# PROV-FL: Privacy-preserving Round Optimal Verifiable Federated Learning

Vishnu Asutosh Dasu*
The Pennsylvania State University
State College, Pennsylvania, USA
vdasu@psu.edu

Sumanta Sarkar
University of Warwick
Coventry, United Kingdom
sumanta.sarkar@warwick.ac.uk

Kalikinkar Mandal
University of New Brunswick
Fredericton, Canada
kmandal@unb.ca

## ABSTRACT

Federated learning is a distributed framework where a server computes a global model by aggregating the local models trained on users' private data. However, for a stronger data privacy guarantee, the server should not access the local models except the aggregated one. One way to achieve this is to use a secure aggregation protocol that comes with the cost of several rounds of interactions between the server and users in the absence of a fully trusted third party (TTP). In this paper, we present PROV-FL, an efficient privacy-preserving federated learning training system that securely aggregates users' local models. PROV-FL requires only one round of communication between the server and users for aggregating local models without a TTP. Based on the homomorphic encryption and differential privacy techniques, we develop two PROV-FL training protocols for two different, namely single and multi-aggregator, scenarios. PROV-FL enjoys the verifiability feature in which the server can verify the authenticity of the aggregated model and efficiently handles users' dynamic joining and leaving. We evaluate and compare the performance of PROV-FL by running experiments on training CNN/DNN models with a diverse set of real-world datasets.

## CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; • **Computing methodologies → Multi-agent systems**.

## KEYWORDS

Federated learning, privacy, machine learning, aggregated authenticator, homomorphic encryption

**ACM Reference Format:**
Vishnu Asutosh Dasu, Sumanta Sarkar, and Kalikinkar Mandal. 2022. PROV-FL: Privacy-preserving Round Optimal Verifiable Federated Learning . In *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security (AISec '22), November 11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3560830.3563729

*Work was done while the author was at TCS Research, India


*AISec '22, November 11, 2022, Los Angeles, CA, USA.*

ACM ISBN 978-1-4503-9880-0/22/11…$15.00
https://doi.org/10.1145/3560830.3563729

## 1 INTRODUCTION

The past decade has seen a significant progress in deep learning and their usage in various applications such as health and finance. Privacy-preserving machine learning (PPML) aims to protect the privacy of sensitive data and the model. *Federated learning (FL)* is a collaborative/distributed machine learning framework in which the server, coordinating the training process, holds the model, and the users locally compute gradients on the dataset, called local gradients, followed by the server computes the updated model by aggregating the local gradients from the users [27]. As the data never leaves local devices, therefore, it provides some level of data privacy. Well-known real-world scenarios of the cloud-edge settings include mobile applications like 5G, where base-stations are edge servers. Federated learning is well-suited for these applications where a cloud server holds the model and the end-devices can perform training locally on their data.

In recent years, various types of attacks such as Inference attack [39] and model inversion attack [17] have emerged that have tried to compromise the privacy of data that is used to train an ML model. In view of these attacks, PPML protocols have been developed. Existing techniques for designing PPML protocols can be broadly classified into four categories: 1) secure multi-party computation techniques, e.g., [3, 11, 13, 28, 29, 33], 2) homomorphic encryption, e.g., [24, 26, 35], 3) differential privacy and homomorphic encryption or secure aggregation, e.g., [9, 40, 42], and 4) leveraging trusted execution environments (e.g., Intel-SGX), e.g., [18, 30]. In the private training using secure multi-party computation, the training data is shared using a secret-sharing protocol among a small set of servers (e.g., 2, 3 or 4-server) (e.g., [11, 28, 29]), and then the training is conducted and the model is secret-shared among participating servers. While secure multi-party computation based techniques provide strong privacy guarantees, users need to upload their data in the secret-shared form, which is opposed to the federated learning setting of keeping private data locally. Thus, there is a risk that if those servers ever collude data privacy will be violated. Moreover, for a large dataset, secure multiparty computation based techniques incur significant computational and communication overheads. In this paper, we develop a private, verifiable, and robust federated learning system. Our key focus, over existing work, is to maintain a good balance among the efficiency, accuracy, and privacy, while providing robustness in the training process.

**Our Contributions.** With that in focus, we design PROV-FL, a communication efficient and strongly secure system for training an ML model in the federated learning setting. Towards the design of PROV-FL, we make two-fold contributions. First, we propose a secure, verifiable and robust model aggregation protocol with

single round in the presence of dynamic user participations or dropouts. Compared to multi-round secure aggregation solutions (e.g., [9, 26]) and solutions that are aided by a fully trusted third party (TTP) (e.g., [42]), PROV-FL achieves a single-round ML model aggregation, without a fully TTP, that is robust against dropouts. Second, we construct the PROV-FL training protocols by combining our new model aggregation protocol in the federated training along with the differential privacy mechanism to provide a good balance between efficiency, privacy and accuracy in training an ML model. The construction of PROV-FL is based on computationally cheap cryptographic primitives, namely additive homomorphic encryption (AHE) (e.g., JL [6]), symmetric-key authenticated encryption (AE), multi-key linearly homomorphic authenticators (MLHA) and differential privacy (DP) operations, where AHE, AE and DP provide the model privacy and the MLHA assures the ML model verifiability guarantee. We provide two training protocols for two different settings, namely single ($\Pi_{\text{GLOBAL\_TRAIN}}$) and multi-aggregator ($\Pi_{\text{MAFL\_TRAIN}}$).

We prove the security of our protocols in the simulation paradigm under the semi-honest adversarial model, where we consider three different colluding scenarios among users, the aggregator(s) and the server. The security of our protocols is based on the (standard) indistinguishability of the AHE, AE, homomorphic authenticator and differential privacy assumptions. Our protocols assure honest participants' input data privacy, and the model privacy against internal and external adversaries.

We implement PROV-FL using `gmpy2`, Charm [4], ECDSA and AES-GCM/CBC in Python using multiprocessing to leverage the state-of-the-art libraries and achieve the best efficiencies out of the available resources. To demonstrate PROV-FL's applicability, we evaluate it in terms of running time and accuracy by conducting extensive experiments on different deep and convolutional neural network (DNN/CNN) models on four different real-world datasets namely MNIST, Fashion MNIST, IMDB Movie Reviews, and Rice Image from [21, 25, 41], and [20], respectively. We present experimental results on the training time of PROV-FL for the single-aggregator setting of without privacy, no differential privacy, and with local DP. For instance, the timing overhead due to cryptographic techniques in PROV-FL with 25 users and a CNN trained on the MNIST dataset, compared to no privacy one, is $\approx 425\times$. Our results show that training in PROV-FL does not compromise users' privacy while achieving above 90% accuracy (with $(0.5, 1\times10^{-5})$-DP noise) and reduced communication and computational complexities. Compared with HybridAlpha [42] which uses Multi Input Functional Encryption (MIFE) [2] for secure aggregation, PROV-FL's execution time is about 103 seconds per user for 118, 110 gradients while HybridAlpha takes 326 seconds per user for the MIFE encryption (using the CiFEr Project's MIFE implementation). A detailed comparison between PROV-FL and HybridAlpha is provided in Section 5.4. Other related works are presented in Section 6.

**Outline.** The rest of the paper is organized as follows. In Section 2, we describe the background and cryptographic primitives that we use in our solution. Section 3 describes the system model. In Figure 2 of Section 4, we provide our protocol and its generalization is discussed in Section 4.2. We provide extensive experimental results in Section 5. We discuss related works in Section 6 followed by concluding remarks in Section 7. Due to page limit, we provide the security proofs in Appendix.

# 2 PRELIMINARIES

## 2.1 Federated learning

*Federated Learning.* The term Federated Learning was coined by McMahan et al. [27], where they proposed a framework to train a machine learning model in a distributed manner with training data distributed across multiple devices. The ML model held by the server is updated by aggregating locally-computed updates on separate user devices. The federated averaging algorithm is applied on the locally trained models from a subset of users. The model computation that is an iterative process can be mathematically defined as follows. Suppose there are $n$ users and each user $U_i$ has a dataset $\mathcal{D}_i$, and the server holds the ML model $\boldsymbol{\theta}$. In federated learning, the server sends the model $\boldsymbol{\theta}$ to the users and each user performs the gradient descent computation on the dataset $\mathcal{D}_i$ as $J_i(\mathcal{D}_i, \theta) = \frac{1}{d_i}\sum_{(\mathbf{x},y)\in\mathcal{D}_i} C(\boldsymbol{\theta}, (\mathbf{x}, y))$, where $C()$ is the cost function and $d_i = |\mathcal{D}_i|$. The local model for user $U_i$ is computed as $\boldsymbol{\theta}^i \leftarrow \boldsymbol{\theta} - \eta\nabla J(\mathcal{D}_i, \boldsymbol{\theta})$, where $\eta$ is the learning rate. After receiving the local models from the users, the server performs an aggregated averaging on the local models to construct the global model as:

$$\boldsymbol{\theta} = \frac{1}{d}\left(d_1\boldsymbol{\theta}^1 + \ldots + d_n\boldsymbol{\theta}^n\right) \tag{1}$$

where $d = \sum_{i=1}^{n} d_i$ is the total size of the dataset $\mathcal{D} = \cup_{i=1}^{n}\mathcal{D}_i$, and $\boldsymbol{\theta}^i$ is the locally trained model on the dataset $\mathcal{D}_i$. This computation is repeated until the model converges.

## 2.2 Cryptographic primitives used in PROV-FL

*2.2.1 Multi-key Linearly Homomorphic Authenticators.* A context hiding homomorphic authentication scheme guarantees that the authenticator of the output of the function does not leak any information about its inputs to the verifier. We use the context hiding multi-key homomorphic authenticator scheme MLHA defined in [36], which facilitates the evaluation of linear functions on inputs held by different users. The scheme provides context hiding guarantees in case of internal adversaries who can corrupt any of the users involved in the computation and external adversaries who do not have this information. Additionally, all parties involved in the computation can ensure that the evaluator has correctly computed the function since the scheme supports public verification. The MLHA scheme supports the evaluation of any linear function $f : \mathcal{M}^n \rightarrow \mathcal{M}$ where $\mathcal{M} \in \mathbb{Z}_p^T$ over a subset of users where each user may contribute more than one message. In the federated learning setting $f$ (Equation 1) performs the simple aggregation of user models, $n$ is the maximum number of users that can participate and each user contributes only one message.

*2.2.2 Additive Homomorphic Encryption.* An Additive Homomorphic Encryption (AHE) scheme $(E, D)$ is a public-key cryptosystem that supports the linear evaluation of plaintext messages using the corresponding ciphertexts. Given two plaintext messages $m_1, m_2$, their corresponding ciphertexts $E(m_1), E(m_2)$ and a constant $k$, one can evaluate encryption of plaintext addition such that $D(E(m_1)\cdot E(m_2)) = m_1 + m_2$ and $D(E(m_1)^k) = k\cdot m_1$.

We use the Joye-Libert (JL) cryptosystem [6] to realize an AHE scheme. Compared to the Pailler cryptosystem [31], the ciphertexts generated by the JL cryptosystem are 2× smaller for the same security parameter.

*2.2.3 Authenticated Encryption.* Authenticated encryption with associated data is a symmetric key encryption that consists of three algorithms (AE.KeyGen, AE.Enc, AE.Dec), where AE.KeyGen is a symmetric key generation algorithm, AE.Enc is a symmetric-key encryption algorithm that accepts a key $k \xleftarrow{R}$ AE.KeyGen and a message and associated data (AD) as input and outputs a ciphertext and a tag. AE.Dec is a decryption algorithm that takes a key, an AD, a ciphertext and a tag, and outputs the original message or $\perp$. In the implementation, we use AES-GCM that offers the IND-CPA and INT-CTXT security guarantees.

*2.2.4 Key Derivative Function.* Key Derivative Function (*KDF*) function is a function with which an input key and other input data are used to generate keying material that can be employed by cryptographic algorithms. One can use HMAC-based KDF in this regard.

*2.2.5 Differential Privacy.* Differential privacy [16] is a mathematical framework that enables us to quantify the information leakage of data held by individuals who participate in a study or analysis. An algorithm is defined as differentially private if it is not possible to tell whether a single datapoint has been used in the analysis or not. A formal definition of DP is as follows:

*Definition 2.1 (Differential Privacy [16]).* A randomized algorithm $\mathcal{M}$ is $(\epsilon, \delta)$-differentially private ($(\epsilon, \delta)$-DP) if for all $\mathcal{S} \subseteq \text{Range}(\mathcal{M})$ and for all datasets $D, D' \in \mathcal{D}$ differing on at most one element, the following condition holds:

$$Pr[\mathcal{M}(D) \in \mathcal{S}] \leq \exp(\epsilon) \cdot Pr[\mathcal{M}(D') \in \mathcal{S}] + \delta, \tag{2}$$

where the probability space is over the coin flips of $\mathcal{M}$. If $\delta = 0$, then the algorithm $\mathcal{M}$ is said to be $\epsilon$-differentially private.

A DP mechanism provides a privacy guarantee by adding some noise to the output of an algorithm or query $f$ applied on a database. The Laplace, exponential, and Gaussian distributions are common mechanisms for sampling noises. The Gaussian mechanism for a dataset $D \in \mathcal{D}$ and the function $f : \mathcal{D} \to \mathbb{R}$ is defined as $\mathcal{M}(D) = f(D) + \mathcal{N}(0, \sigma^2 S_f^2)$, where $\mathcal{N}(0, \sigma^2 S_f^2)$ is a Gaussian distribution with mean 0 and standard deviation $\sigma S_f$. Here, $S_f$ is the $\ell_2$ sensitivity of $f$ which is defined as $\max \|f(D) - f(D')\|_2$, for datasets $D, D' \in \mathcal{D}$ differing on at most one element. The sensitivity $S_f$ and the noise scale $\sigma$ are calibrated according to a desired privacy level. A single application of the Gaussian mechanism to $f$ with sensitivity $S_f$ satisfies $(\epsilon, \delta)$-DP if $\sigma \geq \sqrt{2 \log \frac{1.25}{\delta}}/\epsilon$ and $\epsilon \in (0, 1)$ [16].

*Differential Privacy in Federated Learning.* Using the Gaussian DP mechanism described above, if each user adds the noise $\mathcal{N}(0, \sigma^2 S_f^2)$ independently, assuming there are $n$ users, the total noise adds up to $\mathcal{N}(0, n \cdot \sigma^2 S_f^2)$. However, each user can add a fraction of the noise $\mathcal{N}(0, \sigma^2 S_f^2/n)$ and then use SMC to aggregate the individual values. This ensures that an optimal amount of noise added to the final sum to satisfy the promised $(\epsilon, \delta)$-DP guarantee. In the federated learning setting, [40, 42] have shown that the noise reduction technique coupled with SMC can be used to train machine learning models with a DP guarantee.

# 3 OUR SYSTEM MODEL

In the description, we mention machine learning (ML) in general to mean SGD based machine learning.

## 3.1 Our System Setting and Problem Statement

Our system model consists of three different entities, namely *a set of users* who are willing to contribute their private data to train an ML model, *a centralized server* who conducts the training process and *an aggregator* who assists in the training process. The notion of aggregator comes naturally in many applications such as 5G. The basestation in 5G may act like a gateway that collects and aggregates data from mobile devices and sends the aggregated data to a back-end, centralized server. We can think this type of gateway playing the role of an aggregator in our system model.

Figure 1 shows a high-level overview of our system model. We consider the problem of private distributed training of a machine learning model in the federated learning setting. A distributed training computation is described in Section 4. We follow the federated learning training computation where the server holds the ML model, and the users locally store their private dataset, but share the model that is trained on the local data. Our goal is to design a verifiable, secure and private training protocol with the following properties:

**Correctness:** When all the users and the server follow the prescribed steps of the protocol, it outputs the correct machine learning model.

**Privacy:** The protocol provides a user's data privacy and protects the its local model against the server, other users and any external attacker in the system. Privacy requirement is further categorized as follows.

*Privacy of Computation.* During the federated learning training process, the individual user model updates can leak information about the inputs. For example, in the NLP domain, non-zero gradients can leak information about the words used if a bag-of-words model is used as an embedding to train text-based classifier [42]. Secure Multiparty Computation (SMC) [8] can alleviate this problem by ensuring that only information about the final function is revealed without leaking any information about the inputs. In other words, *Privacy of Computation* prevents information leakage during the training process from the individual $\boldsymbol{\theta}^i$ in Equation (1) beyond the knowledge the aggregated model $\boldsymbol{\theta}$.

*Privacy of Output.* While SMC techniques ensure that the individual models $\boldsymbol{\theta}^i$ are not revealed during aggregation, the aggregated model $\boldsymbol{\theta}$ in Equation (1) can also reveal information about the training data $\mathcal{D}$ (e.g.[17, 39]). Differentially private training techniques, however, can help mitigate the inference over the output model and quantify the privacy loss.

*Privacy of Computation* complemented by *Privacy of Output* techniques offer strong privacy guarantees during and after the federated learning training process.

**Robustness and verifiability:** The protocol is robust so that dynamic users do not affect the training or adds additional computational overhead to other users in the system. The verifiability of the global model is achieved, meaning if the aggregator computation deviates from the actual computation, then it should be detected by the server.

**Efficiency:** The communication and computation costs are minimal. The communication complexity can be divided into two parts. First is the round complexity, which is the number of rounds a user has to send data. Second is the bandwidth complexity, which means how much data is being sent though the network. Lesser the data lower the bandwidth requirement.

## 3.2 Threat Model

We consider the semi-honest adversaries in our system in which an adversary follows the prescribed instructions of the protocol and may try to learn any unintended information from the message exchanges in the protocol. An adversary may compromise some participants in the system. We consider three different adversarial scenarios: 1) an adversary corrupting a set of users and the server; 2) an adversary corrupting only a set of users; and 3) an adversary corrupting a set of users and the aggregator.

The goal of the adversary is to learn information about honest users' private data (e.g., by applying adversarial ML techniques on the locally trained model). We assume that there is no server and aggregator collusion. This assumption is in line with the notion of *collector* [22] that plays a similar role as the aggregator does in our system model. However, it is allowed to collude with some users, that is the aggregator is not fully trusted. Further, we assume the adversarial behaviour of the aggregator as per the *cheap and lazy* adversary as described in [10]. It is cheap as it may try to reduce resource costs which may make the computations error prone. It is lazy as it may not actively modify the users' input. This notion captures a typical cost cutting third party service provider. Thus, the adversary's behaviour is different from an active adversary and it is modeled as a semi-honest adversary. Our threat model is similar to the one considered in [42]. However, we do not assume fully trusted third party. It is also closely related to [9], in the semi-honest setting. Other works are diverse in terms of threat model considered. We discuss their threat models in Section 6.

# 4 OUR SOLUTION

In this section, we first describe our training protocol $\Pi_{\text{Global\_Train}}$ (Figure 2) for a single aggregator scenario, and then a generalized training protocol ($\Pi_{\text{MAFL\_Train}}$) for the multi-aggregator scenario. The crux of the solution is that each user locally updates her model and homomorphically encrypts it (with server's public key) and creates a homomorphic authenticator for the input. These two are sent to an aggregator that homomorphically adds local models and creates a combined authenticator. These two are sent to the server, which can decrypt ciphertext to get the sum of the local models, and can also verify the correctness with the help of the combined authenticator. So this solution offers one round communication complexity from users to the server and verifiability property.

## 4.1 Description of $\Pi_{\text{Global\_Train}}$

In the training process, the server first initializes the model $\boldsymbol{\theta}$ and then distributes the encrypted model to the users ($\mathcal{U}$) where the model encryption is done using the AE scheme with users-server shared symmetric-keys. Then each user $U_i \in \mathcal{U}$ decrypts the encrypted model using the shared key ($SK_i$), and then runs Algorithm 1 with her local data and this initial model to get a locally updated model $\theta^i \leftarrow \texttt{Train}(\boldsymbol{\theta}, \mathcal{D}_i, b, S_f, \sigma, \eta, t)$. This training is enabled with differential privacy for strong privacy guarantees. Then, the user $U_i$ encrypts $\theta^i$ as $C_{\theta^i} \leftarrow \text{Enc}_{\text{pk}_{JL}}(\theta^i)$ by JL AHE scheme with the server's public key $\text{pk}_{JL}$, followed by signing the local model $\theta^i$ using the MLHA signature scheme, $\sigma_i \leftarrow \text{MLHA.Auth}(\text{sk}_{\text{sig}_i}, \Delta, l, \theta^i)$. The user $U_i$ encrypts $C_{\theta^i}$ using the AE scheme with the user-aggregator shared key and sends the encrypted $C_{\theta^i}$ along with $\sigma_i$. After receiving inputs from $U_i$, the aggregator first decrypts encrypted $\{C_{\theta^i}\}$ and evaluates an AHE encrypted sum of local models that have been received from the set of alive user $\mathcal{L}$ in the form $\text{Enc}_{\text{pk}_{JL}}(\sum_{U_i \in \mathcal{L}} \theta^i)$. It also computes an aggregated signature $\sigma$ on $\{\sigma_i\}$ using the MLHA.Eval function. The aggregator sends $\text{Enc}_{\text{pk}_{JL}}(\sum_{U_i \in \mathcal{L}} \theta^i)$ and aggregated signature $\sigma$ to the server. After receiving $\text{Enc}_{\text{pk}_{JL}}(\sum_{U_i \in \mathcal{L}} \theta^i)$ and $\sigma$, the server decrypts the aggregated encrypted model to obtain $\sum_{U_i \in \mathcal{L}} \theta^i$ and subsequently verifies the validity of the sum with the aggregated signature $\sigma$. Steps 1 to 7 in Figure 2 complete 1 epoch of training.

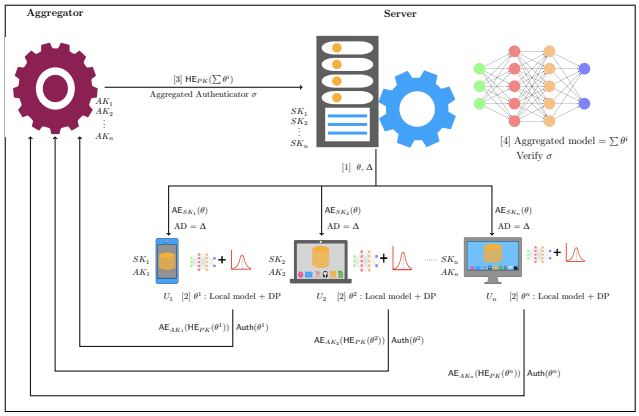



**Figure 1: Description of $\Pi_{\text{Global\_Train}}$**

*Local Training using Differential Privacy.* In machine learning, the differential privacy is applied to two fundamental tasks namely *private prediction* and *private training*. The private prediction tasks only consider the privacy of the outputs of an ML model after training i.e., information leakage of the training data is limited in the prediction phase once model training is complete. This is useful in scenarios where the model weights are not made public and access to the model is facilitated in the form of an API. On the other hand, the private training tasks take into consideration the privacy of the training data, during and after the training process is complete. In the federated learning setting in which the server and users have direct access to the updated global model during the training process, the private training task limits the information

**Our Privacy-preserving Training Protocol** ($\Pi_{\text{GLOBAL\_TRAIN}}$)

**Server:** The server holds the machine learning model $\boldsymbol{\theta}$

**Users:** Each user $U_i \in \mathcal{U}$, where $\mathcal{U}$ is the set of all users, has a private dataset $\mathcal{D}_i$ and shares long term keys $K_{SU_i}$ and $K_{AU_i}$ with the server and the aggregator respectively

▷ **Key setup phase**

**Server:**

- Generates a session ID $\kappa$ and sends this to all the users and the aggregator.
- Generates a session key for each user, $SK_i = KDF(K_{SU_i}, \kappa)$, where $KDF$ is a key derivative function.
- Generates the private and public key pair of the JL HE scheme, $(\mathsf{sk}_{JL}, \mathsf{pk}_{JL}) \leftarrow \mathsf{JL.KeyGen}(1^\lambda)$
- Generates the public parameters $\mathsf{pp} \leftarrow \mathsf{MLHA.Setup}(1^\lambda)$ for the signature scheme and sends it to the users.

**Users:**

- Each user $U_i \in \mathcal{U}$ derives the session keys $SK_i = KDF(K_{SU_i}, \kappa)$ corresponding to the Server and $AK_i = KDF(K_{AU_i}, \kappa)$ corresponding to the aggregator.
- Each user $U_i$ runs $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{MLHA.Keygen}(\mathsf{pp})$ to generate signing and verification keys of the aggregated signature scheme, and sends $\mathsf{vk}_i$ to the server.

**Aggregator:**

- Aggregator derives the session key $AK_i = KDF(K_{AU_i}, \kappa)$ corresponding to each user $U_i$.

▷ **Server initializes the model**

1. Server initializes the model $\boldsymbol{\theta}$ and sets dataset identifier $\Delta$. For each user $U_i$, server computes $\boldsymbol{\theta}_{U_i} \leftarrow \mathsf{AE.Enc}_{SK_i}(\boldsymbol{\theta}, \Delta)$, where $\Delta$ is taken as the associated data (AD). Then sends $\boldsymbol{\theta}_{U_i}$ and $\Delta$ to user $U_i$,.

▷ **Users locally train the model**

2 Each user $U_i \in \mathcal{U}$:

2.1. Applies decryption to get $\boldsymbol{\theta} \leftarrow \mathsf{AE.Dec}_{SK_i}(\boldsymbol{\theta})$.

2.2. Applies Algorithm 1 and gets locally updated model $\theta^i \leftarrow \mathtt{Train}(\boldsymbol{\theta}, \mathcal{D}_i, b, S_f, \sigma, \eta, t)$.

2.3. Encrypts $\theta^i$ by JL homomorphic encryption, $C_{\theta^i} \leftarrow \mathsf{Enc}_{\mathsf{pk}_{JL}}(\theta^i)$.

2.4. Signs $\theta^i$ using MLHA signature scheme, $\sigma_i \leftarrow \mathsf{MLHA.Auth}(\mathsf{sk}_i, \Delta, l, \theta^i)$.

2.5. Further encrypts $C^A_{\theta^i} \leftarrow \mathsf{AE.Enc}_{AK_i}(C_{\theta^i})$, where $\Delta$ is used as the associated data, and sends $C^A_{\theta^i}$ and $\sigma_i$ to the aggregator.

▷ **Aggregator aggregates the model**

3. For all users in $\mathcal{L}$, where $\mathcal{L}$ is the set of alive users

$$C_{\boldsymbol{\theta}^i} \leftarrow \mathsf{AE.Dec}_{AK_i}(C^A_{\theta^i})$$

4. Aggregator aggregates users local models

$$\mathsf{Enc}_{\mathsf{pk}_{JL}}(\boldsymbol{\theta}) = \mathsf{Enc}_{\mathsf{pk}_{JL}}(\sum_{U_i \in \mathcal{L}} \theta^i) \leftarrow \mathsf{JL.Eval}(\mathsf{pk}_{JL}, \{C_{\theta^i}\}_{U_i \in \mathcal{L}})$$

and compute the aggregated signature as $\sigma \leftarrow \mathsf{MLHA.Eval}(f, \{\sigma_i\}_{U_i \in \mathcal{L}})$. It sends the aggregated model $\mathsf{Enc}_{\mathsf{pk}_{JL}}(\boldsymbol{\theta})$ and aggregated signature $\sigma$ to the server.

▷ **Server computes the updated global model**

5. Server decrypts $\mathsf{Enc}_{\mathsf{pk}_{JL}}(\boldsymbol{\theta})$ and obtains the model $\boldsymbol{\theta} \leftarrow \mathsf{Dec}_{\mathsf{sk}_{JL}}(\mathsf{Enc}_{\mathsf{pk}_{JL}}(\sum_{i=1}^{n} \theta^i))$.
6. Then runs the $\mathsf{MLHA.Ver}(\mathcal{P}_\Delta, \{\mathsf{vk}_i\}_{U_i \in \mathcal{L}}, \boldsymbol{\theta}, \sigma)$, if it verifies, then accepts $\boldsymbol{\theta}$.
7. Server updates global model $\boldsymbol{\theta} = \dfrac{\boldsymbol{\theta}}{|\mathcal{L}|}$.
8. Server repeats steps 1 to 7 with the updated global model $\boldsymbol{\theta}$ and continues until the model converges when both server and users possess the final model.

**Figure 2: Our Privacy-preserving Training Protocol ($\Pi_{\text{GLOBAL\_TRAIN}}$)**

leakage of the users' data. PROV-FL leverages the noise reduced variant of the DP-SGD [1] algorithm in [40] for federated learning (Algorithm 1). $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}, d_j)$ denotes the gradient that is obtained after the backpropagation. Aggregating at least $t$[1] locally trained models using Algorithm 1 is $(\epsilon, \delta)$-DP with respect to each sampled batch if $\sigma \geq \sqrt{2\log\frac{1.25}{\delta}}/\epsilon$ [16]. Using the Moments Accountant [1], $k$ epochs of training provides $\left(O\left(b\epsilon\sqrt{k/b}\right), \delta\right)$-DP. One epoch of training is a single call to the `Train` function in Algorithm 1.

*Need for Secure Aggregation.* One may think that why it is necessary to use the secure aggregation technique to aggregate locally updated models, given that some noises have already been added to the local models by Algorithm 1. The DP-SGD algorithm is a local DP algorithm in which each user trains a local ML model on its dataset with a sufficient amount of noise to provide the target an $(\epsilon, \delta)$-DP guarantee. In other words, the DP-SGD prevents any inference on $\boldsymbol{\theta}^i$. Algorithm 1, however, adds a smaller amount of noise to the individual models $\boldsymbol{\theta}^i$ and only prevents any inference over the aggregated model $\boldsymbol{\theta}$. Therefore, a secure aggregation technique is applied to protect individual $\boldsymbol{\theta}^i$. Aggregating models with the local DP guarantee with a large amount of noise results in a significant loss to the model utility. Thus, combining the secure aggregation with a reduced noise in DP-SGD helps preserve the model utility. A comparison of these approaches is further evaluated in Section 5.

## 4.2 Generalized Protocol: $\Pi_{\text{MAFL\_TRAIN}}$

We now describe $\Pi_{\text{MAFL\_TRAIN}}$ that involves multiple aggregators, which is a generalization of the single aggregator-aided training

[1]We note that [40] scale down the amount of noise by a factor of $t - 1$ to ensure that the total noise after aggregating individual models is strictly greater than the amount of noise required. However, similar to [42], we scale down by $t$ which ensures that the noise is greater than or equal to the required amount after aggregation.

**Algorithm 1:** Noise-Reduced Differentially Private Stochastic Gradient Descent (DP-SGD) [40]

**Input:** Model $\boldsymbol{\theta}$, Dataset $\mathcal{D}$, batch rate $b$, sensitivity $S_f$, noise scale $\sigma$, learning rate $\eta$, threshold of alive and honest users $t$

**Output:** Trained model $\boldsymbol{\theta}$

**function** $\texttt{Train}(\boldsymbol{\theta}, \mathcal{D}, b, S_f, \sigma, \eta, t)$:
  **for** *i = 1 to 1/b* **do**
    Randomly sample $d$ from $\mathcal{D}$ with probability $b$
    **for** $d_j \in d$ **do**
      $g_i(d_j) = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, d_j)$
      $g_i(d_j) = g_i(d_j)/max\left(1.0, \frac{\|g_i(d_j)\|_2}{S_f}\right)$
    $g_i = \frac{1}{|d|}\left(\sum_{\forall j} g_i(d_j) + \mathcal{N}\left(0, \frac{\sigma^2}{t} \cdot S_f^2\right)\right)$
    $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \cdot g_i$
  **return** $\boldsymbol{\theta}$

protocol that captures real world scenarios. In the multi-aggregator training scenario, consider a mobile network with a group of mobile phones that are connected to a base station. The base stations are connected to a server. Let us call the group of mobile phones and connected base station as a cluster. In reality there are multiple base stations and hence multiple clusters. These base stations form the backbone of a mobile network and it can be assumed that they are always online for a smooth functioning of the network.

Federated learning in this setup is as follows. Suppose $B_1, \ldots, B_k$ are the base stations and in the cluster of $B_i$ and mobile phones are denoted as users $U_{ij}$. Each base station plays the role of an aggregator. Henceforth, we mention base stations as aggregators. After receiving server model $\boldsymbol{\theta}$, each user $U_{ij}$ runs the model on her data and updates the model locally to get $\theta^{ij}$. Then $U_{ij}$ sends encrypted $\theta^{ij}$ as $\mathsf{Enc}_{\mathsf{pk}_{JL}}(\theta^{ij})$ and the corresponding authenticator $\sigma_{ij}$ to its aggregator $B_i$. Aggregators collect encrypted local models and authenticators from their connected users, and aggregate them and sends to the server. Aggregator $B_i$ aggregates the encrypted models as $\mathsf{Enc}_{\mathsf{pk}_{JL}}(\theta^i) = \mathsf{Enc}_{\mathsf{pk}_{JL}}(\sum_{U_{ij} \in \mathcal{L}_i} \theta^{ij})$, where $\mathcal{L}_i$ is the set of alive users in the cluster of $B_i$. Then $B_i$ aggregates the authenticators as $\sigma_i$ using MLHA.EVAL function. Up to this point, the protocol goes exactly the same way as $\Pi_{\text{Global\_Train}}$ for each cluster, and then it differs in the next step. In the next step, all the aggregators run a secure aggregation protocol such as [9] in the same way as used in [26] (here users role is played by aggregators). Let $\mathsf{Enc}_{\mathsf{pk}_{JL}}(\theta^1 + r_1), \ldots, \mathsf{Enc}_{\mathsf{pk}_{JL}}(\theta^k + r_k)$ be the respective shares of $B_1, \ldots, B_k$ which are sent to the server. Note that $\sum_{i=1}^k r_i = 0$ as per the secure aggregation protocol. Aggregator $B_i$ also sends $\sigma_i$ separately to the server. At the last step, server computes the aggregated model as

$$\boldsymbol{\theta} = \mathsf{Dec}_{\mathsf{sk}_{JL}}\left(\Pi_{i=1}^k \mathsf{Enc}_{\mathsf{pk}_{JL}}(\theta^i + r_i)\right) = \sum_{i=1}^k (\theta^i + r_i) = \sum_{i=1}^k \theta^i.$$

Server also aggregates $\sigma_1, \ldots, \sigma_k$ to get $\sigma$ and then verifies the authenticity of $\boldsymbol{\theta}$. Server initiates next iteration of the training with $\boldsymbol{\theta} = \frac{\boldsymbol{\theta}}{|\mathcal{L}|}$, where $\mathcal{L}$ is the set of alive users. We note that the evaluation of another linear function using the authenticators of individual functions is supported by MLHA scheme. In other words, the server can evaluate a linear function $g(f_1, \ldots, f_k) = c_1 \cdot f_1 + \ldots + c_k \cdot f_k$, where $f_i$ is the function evaluated by the $i^{th}$ base station.

This protocol involving multiple aggregators is still 1-round as the server does not need to run several iterations with the aggregators to unmask the effect of $r_i$ from the aggregation of $\{\mathsf{Enc}_{\mathsf{pk}_{JL}}(\theta^1 + r_1), \ldots, \mathsf{Enc}_{\mathsf{pk}_{JL}}(\theta^k + r_k)\}$ as it was required in [9] or [26]. Due to the fact these base stations are always online, so they all are able to send their shares to the server without fail.

## 4.3 Supporting Dynamic Joins and Dropouts

The MLHA scheme supports the evaluation of a linear function $f : \mathcal{M}^n \to \mathcal{M}$. When used in our training protocol $\Pi_{\text{Global\_Train}}$, we set $n$ to be the maximum number of users that can participate in the training procedure. Assume that the number of users who are willing to participate in the beginning is $k$. Now, if $k = n$, then we cannot allow new users to join without performing the global setup procedure again. When $k < n$, up to $n-k$ additional users can join without disrupting the training. The new users only need to perform the local setup procedure to generate their secret and public keys. Consider, for example, the number of users intially is $k = 7$ and the total number of users that can participate is $n = 10$. Each user $u_i$, where $i \in [1, k]$, holds a message $x_i$. The function $f$ and the inputs of the function to which each user contributes is public information in the MLHA scheme and does not leak information about the user's input. For simplicity, assume the user $u_i$ contributes to the $i^{th}$ input of the function. Then, $f(\mathbf{x}) = \mathbf{c} \cdot \mathbf{x} = \sum_{i=1}^n c_i x_i$. Here, $\mathbf{x} = [x_1, \ldots, x_n]$ is a vector of the users' inputs and $\mathbf{c} = [c_1, \ldots, c_n]$ are the coefficients of $f$. In case of simple aggregation all elements of $\mathbf{c}$ are 1 i.e $c_{i \in [1,n]} = 1$. While the users are sending their local updates, we have the following possible scenarios for $k = 7$ and $n = 10$:

(1) **No users dropout or join**: In this scenario, we have $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, 0, 0, 0]$ and $\mathbf{c} = [1, 1, 1, 1, 1, 1, 1, 0, 0, 0]$. Then, $f(\mathbf{x}) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7$.
(2) **Some users dropout**: Assume that users 2 and 5 dropout and their respective inputs are not sent to the aggregator. Then $\mathbf{x} = [x_1, 0, x_3, x_4, 0, x_6, x_7, 0, 0, 0]$ and $\mathbf{c} = [1, 0, 1, 1, 0, 1, 1, 0, 0, 0]$. In that case, $f(\mathbf{x}) = x_1 + x_3 + x_4 + x_6 + x_7$.
(3) **Some users join**: Assume that 2 new users join in addition to the existing 7 users and they are $u_8$ and $u_9$. In this case, $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, 0]$ and $\mathbf{c} = [1, 1, 1, 1, 0, 1, 1, 1, 1, 0]$. Then, $f(\mathbf{x}) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9$.
(4) **Some users dropout and join**: Assume that 2 new users join in addition to the existing 7 users and they are $u_8$ and $u_9$. While sending the inputs, users $u_3$ and $u_4$ dropout. So, $\mathbf{x} = [x_1, x_2, 0, 0, x_5, x_6, x_7, x_8, x_9, 0]$ and $\mathbf{c} = [1, 1, 0, 0, 1, 1, 1, 1, 1, 0]$. Then, $f(\mathbf{x}) = x_1 + x_2 + x_5 + x_6 + x_7 + x_8 + x_9$.

## 4.4 Security Analysis of Our Protocol

We use the simulation based proof technique (Real/Ideal simulation paradigm) to prove the security of our training protocol $\Pi_{\text{Global\_Train}}$. We consider three trust settings: 1. an adversary

corrupting a set of users and the server (Theorem 4.1), 2. an adversary corrupting a set of users (Theorem 4.2) and 3. an adversary corrupting a set of users and the aggregator (Theorem 4.3). We do not claim the input privacy of the honest users when an adversary corrupts both the server and the aggregator simultaneously. The ideal functionality as per the the description of Federated Learning in Section 2.1 is given in Appendix A.1.

Theorem 4.1 (Adversary corrupting Users-server). *Suppose the additive homomorphic encryption scheme $E()$ is semantically secure and the authenticated encryption* AE *is IND-CPA secure. The protocol $\Pi_{Global\_Train}$ is secure against the semi-honest adversaries, meaning it does not leak any information about private datasets of the honest users.*

The proof is given in Appendix A.2.

Theorem 4.2 (Adversary corrupting Users). *Suppose the additive homomorphic encryption scheme $E()$ is semantically secure and the authenticated encryption* AE *is IND-CPA secure. The protocol $\Pi_{Global\_Train}$ is secure against the semi-honest adversaries, meaning it does not leak any information about private datasets of the honest users.*

We now prove that the protocol preserves the input privacy of the honest users when a semi-honest adversary corrupts a set of users and the aggregator, meaning the adversary does not learn any information about the honest users' input privacy.

Theorem 4.3 (Adversary corrupting Users-aggregator). *Suppose the additive homomorphic encryption scheme $E()$ is semantically secure and the authenticated encryption* AE *is IND-CPA secure. Then $\Pi_{Global\_Train}$ is secure against the semi-honest adversaries, meaning it does not leak any information about private datasets of the honest users.*

The proof is given in Appendix A.2. In the following theorem, we provide the security property of $\Pi_{\text{MAFL\_Train}}$.

Theorem 4.4. *Consider the training protocol in the multi-aggregator scenario. Suppose the additive homomorphic encryption scheme $E()$ is semantically secure and the authenticated encryption* AE *is IND-CPA secure. The multi-aggregator training protocol $\Pi_{MAFL\_Train}$ is secure against the semi-honest adversaries who can corrupt users-and-server, users or users-and-aggregators, meaning the protocol does not leak any information about private datasets of honest users.*

## 4.5 Choice of Parameters

Suppose, $N$ is the total number of users who are running the PROV-FL protocol. In reality, some users will drop out due to network or some other issues. Let $\mu$ be the fraction of dropping out users on the average. Further suppose that $\psi$ is the fraction of colluding users. Therefore, in the worst case, the number of honest users is $\mathcal{H} = (1-\mu-\psi)N$. In order to resist active adversarial coalition, we set $(1-\mu-\psi) > \frac{2}{3}$. We also have the parameter $t$ which is the threshold for the number of alive and honest users in Algorithm 1 that decides the amount by which the noise needs to be scaled down to achieve $(\epsilon, \delta)$ privacy in the aggregated model. Then $(1-\mu-\psi) \geq \frac{t}{N}$, thus $(1-\mu-\psi) \geq \max(\frac{t}{N}, \frac{2}{3})$. We also recommend to set $\epsilon \leq 1$ for DP to ensure strong privacy guarantees. We would like to note that $\delta \leq \frac{1}{d}$ is generally considered where $d$ is the total number of training samples [1].

# 5 EXPERIMENTAL EVALUATION

## 5.1 Implementation Details

We implement PROV-FL in Python 3.7. We use `gmpy2==2.1.3`[2] for multi precision integers and floats, Charm[4] with the RELIC[3] backend for elliptic curve arithmetic over a 256-bit Barreto-Naehrig (BN) curve, ECDSA with the NIST P-256 curve and SHA-256 hash function from `fastecdsa==2.2.3`[4] to realise a signature scheme, AES-128 CBC and AES-128 GCM from `cryptography==36.0.1`[5] to realise a pseudorandom function and a secure channel, and `joblib==1.1.0`[6] for multiprocessing. The ML models are implemented using the Keras API in `tensorflow-privacy==0.7.3` and `tensorflow==2.8.0`. All experiments were conducted on a consumer grade laptop with a Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz processor, 16 GB RAM, and Ubuntu MATE 18.04.5 LTS.

## 5.2 Dealing with Floating Point Numbers in Cryptographic Algorithms

The machine learning model $\boldsymbol{\theta}$ is represented using 32-bit floating point numbers which can be positive or negative. However, the additive homomorphic encryption and signature schemes operate over the integers and do not natively support floating point numbers. To resolve this issue, we represent the floating point numbers as integers in $\mathbb{Z}_p$, where the 256-bit integer $p$ is the prime order of the cyclic groups used in the MLHA scheme. Similar to [26], the floating point numbers are encoded to integers before performing the cryptographic operations and decoded after they are complete.
**Encode:** The message space $\mathbb{Z}_p$ is divided into two halves. Positive floating point numbers are encoded in $[0, \frac{p}{2} - 1]$ and negative numbers are encoded in $[\frac{p}{2}, p-1]$. Given an integer $\tau \in [1, 127]$, a positive floating point number $x$ is encoded as $\hat{x} = \mathsf{Encode}(x, \tau) = \mathtt{round}(x \cdot 2^{\tau})$. If $x$ is negative, then it is encoded as $\hat{x} = p - \mathsf{Encode}(|x|, \tau)$. The `round` function rounds off the operand to the nearest integer. The integer $\tau$ is chosen accordingly to ensure that no overflow error occurs in the message space. In our experiments, we set $\tau = 90$.
**Decode:** Given the encoding of $x$ as $\hat{x} \in \mathbb{Z}_p$, the decoding is computed as $x = \mathsf{Decode}(\hat{x}, \tau) = -\frac{p - \hat{x}}{2^{\tau}}$ if $\hat{x} \geq \frac{p}{2}$, else $\frac{\hat{x}}{2^{\tau}}$.

## 5.3 Datasets, Models, and Experimental Settings

We evaluate the performance of PROV-FL using the following settings:

- **No Privacy**: Federated learning without any privacy guarantees to serve as a benchmark against which we can compare the performance of PROV-FL.
- **PROV-FL**: Our privacy-preserving federated learning protocol with privacy of computation and privacy of output guarantees

[2] https://gmpy2.readthedocs.io/en/latest/
[3] https://github.com/relic-toolkit/relic
[4] https://fastecdsa.readthedocs.io/en/v2.2.3/
[5] https://cryptography.io/en/36.0.1/
[6] https://joblib.readthedocs.io/en/latest/

**Table 1: Description of Datasets and Machine Learning Models**

| Experiment ID | Dataset | Dataset Type | Model Type | No: of model parameters | Users | Minimum Alive Users per Epoch |
|---|---|---|---|---|---|---|
| 1 | MNIST[21] | Grayscale Image | CNN | 26,010 | 25 | 18 (≥ 70%) |
| 2 | Fashion-MNIST[41] | Grayscale Image | DNN | 101,770 | 30 | 24 (≥ 80%) |
| 3 | IMDB Movie Reviews[25] | Text | DNN | 80,305 | 10 | 7 (≥ 70%) |
| 4 | Rice Image Dataset[20] | RGB Image | CNN | 79,445 | 25 | 18 (≥ 70%) |

using Algorithm 1 and secure aggregation through additive homomorphic encryption.

- **PROV-FL (No DP)**: PROV-FL without privacy of output i.e., no differential privacy. Users train their models using vanilla SGD and their models are aggregated using secure aggregation.
- **Local DP**: Local training of individual user models using their private datasets trained using DP-SGD and aggregated without secure aggregation. In this case, each user adds enough noise to satisfy the DP guarantee of their own data in isolation.

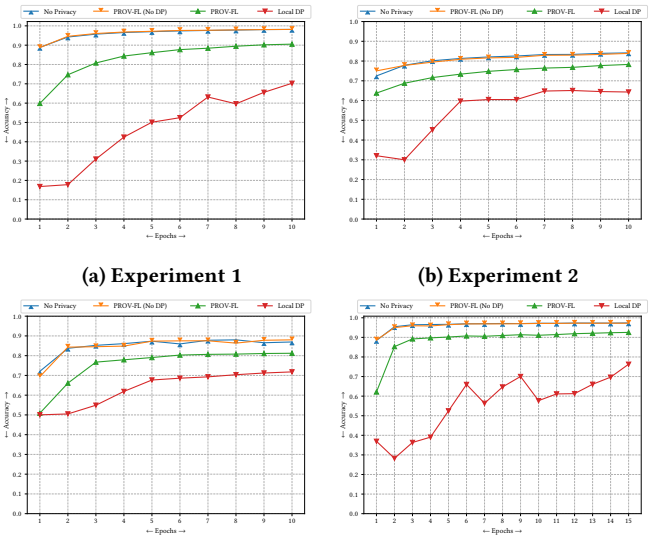
A summary of the different datasets and ML models used in our experiments are given in Table 1. In Experiment 1, we train a CNN on the MNIST dataset. The dataset contains 28×28 grayscale images of handwritten digits from 0 - 9. The ML problem is a 10-class classification task. The train and test datasets contain 60, 000 and 10, 000 images respectively. We use a CNN model with 2 convolution layers and 2 linear layers.[7]. ReLU is used as the activation function and Maxpool as the pooling layer after the convolutions. The train dataset is split equally among all the users, with each user having 2, 400 samples.

In Experiment 2, we train a DNN on the Fashion-MNIST dataset. The dataset contains 28 × 28 grayscale images of clothing items. Similar to MNIST, the ML problem is a 10-class classification task. The train and test datasets contain 60, 000 and 10, 000 images respectively. We use a DNN model with 2 linear layers and ReLU as the activation function[8]. The train dataset is split equally among all the users, with each user having 2, 000 samples.

In Experiment 3, we train a DNN on the IMDB Movie Reviews dataset that contains 50, 000 movie reviews in text format. The ML problem is sentiment analysis task formulated as a binary classification problem where we need to predict whether the movie review is positive or negative. The train and test datasets contain 25, 000 samples each. During the pre-processing step, the 5, 000 most frequent words are kept. The text sequences are then represented as binary vectors of length 5, 000 where each co-ordinate of the vector is 1 if the corresponding word is found in the sequence. The train dataset is split equally among all the users, with each user having 2, 500 samples. We use a DNN model with 3 linear layers and ReLU as the activation function[9].

In Experiment 4, we train a CNN on the Rice Image Dataset. The dataset contains 250 × 250 RGB images of rice grains. The ML problem is a 5-class classification task. We split the dataset containing 75, 000 images into 62, 500 and 12, 500 train and test images respectively. We pre-process the images by resizing them to 50 × 50 RGB. We create a CNN model with 2 convolution layers

[7]https://github.com/tensorflow/privacy/blob/master/tutorials/mnist_dpsgd_tutorial_keras.py
[8]https://www.tensorflow.org/tutorials/keras/classification
[9]https://www.kaggle.com/code/drscarlat/imdb-sentiment-analysis-keras-and-tensorflow/notebook




(a) Experiment 1 (b) Experiment 2

(c) Experiment 3 (d) Experiment 4

**Figure 3: Accuracies in different settings and experiments**

followed by 2 linear layers. The first and second convolution layers have 32 and 16 filters each. Maxpool is used as the pooling layer after the convolutions. The kernel size for the convolution and maxpool layers are $(3, 3)$ and $(2, 2)$ respectively. The size of the following linear layers are 32 and 5. The activation function is ReLU for all the layers except the last linear layer. The train dataset is split equally among all the users, with each user have 2, 500 samples.

For all experiments, we set the batch rate $b = 0.01$, L2-norm clipping (sensitivity) $S_f = 1.5$, and use $(0.5, 1 \times 10^{-5})$-DP. The learning rate in Experiment 2 is set to 0.15 and in the other experiments we use a learning rate of 0.1. The pixel values in all the image datasets are scaled down to lie in $[0, 1]$. The DP guarantee is estimated using tensorflow-privacy's `compute_dp_sgd_privacy` utility. During our experiments, we do not perform hyperparameter optimization to find the most optimal hyperparameters. Our intention is to analyse and determine the utility of the model in the No Privacy, PROV-FL, PROV-FL (No DP), and Local DP settings keeping the same model architecture and hyperparameters fixed. While we demonstrate PROV-FL on DNN and CNN models, we note that it supports other ML models such as SVMs, linear/logistic regression, decision trees, and other variants of neural networks. PROV-FL is compatible with all the noise reduced DP ML algorithms proposed in [40].

## 5.4 Experimental Results

*5.4.1 Model Performance in different settings.* Figure 3 shows the accuracy on the test dataset after every epoch of training for the different settings and experiments. From the four graphs, we make the following observations. First, the accuracy in the No Privacy and PROV-FL (No DP) settings is the same. Second, there is a minor loss in accuracy in the PROV-FL setting. Third, the Local DP setting is significantly worse in terms of accuracy and the accuracy curves are not smooth. The first observation can be attributed to the fact that the error, if any, that is introduced while encoding

**Table 2: Data sent/received by different parties per epoch**

| | Data Transfer per Epoch in Megabits (Mb) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | No Privacy/Local DP | | | PROV-FL/PROV-FL (No DP) | | | | | |
| | User | | Server | User | | Aggregator | | Server | |
| Experiment ID | *Sent/Received* | *Sent* | *Received* | *Sent* | *Received* | *Sent* | *Received* | *Sent* | *Received* |
| 1 | 0.10 | 0.10 | 2.6 | 9.99 | 0.10 | 10.0 | 249.71 | 0.10 | 10.0 |
| 2 | 0.41 | 0.41 | 12.21 | 39.08 | 0.41 | 39.09 | 1172.40 | 0.41 | 39.09 |
| 3 | 0.32 | 0.32 | 3.21 | 30.84 | 0.32 | 30.84 | 308.38 | 0.32 | 30.84 |
| 4 | 0.32 | 0.32 | 7.94 | 30.51 | 0.32 | 30.51 | 762.68 | 0.32 | 30.51 |

**Table 3: Average time taken per epoch by different entities**

| | Average Time per Epoch (secs) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | No Privacy | | PROV-FL (No DP) | | | PROV-FL | | | Local DP | |
| Experiment ID | User | Server | User | Aggregator | Server | User | Aggregator | Server | User | Server |
| 1 | 0.36 | 0.02 | 23.07 | 15.17 | 119.87 | 24.83 | 17.11 | 120.04 | 2.24 | 0.03 |
| 2 | 0.13 | 0.03 | 90.04 | 82.72 | 477.03 | 90.2 | 82.4 | 476.38 | 0.76 | 0.03 |
| 3 | 0.18 | 0.06 | 71.28 | 20.93 | 373.17 | 71.61 | 21.95 | 375.02 | 0.87 | 0.06 |
| 4 | 1.46 | 0.06 | 72.02 | 52.14 | 369.27 | 98.47 | 49.55 | 370.85 | 29.34 | 0.07 |

and decoding floats as integers is neglible. This does not have any discernible impact on the accuracy. The minor loss in accuracy in the PROV-FL setting is because of the Algorithm 1, which adds noise to every batch of the computed gradients. However, the loss in accuracy is not as significant as the Local DP setting as the amount of noise added in PROV-FL is sufficient to only protect the aggregated model and not the individual user models. This highlights the advantage of adding less noise and using secure aggregation to combine the individual models. The Local DP setting is the worst in terms of performance as an overwhelming amount of noise is added to each user model. The uneven and erratic nature of the accuracy curve can be attributed to the large amount of noise in the aggregated model. Our results show that PROV-FL can effectively preserve model utility while providing strong security and privacy guarantees. We perform experiments with a strong privacy guarantee of $(0.5, 1\times10^{-5})$-DP. For weaker guarantees, the performance of PROV-FL will be closer to the No Privacy setting. In every case, the Local DP setting will perform worse than PROV-FL due to extra noise that is accumulated in the aggregated model.

*5.4.2 Communication Cost.* Table 2 highlights the communication overhead incurred in different settings. The communication overhead for the aggregator and server depends on the number of users alive. As we randomly dropout some users each epoch in our experiments, the results reported in Table 2 correspond the worst case scenario in terms data transfer i.e, all users are alive. Local DP/No Privacy settings directly send the unencrypted weights of the model which are 32-bit floating point numbers. The DP-SGD training procedure does not result in any additional data transfer. In the PROV-FL/PROV-FL (No DP) settings, the JL encrypted model weights are 3072-bits each. The elliptic curve coordinates of elements of the MLHA scheme in $\mathbb{G}1$ and $\mathbb{G}2$ belong to $E(\mathbb{F}_p)$ and $E(\mathbb{F}_{p^2})$ respectively, where $p$ is a 256-bit prime and $E$ is a BN curve. The size of the ECDSA signature is 576 bits.

*5.4.3 Computation Cost.* Let $x$ be the number of parameters in the machine learning model, $N$ is the number of users, $y$ is the number of iterations in one epoch, and $d$ is the number of training samples in each iteration. First we derive the cost incurred to the user. As a user runs Algorithm 1, Homomorphic encryption and MLHA functionalities, the cost amounts to 1 AEAD encryption, 1 AEAD decryption, 1 PRF evaluation, 1 ECDSA signature evaluation, $2x + 6$ elliptic curve scalar multiplications, $2x + 2$ elliptic curve point additions, $x$ JL homomorphic encryptions, and $O(x * y * d)$ floating point operations. In case of aggregator, it only has to aggregate the authenticators and the ciphertexts. Thus the cost is as follows: $N$ AEAD decryptions, $2N$ elliptic curve point additions and $N \times x$ JL homomorphic evaluations. Finally we determine the cost at the server's end. Server has to run JL decryption and MLHA verification algorithm, so the cost is as follows: $N$ AEAD encryptions, $x$ JL decryptions, 1 ECDSA verification, $2N + 4$ pairing operations, $3N + 3$ scalar multiplications, $N + x$ point additions. Table 3 highlights the time incurred in different settings and experiments in Figure 3. We simulate all the users, aggregator, and server on a single machine and the reported times do not account for any overheads that are incurred during data transfer over a network. The time per epoch is computed as: Average time per user + Aggregator time + Server time, since each user trains their models in parallel. The execution time is computed uses Python's `time.perf_counter`. The Local DP and No Privacy experiments were conducted without setting up a secure channel. While the laptop we run our experiments on is representative of a user device, dedicated server hardware is more powerful and capable of efficiently handling parallel workloads. We believe that the server times can be significantly improved on better hardware.

*5.4.4 Comparing with HybridAlpha.* We note that HybridAlpha is the closest protocol to PROV-FL in terms of functionality. HybridAlpha uses Multi Input Functional Encryption (MIFE) [2] for secure aggregation. It provides privacy of computation and output and supports fully dynamic participation in a single round of communication but requires a fully trusted third party. To compare PROV-FL with HybridAlpha, we compute the time required by the users to encrypt 118, 110 gradients, which is the benchmark reported in their paper. We use CiFEr's[10] implementation of the MIFE scheme as HybridAlpha's source code is not public. The MIFE implementation takes 326 seconds per user for encryption. Similar to PROV-FL's implementation, we use multiprocessing to perform parallel encryption. PROV-FL takes 103 seconds per user ($\mathsf{Enc}_{\mathsf{pk}_{JL}}$ + MLHA.Auth time only). When we compare the reported results in HybridAlpha to perform the aggregation of 118, 110 gradients of 10 users, PROV-FL takes 737 seconds while HybridAlpha reports 34 seconds (user time + server time). The following are the reasons for time difference. First, we develop an end-to-end implementation for benchmarking which accounts for the overheads involved in the operations required to setup a secure channel. HybridAlpha only reports the execution times of the individual modules and assumes a secure channel is present. Second, the server to benchmark HybridAlpha is significantly more powerful than the laptop that we have used for these experiments.

The approach for designing PPML schemes is diverse, with varying privacy criteria (privacy of computation and/or output), experimental settings, and hardware. For example, Truex et. al. [40] use threshold homomorphic encryption and require 0.001095 and 0.007112 seconds for client and server cryptographic operations respectively. They use a CNN with 3 linear layers and 10 users for MNIST and do not report full protocol execution time. However, we use a CNN with 2 convolution layers and 2 linear layers with

[10] https://github.com/fentec-project/CiFEr

**Table 4: Comparison with Related Works**

| | Single Round | Verifiable Aggregation | Fully Trusted Third Party | Dynamic Participation |
|---|---|---|---|---|
| **PROV-FL** | Yes | Yes | No | Dropouts and Joins |
| **Bonawitz et. al.** [9] | No | No | – | Dropouts |
| **PRIV-FL** [26] | No | No | – | Dropouts |
| **HybridAlpha** [42] | Yes | No | Yes | Dropouts and Joins |
| **Truex et. al.** [40] | No | No | – | – |
| **PySyft** [34] | No | No | – | – |

25 users and provide detailed timing information. We provide an approach-related comparison with other works in Section 6 and highlight prominent works in Table 4.

# 6 RELATED WORKS

Table 4 summarizes the comparison between PROV-FL and related schemes. Secure aggregation protcols along with DP in federated learning settings can be used to provide privacy of computation and output. We categorize and discuss the remaining related works in our literature review into secure aggregation and privacy-preserving federated learning protocols.

## 6.1 Secure Aggregation

Bonawitz et al. [9] proposed a secure aggregation protocol that supports dropouts. However, their solution requires at least 3 rounds of communication. More recently, the authors extend their work [5] to support a large number of users with lower communication complexity. However, this solution too requires multiple rounds of communication and does not support dynamic joins.

Shi et al. [37] proposed a secure aggregation protocol with differential privacy that does not support dropouts. Additionally, the scheme is suitable for small plaintext spaces only which is not viable in the federated learning setting as floating point values used in ML models are encoded as large integers to ensure compatibility with cryptographic protocols.

Joye et al. [19] proposed a secure aggregation protocol which addresses the issue of small plaintext spaces in [37]. However, it does not support dynamic participation. Moreover, the ciphertexts are large since their protocol is built over Paillier cryptosystem. A follow up work [7] uses smaller ciphertexts while retaining the functionality of supporting large plaintext spaces but does not support dynamic participation.

Leontiadis et al. [22] proposed a single round secure aggregation protocol with dropout support using a "semi-trusted" third party. However, heir protocol is based on [19] and inherits the issue of large ciphertext spaces.

PUDA [23] introduced the notion of *aggregate unforgeability* and considers the case of an untrusted aggregator who has to prove to an external verifier that it has correctly aggregated the sum of the participating users. However, their scheme is built upon [37] and does not support dynamic participation.

Chan et al. [12] introduced a generic framework in which any secure aggregation protocol can be used as a building block to support dropouts. Their approach is based on fault tolerance via duplication, each participant needs to send their data multiple times ($O(\log n)$ where $n$ is the number of total users).

The solutions presented above either require multiple rounds of communication [5, 9], do not support fully dynamic participation [7, 12, 19, 22, 23, 37] or require higher communication per user [12]. PROV-FL overcomes all these shortcomings.

## 6.2 Privacy-preserving Federated Learning

Shokri et al. [38] proposed a variant of SGD algorithm for federated learning, where each participant chooses a fraction of the server's model parameters and updates their local models. Selected parameters with DP noise are uploaded to the server after local training. The authors did not propose a concrete implementation of how the participants exchange gradients and assumed all users are honest.

Chase et al. [13] proposed a collaborative neural network training framework using garbled circuits, DP and secret sharing. Participants send masked model updates and the masks to two trusted servers, who then use garbled circuits to compute the aggregated model. Their solution requires a single round of communication but is not resistant to dropouts and assumes all parties are honest.

PATE [32] presented an ensemble approach to private learning where several "teachers" train local models on private datasets. A fully-trusted aggregator provides an interface with DP guarantees to a "student" model with an unlabelled public dataset. The student can obtain labels by querying the teachers through the interface. However, this approach assumes all participants are honest.

PySyft [34] is a federated learning framework based on the SPDZ protocol [15]. PySyft offers privacy of computation and output guarantees but requires multiple rounds of communication and is not resilient to dropouts.

Truex at al. [40] proposed a federated learning framework which used threshold homomorphic encryption [14] to aggregate user models. Their approach considers privacy of computation and output but requires 3 rounds of communication and does not support dynamic participation.

PrivFL [26] is a framework that only supports linear and logistic regression algorithms. It only considers privacy of computation and requires multiple rounds of communication.

Existing solutions either require multiple rounds of communication [26, 34, 40], or require fully trusted entities throughout the execution of the protocol [32, 42] or assume that all participants are honest [13, 32, 38]. PROV-FL overcomes all these shortcomings.

Other PPML approaches are based on SMC [3, 11, 24, 28, 29], where users upload their encrypted data or secret shared data to one or more servers which then train ML models or use trusted execution environments [18, 30]. These approaches are different from PROV-FL in which the dataset does not leave the user system.

# 7 CONCLUSION AND FUTURE WORKS

We have presented PROV-FL, a privacy-preserving federated learning framework with verifiable aggregation and dynamic participation that needs only one round of communication. Our idea can also be tried on other machine learning techniques such as support vector machine in the distributed setup. Solution that does not rely on any third party and enjoys all the properties of PROV-FL will be an interesting research direction in future.

*Acknowledgment.* The authors would like to thank the anonymous reviewers of AISec '22 for their valuable feedback and comments to improve the quality of the paper.

# REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In

*Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 308–318.

[2] Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. 2018. Multi-Input Functional Encryption for Inner Products: Function-Hiding Realizations and Constructions Without Pairings. In *Advances in Cryptology – CRYPTO 2018*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer International Publishing, Cham, 597–627.

[3] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. 2019. QUOTIENT: Two-Party Secure Neural Network Training and Prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS'19)*. ACM, New York, NY, USA, 1231–1247.

[4] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. 2013. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptographic Engineering* 3, 2 (2013), 111–128.

[5] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1253–1269.

[6] Fabrice Benhamouda, Javier Herranz, Marc Joye, and Benoît Libert. 2017. Efficient Cryptosystems From $2^k$-th Power Residue Symbols. *J. Cryptol.* 30, 2 (2017), 519–549.

[7] Fabrice Benhamouda, Marc Joye, and BenoîT Libert. 2016. A New Framework for Privacy-Preserving Aggregation of Time-Series Data. *ACM Trans. Inf. Syst. Secur.* 18, 3, Article 10 (March 2016), 21 pages.

[8] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. 2009. Secure Multiparty Computation Goes Live. In *Financial Cryptography and Data Security, 13th International Conference, FC 2009 (Lecture Notes in Computer Science, Vol. 5628)*, Roger Dingledine and Philippe Golle (Eds.). Springer, 325–343.

[9] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1175–1191.

[10] Kevin D. Bowers, Marten van Dijk, Ari Juels, Alina Oprea, and Ronald L. Rivest. 2011. How to tell if your cloud files are vulnerable to drive crashes. In *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS*, Yan Chen, George Danezis, and Vitaly Shmatikov (Eds.). ACM, 501–514.

[11] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. 2020. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *Proc. Priv. Enhancing Technol.* 2020, 2 (2020), 459–480.

[12] T. H. Hubert Chan, Elaine Shi, and Dawn Song. 2012. Privacy-Preserving Stream Aggregation with Fault Tolerance. In *Financial Cryptography and Data Security*, Angelos D. Keromytis (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 200–214.

[13] Melissa Chase, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, and Peter Rindal. 2017. Private Collaborative Neural Network Learning. Cryptology ePrint Archive, Report 2017/762.

[14] Ivan Damgård and Mads Jurik. 2001. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *Public Key Cryptography*, Kwangjo Kim (Ed.). Springer, Berlin, Heidelberg, 119–136.

[15] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology – CRYPTO 2012*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 643–662.

[16] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9, 3–4 (Aug. 2014), 211–407.

[17] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 1322–1333.

[18] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving Machine Learning as a Service. *CoRR* abs/1803.05961 (2018). arXiv:1803.05961

[19] Marc Joye and Benoît Libert. 2013. A Scalable Scheme for Privacy-Preserving Aggregation of Time-Series Data. In *Financial Cryptography and Data Security*, Ahmad-Reza Sadeghi (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 111–125.

[20] Murat Koklu, Ilkay Cinar, and Yavuz Selim Taspinar. 2021. Classification of rice varieties with deep learning methods. *Computers and Electronics in Agriculture* 187 (2021), 106285.

[21] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010).

[22] Iraklis Leontiadis, Kaoutar Elkhiyaoui, and Refik Molva. 2014. Private and Dynamic Time-Series Data Aggregation with Trust Relaxation. In *Cryptology and Network Security*, Dimitris Gritzalis, Aggelos Kiayias, and Ioannis Askoxylakis (Eds.). Springer International Publishing, Cham, 305–320.

[23] Iraklis Leontiadis, Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva. 2015. PUDA – Privacy and Unforgeability for Data Aggregation. In *Cryptology and Network Security*, Michael Reiter and David Naccache (Eds.). Springer International Publishing, Cham, 3–18.

[24] Qian Lou, Bo Feng, Geoffrey Charles Fox, and Lei Jiang. 2020. Glyph: Fast and Accurately Training Deep Neural Networks on Encrypted Data. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.).

[25] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, 142–150.

[26] Kalikinkar Mandal and Guang Gong. 2019. PrivFL: Practical Privacy-Preserving Federated Regressions on High-Dimensional Data over Mobile Networks. In *Proceedings of the ACM SIGSAC CCSW*. Association for Computing Machinery, 57–68.

[27] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. 2016. Federated Learning of Deep Networks using Model Averaging. *CoRR* abs/1602.05629 (2016). arXiv:1602.05629

[28] Payman Mohassel and Peter Rindal. 2018. ABY3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. 35–52.

[29] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 19–38.

[30] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors. In *USENIX Security Symposium*. 619–636.

[31] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology — EUROCRYPT '99*, Jacques Stern (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 223–238.

[32] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. 2018. Scalable Private Learning with PATE. In *International Conference on Learning Representations, ICLR '18*. OpenReview.net.

[33] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *Proceedings of ASIACCS '18*. ACM, 707–721.

[34] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. 2018. A generic framework for privacy preserving deep learning. *CoRR* abs/1811.04017 (2018). arXiv:1811.04017

[35] Sinem Sav, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. 2021. POSEIDON: Privacy-Preserving Federated Neural Network Learning. In *NDSS '21*. The Internet Society.

[36] Lucas Schabhüser, Denis Butin, and Johannes Buchmann. 2019. Context Hiding Multi-key Linearly Homomorphic Authenticators. In *Topics in Cryptology – CT-RSA 2019*, Mitsuru Matsui (Ed.). Springer International Publishing, Cham, 493–513.

[37] Elaine Shi, T.-H. Hubert Chan, Eleanor Gilbert Rieffel, Richard Chow, and Dawn Song. 2011. Privacy-Preserving Aggregation of Time-Series Data. In *NDSS '11*. The Internet Society.

[38] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-Preserving Deep Learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, 1310–1321.

[39] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy, SP '17*. IEEE Computer Society, 3–18.

[40] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. 2019. A Hybrid Approach to Privacy-Preserving Federated Learning. In *Proceedings of AISec '19*. ACM, 1–11.

[41] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR* abs/1708.07747 (2017). arXiv:1708.07747

[42] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. 2019. HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning. *Proceedings of the ACM Workshop on Artificial Intelligence and Security - AISec'19*

(2019).

# A APPENDIX

## A.1 Ideal Functionality $\mathcal{F}_{\text{Global\_Train}}$

**Inputs:**
**Server:** Model $\theta$
**User ($U_i$):** Dataset $\{\mathcal{D}_i\}$ for all $i = 1 \dots n$
**Outputs:**
– **User ($U_i$):** Aggregated model $\theta$
– **Server:** Aggregated model $\theta$

## A.2 Proofs

Proof of Theorem 4.1.

Proof. Let $C$ denote the set of corrupted users, $\mathcal{L}$ denote the set of alive users, and $\mathcal{D}$ denote the set of dropout users where $\mathcal{U} = \mathcal{L} \cup \mathcal{D}$ and $\mathcal{L} = \mathcal{H} \cup C$. We construct a simulator $\mathcal{S}$ through a series of hybrids and we show that two consecutive hybrids are indistinguishable.

**Hyb 0:** This hybrid is a random variable corresponding to the real-world execution of the protocol $\Pi_{\text{Global\_Train}}$.

**Hyb 1:** This hybrid is exactly the same as **Hyb 0** except that the simulator $\mathcal{S}$ uniformly randomly samples the keys $AK_i$ of the honest users $U_i$ in $\mathcal{L}\backslash C$ that they share with the aggregator. The security of KDF assures that this hybrid is indistinguishable from **Hyb 0**.

**Hyb 2:** In this hybrid, all ciphertexts of the honest users in $\mathcal{H}$ that are sent to the aggregator are substituted with encryption of the zero local model. However, the honest users in $\mathcal{H}$ in the real world execution respond with the encryption of $\mathsf{Enc}_{\mathsf{pk}_{JL}}(\boldsymbol{\theta}^i), i \in \mathcal{H}$ with the key $AK_i$. Thus, the IND-CPA security of the encryption scheme $\mathbf{AE}.\mathbf{Enc}()$ guarantees that this hybrid is indistinguishable from the previous one.

**Hyb 3:** This hybrid is exactly same as the previous hybrid, except that $\mathsf{Enc}_{\mathsf{pk}_{JL}}(\boldsymbol{\theta})$ is substituted by encryption of the zero model, i.e., $\mathsf{Enc}_{\mathsf{pk}_{JL}}(0)$, which is indistinguishable due to the semantic security of the additive HE scheme. Further the simulator calls the adversary $\mathcal{A}$ to receive $\boldsymbol{\theta}$ which is obtained by decrypting $\mathsf{Enc}_{\mathsf{pk}_{JL}}(\boldsymbol{\theta})$. The sum of honest users' (in $\mathcal{H}$) local model, denoted by $\boldsymbol{\theta}^{\mathcal{H}}_{\text{loc}}$, can be derived as

$$\boldsymbol{\theta}^{\mathcal{H}}_{\text{loc}} = \sum_{i\in\mathcal{H}} \boldsymbol{\theta}^i_{\text{loc}} = \sum_{i\in\mathcal{L}} \boldsymbol{\theta}^i_{\text{loc}} - \sum_{i\in C} \boldsymbol{\theta}^i_{\text{loc}}$$

where $\mathcal{L}$ is the set of alive users, $C$ is the set of alive corrupted users and $\boldsymbol{\theta}^i_{\text{loc}}$ is the local model of the honest user $i \in \mathcal{H}$ trained using Algorithm 1. The local model of user $U_i$ can be represented as:

$$\boldsymbol{\theta}^i_{\text{loc}} = \boldsymbol{\theta} - \eta \cdot \sum_{e=1}^{1/b} \left( \frac{1}{|d|} \left( \sum_{\forall j} g_e(d_j) + \mathcal{N}\left(0, \frac{\sigma^2}{t} \cdot S_f^2\right)\right)\right)$$

where $e$ is the batch, $b$ is the batch rate, and $\boldsymbol{\theta}$ is the global model. The sum of the models of the honest alive users $\mathcal{H}$ is then,

$$\sum_{i\in\mathcal{H}} \boldsymbol{\theta}^i_{\text{loc}} = |\mathcal{H}| \cdot \boldsymbol{\theta} - \eta \cdot \sum_{e=1}^{1/b} \left( \frac{1}{|d|} \left( \sum_{i\in\mathcal{H}} \sum_{\forall j} g_e(d_j) + \mathcal{N}\left(0, \frac{|\mathcal{H}|\sigma^2}{t} \cdot S_f^2\right)\right)\right)$$

By averaging the aggregate local models we get $\frac{\sum_{i\in\mathcal{H}} \boldsymbol{\theta}^i_{\text{loc}}}{|\mathcal{H}|}$ which is

$$\boldsymbol{\theta} - \eta \cdot \sum_{e=1}^{1/b} \left( \frac{1}{|\mathcal{H}| \cdot |d|} \left( \sum_{i\in\mathcal{H}} \sum_{\forall j} g_e(d_j) + \mathcal{N}\left(0, \frac{|\mathcal{H}|\sigma^2}{t} \cdot S_f^2\right)\right)\right)$$

Since $t \leq |\mathcal{H}|$, the amount of noise added to $\boldsymbol{\theta}^{\mathcal{H}}_{\text{loc}}$ is sufficient to provide $(\epsilon, \delta)$-DP guarantee. As the addition of the Gaussian noises gives the $(\epsilon, \delta)$-indistinguishability, this hybrid is indistinguishable from **Hyb 2**.

**Hyb 4:** In this hybrid, the simulator works exactly the same way as in [36, Theorem 5], which outputs authenticators perfectly indistinguishable from the ones obtained by running MLHA.Eval. The joint view of colluding users in $C$ and the server in simulated world is indistinguishable from that of the real world. Therefore, this hybrid is indistinguishable from the previous one.

**Hyb 5:** Simulator repeats **Hyb 1** to **Hyb 4** for polynomial number of times, however, it remains indistinguishable from the real world due to composition theorem.

□

Proof of Theorem 4.3.

Proof. We construct a simulator $\mathcal{S}$ that runs the adversary and emulates the honest users outputs during the execution of the protocol.

**Hyb 0:** This hybrid corresponds to the execution of the real-world execution of the protocol $\Pi_{\text{Global\_Train}}$.

**Hyb 1:** The difference between this hybrid and **Hyb 1** is that in this hybrid, the simulator $\mathcal{S}$, working in the same way as in [36, Theorem 5]; uniformly samples $sk_i \in \mathcal{H}$ and computes $vk_i$ accordingly and uses the zero local model. The simulator calls the adversary $\mathcal{A}$ to obtain individual signed local models $\sigma_i$. The security of the MLHA signature scheme assures that **Hyb 0** and **Hyb 1** are indistinguishable.

**Hyb 2:** In this hybrid, the simulator calls the adversary $\mathcal{A}$ to obtain $C^i_\theta$, for$U_i \in \mathcal{H}$, $C^i_\theta$ is replaced by the encryption of the zero local model i.e., $\mathsf{Enc}_{\mathsf{pk}_{JL}}(\mathbf{0})$ for the honest users in $\mathcal{H}$. This hybrid is indistinguishable from the previous one due to the semantic security of the additive HE scheme.

**Hyb 3:** In this hybrid, the simulator calls the adversary $\mathcal{A}$ to obtain the encrypted model $\mathsf{Enc}_{\mathsf{pk}_{JL}}(\boldsymbol{\theta})$ and $\sigma$. The simulator substitutes the encrypted model with encryption of the zero model $\mathbf{0}$. The signature can be forged with a negligible probability. As the simulator has access to the local models of the users in $C$, the encryption of the sum of the honest users' local model can be computed as

$$\mathsf{Enc}_{\mathsf{pk}_{JL}}\left(\boldsymbol{\theta}^{\mathcal{H}}_{\text{loc}}\right) = \mathsf{Enc}_{\mathsf{pk}_{JL}}\left(\sum_{i\in\mathcal{H}} \boldsymbol{\theta}^i_{\text{loc}}\right) = \mathsf{Enc}_{\mathsf{pk}_{JL}}\left(\sum_{i\in\mathcal{L}} \boldsymbol{\theta}^i_{\text{loc}} - \sum_{i\in C} \boldsymbol{\theta}^i_{\text{loc}}\right) = \prod_{i\in\mathcal{H}} \mathsf{Enc}_{\mathsf{pk}_{JL}}(\boldsymbol{\theta}^i_{\text{loc}}).$$

Due to the semantic security of the additive HE scheme, $\mathsf{Enc}_{\mathsf{pk}_{JL}}\left(\boldsymbol{\theta}^{\mathcal{H}}_{\text{loc}}\right)$ is indistinguishable from $\mathsf{Enc}_{\mathsf{pk}_{JL}}(\mathbf{0})$ or any individual or combination of encrypted values. Thus **Hyb 3** is indistinguishable from **Hyb 2**.

**Hyb 4:** The simulator repeats **Hyb 1** to **Hyb 3** for a polynomial number of times, however, it remains indistinguishable from the real world due to the composition theorem.

□