_{A}.
- $\mathcal{H}_1$: This hybrid is the same as $\mathcal{H}_0$, except that for each non-corrupted client $C_i \in \{C_1, \ldots, C_n\} \backslash \mathbf{C}$, in Round 2, for each $C_j \in \mathbf{C}$, we set $\mathbf{s}_{i,j} \leftarrow \mathbb{Z}_q^m$. Since the view of the adversary after Round 2 contains $\{\mathbf{s}_{i,j}\}_{\substack{i \in \mathbf{T} \\ j \notin \mathbf{T}}}$, then by the security of the additive secret sharing scheme we have that $\mathcal{H}_0 \equiv \mathcal{H}_1$.
- $\mathcal{H}_2$: In this hybrid, it will be more convenient to index the non-corrupted clients by $C_{i_1}, \ldots, C_{i_r}$. For each $j \in [r-1]$, in Round 2, C_{i_j} generates shares $\{\mathbf{s}_{i_j,t}\}_{t \in [n]}$ of 0, while C_{i_r} generates shares $\{\mathbf{s}_{i_r,t}\}_{t \in [n]}$ of $\mathbf{z}'$. Similarly, by the security of the additive secret sharing scheme we have that $\mathcal{H}_1 \equiv \mathcal{H}_2$.

We define $\mathcal{S}$ by $\mathcal{H}_2$, and it follows that $\mathsf{Real}_{\Pi_{\mathrm{A}}, \mathbf{P}, \mathbf{C}, \mathcal{A}}(1^\lambda, \{\mathbf{x}_i\}_{i \in [n]}) \equiv \mathcal{H}_0 \equiv \mathcal{H}_1 \equiv \mathcal{H}_2 \equiv \mathsf{Ideal}_{\Pi_{\mathrm{A}}, \mathbf{P}, \mathbf{C}, \mathcal{A}, \mathcal{S}}\left(1^\lambda, \{\mathbf{x}_i\}_{i \notin \mathbf{T}}, \mathbf{z}\right)$. □

D Protocol Π_{B}

The protocol Π_{B}, described below, achieves a client-private secure aggregation protocol with semi-honest security through a simple modification to the two-round semi-honest secure variant of [3] (the two-round variant is described in [7]). At a high level, the protocol works by each client C_i computing a quasi-one-time pad of its private input $\mathbf{x}_i \in \mathbb{Z}_q^m$ as $\mathbf{y}_i := \mathbf{x}_i + \mathbf{r}_i \in \mathbb{Z}_q^m$, where the clients' random masks $\mathbf{r}_1, \ldots, \mathbf{r}_n \leftarrow \mathbb{Z}_q^m$ are chosen such that $\sum_i \mathbf{r}_i = \mathbf{0} \in \mathbb{Z}_q^m$. Each client C_i then simply encrypts $\mathbf{y}_i$ for every other client C_j under a shared symmetric key $\mathsf{k}_{i,j}$, and sends the ciphertexts to the server, which routes them to the appropriate clients. Each client C_i then has a set of one-time pads $\{\mathbf{y}_j\}_{j \in [n]}$ whose random masks sum to zero, hence the client computes $\mathbf{z} := \sum_{j \in [n]} \mathbf{y}_j = \sum_{j \in [n]} \mathbf{x}_j$, as desired. Figure 2 contains the full protocol description of Π_{B}.

Correctness. We prove the correctness of Π_{B} in Lemma 8 below.

Lemma 8 (Correctness)**.** *After an execution of Π_{B}, the server outputs to each client C_i $(i \in [n])$ ciphertexts $(\mathbf{c}_{j,i})_{j \in [n] \backslash \{i\}} \in \mathbf{C}^{m(n-1)}$, and each client outputs $\sum_i \mathbf{x}_i \in \mathbb{Z}_q^m$.*

Setup: All parties have access to the protocol security parameter $\lambda \in \mathbb{N}$, a key agreement scheme $\mathsf{KA} = (\mathsf{Gen}, \mathsf{Agree})$ with key space $\mathbf{K}$, a pseudorandom generator $G : \mathbf{K} \to \mathbb{Z}_q^m$, and an authenticated encryption scheme $\mathsf{AE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with ciphertext space $\mathbf{G}$.

Input: Each client C_i has a private input $\mathbf{x}_i \in \mathbb{Z}_q^m$; the server S has no input.

Output: The server outputs ciphertexts $(\mathbf{c}_{j,i})_{j \in [n] \setminus \{i\}} \in \mathbf{G}^{m(n-1)}$ to each client C_i $(i \in [n])$; each client then outputs $\sum_{i=1}^{n} \mathbf{x}_i \in \mathbb{Z}_q^m$.

Round 1:

- $\mathrm{C}_i \to S$: Generate $(\mathsf{pk}_i^{(b)}, \mathsf{sk}_i^{(b)}) \leftarrow \mathsf{KA.Gen}(1^\lambda), \forall b \in \{0,1\}$, and output $(\mathsf{pk}_i^{(0)}, \mathsf{pk}_i^{(1)})$.
- $\mathrm{S} \to C_i$: Output $\left((\mathsf{pk}_j^{(0)}, \mathsf{pk}_j^{(1)})\right)_{j=1}^{n}$.

Round 2:

- $\mathrm{C}_i \to S$: For all $j \in [n] \setminus \{i\}, b \in \{0,1\}$, compute $\mathsf{k}_{i,j}^{(b)} \leftarrow \mathsf{KA.Agree}(\mathsf{sk}_i^{(b)}, \mathsf{pk}_j^{(b)})$. For all $j \in [n] \setminus \{i\}$, compute $\mathbf{r}_{i,j} \leftarrow G(\mathsf{k}_{i,j}^{(1)})$. Let $\mathbf{y}_i = \mathbf{x}_i + \sum_{j<i} \mathbf{r}_{i,j} - \sum_{j>i} \mathbf{r}_{i,j} \in \mathbb{Z}_q^m$. For all $j \in [n] \setminus \{i\}, k \in [m]$, compute $c_{i,j,k} \leftarrow \mathsf{AE.Enc}(\mathsf{k}_{i,j}^{(0)}, y_{i,k})$. For all $j \in [n] \setminus \{i\}$, let $\mathbf{c}_{i,j} = (c_{i,j,k})_{k \in [m]} \in \mathbf{G}^m$. Output $(\mathbf{c}_{i,j})_{j \in [n] \setminus \{i\}}$.
- $\mathrm{S} \to C_i$: Store $\left((\mathbf{c}_{j,i})_{j \in [n]}\right)_{i \in [n]}$. Output $(\mathbf{c}_{j,i})_{j \in [n] \setminus \{i\}}$ to each client C_i.
- C_i : For all $j \in [n] \setminus \{i\}, k \in [m]$, compute $w_{j,k} \leftarrow \mathsf{AE.Dec}(\mathsf{k}_{i,j}^{(0)}, c_{j,i,k})$. For all $j \in [n]$, let $\mathbf{w}_j = (w_{j,k})_{k \in [m]} \in \mathbb{Z}_q^m$ if $j \neq i$, or $\mathbf{w}_j = \mathbf{y}_i$ otherwise. Output $\mathbf{z} = \sum_{j \in [n]} \mathbf{w}_j \in \mathbb{Z}_q^m$.

Figure 4: Protocol Π_{B}

Proof. In Round 1, each client C_i uses the key agreement scheme to generate two sets of public/secret key pairs $\left((\mathsf{pk}_i^{(b)}, \mathsf{sk}_i^{(b)})\right)_{b \in \{0,1\}}$, and sends $(\mathsf{pk}_i^{(0)}, \mathsf{pk}_i^{(1)})$ to the server. The server then forwards $(\mathsf{pk}_i^{(0)}, \mathsf{pk}_i^{(1)})_{i \in [n]}$ to each client. For each pair of clients (C_i, C_j) $(i \neq j)$, and for each $b \in \{0,1\}$, C_i (resp., C_j) uses their secret key $\mathsf{sk}_i^{(b)}$ (resp., $\mathsf{sk}_j^{(b)}$) and the public key $\mathsf{pk}_j^{(b)}$ (resp., $\mathsf{pk}_i^{(b)}$) of client C_j (resp., C_i) to compute a shared random key $\mathsf{k}_{i,j}^{(b)} = \mathsf{k}_{j,i}^{(b)}$.

Now, each client C_i computes $\mathbf{r}_{i,j} \leftarrow G(\mathsf{k}_{i,j}^{(1)}), \forall j \in [n] \setminus \{i\}$, $\mathbf{y}_i = \mathbf{x}_i + \sum_{j<i} \mathbf{r}_{i,j} - \sum_{j>i} \mathbf{r}_{i,j} \in \mathbb{Z}_q^m$, uses the authenticated encryption scheme to encrypt each component of $\mathbf{y}_i$ under $\mathsf{k}_{i,j}^{(0)}$ to obtain a vector of ciphertexts $\mathbf{c}_{i,j}, \forall j \in [n] \setminus \{i\}$, and outputs $(\mathbf{c}_{i,j})_{j \in [n] \setminus \{i\}}$ to the server. The server forwards $(\mathbf{c}_{j,i})_{j \in [n] \setminus \{i\}}$ to each client C_i.

Now, each client C_i uses the authenticated encryption scheme to decrypt the components of each $\mathbf{c}_{j,i}$, using $\mathsf{k}_{i,j}$, to recover $\mathbf{y}_j \in \mathbb{Z}_q^m, \forall j \in [n] \setminus \{i\}$. C_i then computes $\mathbf{z} = \sum_{j \in [n]} \mathbf{y}_j = \sum_{j \in [n]} \mathbf{x}_j + \sum_{j<k} \mathbf{r}_{j,k} + \sum_{j>k} \mathbf{r}_{j,k} = \sum_{j \in [n]} \mathbf{x}_j + \sum_{j<k} \mathbf{r}_{j,k} - \sum_{j>k} \mathbf{r}_{k,j} = \sum_{j \in [n]} \mathbf{x}_j$, since each $\mathbf{r}_{j,k} = \mathbf{r}_{k,j}$. $\square$

Security. We now prove that Π_{B} is secure with respect to **S1** and **S2** in the semi-honest model. Let $\mathcal{A}$ be a semi-honest adversary controlling a subset $\mathbf{C} \subseteq \mathbf{P}$ of corrupted parties. First, we note that the security **S2** of Π_{B} when $\mathbf{C} = \{S\}$ follows immediately from the semantic security of the authenticated encryption scheme. So, it suffices to prove the security **S1** of Π_{B} when there exists some client $C_i \in \mathbf{C}$. Lemma 9 below completes the security proof.

Lemma 9 (Security). *Let $\mathcal{A}$ be a semi-honest adversary which corrupts a subset $\mathbf{C} \subseteq \mathbf{P}$ of parties such that some client $C_i \in \mathbf{C}$. Then, Π_{B} is secure against $\mathcal{A}$ controlling $\mathbf{C}$.*

Proof. Let $\mathbf{C} \subseteq \mathbf{P}$, and $\mathsf{Real}_{\Pi_{\mathrm{B}},\mathbf{P},\mathbf{C},\mathcal{A}}(1^\lambda, \{\mathbf{x}_i\}_{i\in[n]})$ denote the distribution of the view of $\mathcal{A}$ in a real execution of Π_{B} in which $\mathcal{A}$ corrupts $\mathbf{C}$. We'll construct a PPT simulation algorithm $\mathcal{S}$ which simulates the view of $\mathcal{A}$ without access to the non-corrupted clients' inputs. By definition of the ideal execution of Π_{B} (Figure 3), this completes the proof. Let $\mathbf{T} = \{i \in [n] : C_i \notin \mathbf{C}\}$. We may assume WLOG that $\mathbf{T} \neq \emptyset$.

Just as in the proof of Lemma 9, we may endow $\mathcal{S}$ with the sum $\mathbf{z} := \sum_{i\in\mathbf{T}} \mathbf{x}_i \in \mathbb{Z}_q^m$ of the non-corrupted parties' inputs. We thus replace $\mathsf{Ideal}_{\Pi_{\mathrm{B}},\mathbf{P},\mathbf{C},\mathcal{A},\mathcal{S}}\left(1^\lambda, \{\mathbf{x}_i\}_{i\notin\mathbf{T}}\right)$ with $\mathsf{Ideal}_{\Pi_{\mathrm{B}},\mathbf{P},\mathbf{C},\mathcal{A},\mathcal{S}}\left(1^\lambda, \{\mathbf{x}_i\}_{i\notin\mathbf{T}}, \mathbf{z}\right)$.

We now proceed by a standard hybrid argument.

- $\mathcal{H}_0$: This hybrid is simply a real execution of Π_{B}.
- $\mathcal{H}_1$: For each $C_i \in \{C_1, \ldots, C_n\} \backslash \mathbf{C}$, we choose $\mathbf{r}_i \leftarrow \mathbb{Z}_q^m$ such that $\sum_{i\in\mathbf{T}} \mathbf{r}_i = \mathbf{z} \in \mathbb{Z}_q^m$, and C_i instead lets $\mathbf{y}_i := \mathbf{r}_i \in \mathbb{Z}_q^m$. Note that since the adversary corrupts some client C_j, then each symmetric key $\mathsf{k}_{i,j}^{(0)}$ ($i \in [n]$ s.t. $C_i \notin \mathbf{C}$) falls into the adversary's view, hence so does each $\mathbf{y}_i$. By Lemma 6.1 in [3], we have that $\mathcal{H}_0 \equiv \mathcal{H}_1$.

We define $\mathcal{S}$ by $\mathcal{H}_1$, and it follows that $\mathsf{Real}_{\Pi_{\mathrm{B}},\mathbf{P},\mathbf{C},\mathcal{A}}(1^\lambda, \{\mathbf{x}_i\}_{i\in[n]}) \equiv \mathcal{H}_0 \equiv \mathcal{H}_1 \equiv \mathsf{Ideal}_{\Pi_{\mathrm{B}},\mathbf{P},\mathbf{C},\mathcal{A},\mathcal{S}}\left(1^\lambda, \{\mathbf{x}_i\}_{i\notin\mathbf{T}}, \mathbf{z}\right)$. $\square$