Aggregation Service for Federated Learning: An Efficient, Secure, and More Resilient Realization

Yifeng Zheng, Shangqi Lai, Yi Liu, Xingliang Yuan, Xun Yi, and Cong Wang, *Fellow, IEEE*

Abstract—Federated learning has recently emerged as a paradigm promising the benefits of harnessing rich data from diverse sources to train high quality models, with the salient features that training datasets never leave local devices. Only model updates are locally computed and shared for aggregation to produce a global model. While federated learning greatly alleviates the privacy concerns as opposed to learning with centralized data, sharing model updates still poses privacy risks. In this paper, we present a system design which offers efficient protection of individual model updates throughout the learning procedure, allowing clients to only provide obscured model updates while a cloud server can still perform the aggregation. Our federated learning system first departs from prior works by supporting lightweight encryption and aggregation, and resilience against drop-out clients with no impact on their participation in future rounds. Meanwhile, prior work largely overlooks bandwidth efficiency optimization in the ciphertext domain and the support of security against an actively adversarial cloud server, which we also fully explore in this paper and provide effective and efficient mechanisms. Extensive experiments over several benchmark datasets (MNIST, CIFAR-10, and CelebA) show our system achieves accuracy comparable to the plaintext baseline, with practical performance.

Index Terms—Federated learning, secure aggregation, privacy, quantization, computation integrity

1 Introduction

Federated learning has rapidly emerged as a fascinating machine learning paradigm [1] which allows models to be trained on data dispersed over a number of mobile devices while each client can keep its dataset locally. Clients never share the raw datasets, and instead only periodically share model updates locally trained with their datasets, which are then aggregated by a coordinating server to produce a global model. Federated learning thus promises reaping the benefits of harnessing rich data from diverse sources to train high quality models, without clients being worried about the security and privacy risks of centralizing their raw data to a single place for training as in conventional practice.

While sharing model updates instead of raw datasets has greatly alleviated the privacy concerns as opposed to learning with centralized data, it still entails risks of private information leakage [2]. Protection of individual model updates is thus still necessary. Additively homomorphic encryption, which allows addition of private plaintext values to be securely performed over their ciphertexts, offers a feasible solution. Some recent works using this technique for building privacy-preserving federated learning systems have been presented [3], [4], [5], [6]. However, homomorphic encryption is expensive and also poses additional trust assumptions (e.g., sharing a private key across data holders) in the context of federated learning (see Section 2 for more discussion). A different work is due to Bonawitz et al. [7], which is free of expensive cryptography and allows lightweight encryption and aggregation.

When designing secure federated learning systems, resilience against drop-out clients failing to submit model updates for aggregation is a practical requirement as some clients may face issues like poor network connections, energy constraints, or temporary unavailability [8]. The work [7] provides mechanisms to handle drop-out clients, yet it undesirably prevents drop-out clients from directly and safely engaging in any future rounds of aggregation unless a new key setup is re-conducted. Therefore, how to achieve secure and lightweight processing in federated learning while maintaining practical drop-out resilience with no impact on drop-out clients' participation in any future rounds remains to be fully explored.

In addition, most of prior works largely overlook the communication efficiency aspect of federated learning in the encrypted domain, which could be affected due to ciphertext size expansion or loss of precision in training due to the adaption of plaintext processing for compatibility with cryptographic processing. Furthermore, most of them only provide semi-honest security against the passively adversarial server, assuming the server faithfully conduct the designated processing, and they are not resilient to an active adversary that may compromise the integrity of the processing at the server side.

In light of the above observations, in this paper, we present a new system design enabling federated learning services with lightweight secure and resilient aggregation. Our system enables clients holding proprietary datasets to only provide obscured model updates while aggregation can still be supported at a cloud server to produce a global

- *Yifeng Zheng is with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Shenzhen 518055, China. E-mail: yifeng.zheng@hit.edu.cn.*
- *Shangqi Lai and Xingliang Yuan are with the Faculty of Information Technology, Monash University, Australia. E-mail: shangqi.lai@monash.edu, xingliang.yuan@monash.edu.*
- *Yi Liu and Cong Wang are with the Department of Computer Science, City University of Hong Kong, Hong Kong. E-mail: 97liuyi@gmail.com, congwang@cityu.edu.hk.*
- *Xun Yi is with the School of Computing Technologies, RMIT University, Australia. E-mail: xun.yi@rmit.edu.au.*

model. By newly adapting a cherry-picked aggregation protocol for federated learning that we identified from the literature [9], [10], our system promises practical efficiency on encrypting the model updates at the client as well as aggregating the obscured model updates at the cloud server. No expensive cryptographic operations are required throughout the federated learning procedure. Meanwhile, our system is drop-out resilient and outperforms the best prior work [7] in the sense that the secret keys of drop-out clients are kept private so they still can directly participate at a later stage without the need for a new key setup.

With the above new secure design point for federated learning as a basis, we explore and present refinements in terms of boosted communication efficiency and stronger security. We start with consideration on the communication side, a known bottleneck for federated learning, due to various reasons like large sized models, limited client uplink bandwidth, and large numbers of participating clients [1]. We thus explore the potential of compressing model updates before secure aggregation so as to reduce the communication overheads in our system.

In the literature, various kinds of techniques for compressing the model updates have been proposed, such as sparsification, subsampling, and quantization [8]. Our observation is that quantization delicately represents the (fractional) values in a model update as integers, which is also the type of data required for secure aggregation. So our insight is to integrate the advancements in quantization with secure aggregation. However, most of existing quantization schemes are not secure aggregation friendly, as de-quantization requiring computation beyond summation has to be conducted before the quantized model updates can be aggregated. We make an observation that a newly developed quantization technique [6] (originally tailored for homomorphic encryption) can suit our purpose as no de-quantization is required before aggregation. Building on this new technique, our system allows clients to provide obscured quantized model updates, achieving a reduction in the communication overhead. To correctly integrate this quantization technique and make it function well, we address some practical considerations in our system including the prevention of overflow in aggregating quantized values and the identification of negative values in de-quantization.

Apart from the boosted communication efficiency side, we also investigate mechanisms to make our system more resilient, achieving stronger security against an actively adversarial cloud server, beyond the semi-honest security setting commonly assumed. The goal here is to ensure that the processing for the federated learning service is correctly enforced at the cloud server. Aiming for a pragmatic solution, we resort to the emerging techniques of hardware-assisted trusted execution environments (TEEs) to shield the computation integrity at the cloud server side throughout the federated learning procedure. As a practical instantiation, our system makes use of the increasingly popular Intel SGX [11], [12]. We do not rely on the confidentiality guarantee which is originally targeted by trusted hardware, due to the emergence of various side-channel attacks [13]. This is different from most of existing works which assume both confidentiality and integrity when using trusted hardware for secure computation. We give the abstract functionality assumed out of trusted hardware for our federated learning system, and further present a concrete protocol that renders federated learning with secure aggregation with computation integrity against an actively adversarial server.

We conduct extensive experiments over the popular benchmark datasets (MNIST, CIFAR-10, and CelebA), and different deep neural network models, with $21,840$ parameters, $23,272,266$ parameters, and $13,962,562$ parameters, respectively. We evaluate the accuracy evolution over varying rounds and demonstrate our system shares similar behavior with the plaintext baseline and achieve comparable accuracy, even if quantization is applied to the model updates (which achieves up to $4\times$ reduction in communication). The client-side security cost in encryption as well as in dealing with drop-out is thoroughly examined, which is on the order of a few seconds for the MNIST model and of a few minutes for larger CIFAR-10 and CelebA models.

Our evaluation on the server-side aggregation demonstrates that our system with security against an actively adversarial cloud server incurs almost no overhead over the semi-honest adversary setting. We also make a performance comparison with the state-of-the-art [7] (which exposes the secret keys of drop-out clients). The results demonstrate that our system is (up to $39\times$) much more efficient under zero client drop-out and only incurs small computation overhead (limited to $2.3\times$) under varying drop-out rates. Besides, our system (with a semi-honest cloud server) has less server computation as the drop-out rates and number of participating clients increase.

We highlight our contributions as follows:

- We present a new system design enabling federated learning services with lightweight secure and resilient aggregation. Compared to prior work, our system can handle client drop-out while keeping their secret keys confidential, so their direct participation in future rounds is not affected.
- We explore quantization-based model compression and newly make unique integration with secure aggregation to boost the communication efficiency in our system. We also present practical mechanisms to make our system more resilient, achieving stronger security against an actively adversarial server.
- We conduct extensive experiments over multiple real-world datasets and extensively evaluate the accuracy, client-side, and server-side performance. The results demonstrate that our system achieves accuracy comparable to the plaintext baseline, with practical performance.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 introduces some preliminaries. Section 4 gives a system overview. Section 5 presents the design of secure aggregation in our system. Section 6 shows how to integrate quantization to boost the communication efficiency. Section 7 gives the design of endowing our system with integrity. Section 8 presents the experiments. Section 9 concludes the whole paper.

2 Related Work

Our research is related to the line of work on federated learning with secure aggregation to protect the individual

model updates. To the best of our knowledge, none of existing works tackle exactly the same problem as our work, i.e., federated learning with secure and lightweight aggregation, drop-out resilience, and computation integrity against an adversarial server.

Some works rely on expensive homomorphic encryption [4], [5], [6], incurring high performance overheads. The use of homomorphic encryption also requires all clients to either share a common secret key [5], [6], or hold secret key shares which have to be generated via expensive multi-party protocols or distributed by a trusted third party (TTP) [4]. Xu et al. [3] propose a scheme using functional encryption, requiring a TTP for setting up keys as well as getting involved in the learning process. Our system is free of such a TTP.

In [7], Bonawitz et al. propose a lightweight encryption scheme supporting secure aggregation for federated learning. Their scheme can handle drop-out clients in an aggregation round, but would reveal their secret keys. When the scheme is used in federated learning, drop-out clients in a certain round are thus not able to directly and safely participate in any future rounds *unless* a new key setup is conducted. We note that the design of [7] has a double-masking mechanism, which is to prevent the server from learning the data of users who are too late in sending their masked vectors and are assumed to drop-out by the server (so the server recovers their secret keys).

There are some follow-up works [14], [15], [16] that attempt to the improve [7] from the *performance* aspect, yet they require additional assumptions regarding client topology information [14], [16] or delicate privacy-efficiency balance [15]. Specifically, the work of So et al. [14] applies a multi-group circular strategy for model aggregation, which partitions clients into communication groups and operates under a multi-group communication structure that relies on the topology of users and leads to a number of execution stages that has to be conducted sequentially across the groups. Each client in a group has to communicate with every client in the next group. The work of Choi et al. [16] has to appropriately organize clients in a graph structure based on the specific topology information and uses the graph to represent how public keys and secret shares are assigned to the other clients, so each client is aware of its neighboring clients. The work of Kadhe et al. [15] constructs a multi-secret sharing scheme and leverages it to design a secure aggregation scheme, which needs to delicately balance the trade-off between the number of secrets, privacy threshold, and dropout tolerance. Meanwhile, the resilience against client dropout is restricted to some pre-set *constant* fraction of clients.

The most related prior work thus is [7]. Compared with [7], our system also allows lightweight encryption for the clients but does not reveal the secret keys of drop-out clients in any rounds. That said, drop-out or non-selected clients in a certain round are still able to directly have safe participation in future rounds in our system. It is noted that the protocol in [7] consists of multiple rounds (including key setup), which need to be all executed over the clients selected in each iteration when applied to federated learning. Given the possibility of client drop-out being considered, it may not be promising to assume that each selected client in an iteration will well get involved in those interactions for key setup. A new key setup in each iteration of federated learning thus might hinder that iteration in proceeding normally. Indeed, once the number of dropped clients (with all interactions rounds considered in an iteration of federated learning) exceeds a threshold, the current iteration of federated learning has to abort and start over. In contrast, our protocol has minimal interactions among the selected clients and the server in each iteration of federated learning. In fact, as opposed to plaintext-domain federated learning, our protocol just needs one additional round of interaction for handling client drop-out upon secure aggregation. It is further worth noting that compared with existing works, our system ambitiously and newly provides assurance on computation integrity against the cloud server.

TABLE 1
Comparison of federated learning systems with secure aggregation. "**L**"ightweight cryptography."**R**"esilience against drop-out clients so that secure aggregation can still be correctly accomplished. "**K**"eys kept secret in handling dropouts so that drop-out clients' secret keys are not exposed. "**S**"ecurity for computation integrity against the server. "**C**"ompression and security co-design for high communication efficiency while ensuring security. "**I**"ndividual keys without TTP.

System	L	R	K	S	C	I
Xu et al. [3]	✗	✓	✓	✗	✗	✗
Phong et al. [5]	✗	✗	✓	✗	✗	✗
Truex et al. [4]	✗	✓	✓	✗	✗	✗
Zhang et al. [6]	✗	✓	✓	✗	✓	✗
Mandal et al. [17]	✗	✓	✓	✗	✗	✗
Bonawitz et al. [7]	✓	✓	✗	✗	✗	✓
This work	✓	✓	✓	✓	✓	✓

In an independent work, Mandal et al. [17] consider a special scenario where they hide the model from the clients and rely on homomorphic encryption to build specific federated linear/logistic regression models. Our system follows most prior works with focus on protecting individual model updates and is general for training any models. We note that some previous works [3], [4] also integrate orthogonal differential privacy techniques by adding calibrated noises to local model updates and lead to an inherent trade-off on accuracy and privacy with delicate parameter tuning, which is complementary to our system. Table 1 summarizes the comparison with the most related works discussed above.

Our work is also relevant to prior work on building hardware-assisted security applications. Most of existing works assume confidentiality and integrity guarantees from the trusted hardware (e.g., Intel SGX) for secure computation (e.g., [18], [19], [20], [21], [22], to just list a few). Given that the confidentiality guarantees may face threats from various side-channel attacks, a trending practice is to relax the trust assumptions and only assume integrity. A few works have been presented in different contexts including zero-knowledge proofs [13], multi-party machine learning inference [23], client-side checking of model training [24], and aggregation over blockchain-stored data under homomorphic encryption [25]. Inspired by this trend, our system only assumes minimally trusted hardware with integrity guarantees, and explores a new design point for enforcing server-side computation correctness throughout the federated learning procedure.

TABLE 2
Key Notations

Notation	Description
$\mathbf{w}$	Aggregate model
$\mathbf{w}_k^t$	Model update from client $\mathcal{C}_k$ in round t
(SK_i, PK_i)	Key pair of client $\mathcal{C}_i$ in secure aggregation
$CK_{i,j}$	Shared key computed from $\mathbf{SK}_i$ and $\mathbf{PK}_j$
$\mathcal{T}$	Set of selected clients in each round
$\mathcal{T}_o$	Set of online clients in each round
$\mathcal{T}_d$	Set of drop-out clients in each round
$\mathbf{r}_k$	The vector of blinding factors of client $\mathcal{C}_k$
Q_r	The r-bit quantizer used in our system
Q_r^{-1}	The r-bit de-quantizer used in our system
$(sk_{\mathcal{C}_i}, pk_{\mathcal{C}_i})$	Signing key pair of client $\mathcal{C}_i$

3 Preliminaries

3.1 Federated Learning

Federated learning enables multiple data owners to jointly solve an optimization problem which could be formulated as: $\min \sum_{s=1}^{S} \frac{1}{S} \cdot L(\mathbf{w}, \mathcal{D}_s)$, where S is the number of data owners, $L(\mathbf{w}, \mathcal{D}_s)$ is a loss function capturing how well the parameters $\mathbf{w}$ (treated as a flattened vector) model the local dataset $\mathcal{D}_s$. During the learning procedure, each data owner only shares a locally trained model update. The model updates are typically aggregated by a server and used to update the global model. This is an iterative procedure and runs in multiple rounds.

Each round proceeds through the following steps: (1) A fraction of the data owners (say K data owners) is selected by the server and a current global model $\mathbf{w}$ is sent to these data owners. (2) Each selected data owner then performs training over its local dataset, for which any optimizers could be used, though stochastic gradient descent (SGD) is the most popular one. With SGD, the k-th selected data owner updates the local model parameters via $\mathbf{w}_k \leftarrow \mathbf{w} - \eta \cdot \nabla L(\mathbf{w}; \beta)$, where β is a batch randomly sampled from the local dataset and η is the learning rate. A full iteration over the whole local dataset is referred to as an epoch, and the local training could be performed over multiple epochs. (3) Once the local training is done, an individual model update $\mathbf{w}_k$ is shared for aggregation: $\mathbf{w} = \frac{1}{K} \sum_{k=1}^{K} \mathbf{w}_k$, which produces an updated global model for next round. In Table 2, we provide a summary of the main notations used in this paper.

3.2 Transparent Enclave

Trusted hardware techniques (e.g., Intel SGX [11], [12]) allow the creation of protected memory regions called enclaves which are isolated from the rest of a host's software, including the operating system. Trusted hardware is designed with the aim of offering confidentiality and integrity assurance. Yet, it has been recently shown that the confidentiality guarantee may be compromised by a series of side-channel attacks [13]. The goal of achieving confidentiality with trusted hardware thus remains elusive so far. A recent trend on building trusted hardware based security applications [13], [23] is thus to only leverage the computational integrity. By relying on only the integrity, it is assumed that the internal states (i.e., all the code and data) of enclaves are transparent to the host (or the corrupted party). Only secure code attestation and secure signing functionality are required. Enclaves with such minimal trust assumption are referred to as transparent enclaves [13].

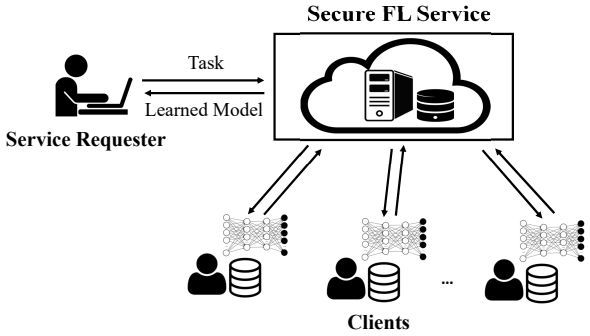

Fig. 1. Overview of our system architecture.

4 System Overview

4.1 Architecture

Fig. 1 illustrates our system architecture, which involves three actors: the service requester (or *requester* for short), clients, and the secure federated learning service which bridges the requester and clients. The requester wants to harness the power of federated learning and releases a task via the secure federated learning service. Such service could be hosted on the cloud, and run by a cloud server. Here the cloud server is an abstraction and can be implemented by an actual server or a cluster of servers. The clients are data owners interested in working on the federated learning task and sign up at the cloud server. All participants in the system agree in advance on a common machine learning model architecture and common learning objective. The clients may receive rewards from the requester for the participation.

To realize the secure federated learning service, our first aim is that in each round the cloud server should be able to perform aggregation of the individual model updates without seeing them in the clear. So, we will craft our design such that each client selected in a round can just provide an obscured model update, which still effectively allows aggregation. For high efficiency, our system will preclude the use of expensive homomorphic encryption and only rely on lightweight cryptography. Considering that communication is a known bottleneck in federated learning, our system will also cherry-pick and integrate appropriate quantization techniques to compress the model updates, so that a client can just share an obscured and quantized model update.

In addition to the confidentiality protection of individual model updates in each round of federated learning, our system also ambitiously aims to *efficiently* ensure that the federated learning service is correctly provisioned by the cloud server. While theoretically (expensive) cryptography-based verifiable computation techniques may help [26], we aim for a pragmatic solution and observe the trending practice is to leverage hardware-assisted trusted execution environments (TEE) [13]. Our system follows this trend and

only assumes a minimally trusted hardware with integrity guarantees. Specifically, we will rely on a transparent enclave to shield the computation integrity at the cloud server.

4.2 Threat Model

Our system for secure federated learning considers an adversarial cloud server. It could be semi-honest, which means the cloud server will honestly follow the protocol for the federated learning service, yet is curious about individual model updates so as to infer clients' local datasets. The semi-honest threat model is commonly adopted in privacy-preserving data-centric services in cloud computing [27], [28]. For this setting our system aims to maintain the confidentiality of individual model updates. The cloud server is only allowed to learn an aggregate model update and global model. Beyond the semi-honest adversary setting, a stronger threat model will also be considered where the cloud server may not correctly follow the designated computation. For this setting, our system further aims to enforce correct computation at the cloud server besides the confidentiality of individual model updates.

Collusion between the cloud server and a subset of selected clients in each round is also likely, and for such case our system aims to maintain the confidentiality of model updates of honest clients. Here, we are not concerned with the extreme (and also hardly realistic given that clients are many and geographically distributed) case that there is only one honest client and all other clients are compromised by the cloud server. Note that the assumption of at least one honest client is necessary in the problem of secure data aggregation and also (either explicitly or implicitly assumed) in existing works [7], [10], [29], [30]. It is obvious that for any practical secure aggregation schemes that securely compute the sum, if the aggregator colludes with all but one of the clients, the data of the colluding clients can be surely removed from the sum and the honest client's data will be revealed. Therefore, like existing works, our system does not provide data protection against the extreme colluding case.

It is noted that our system currently does not protect against possible attacks on the aggregate model updates and global model, which can directly be mitigated by letting clients do noise addition as per complementary differential privacy techniques [4]. Adversarial attacks like poisoning attacks [31] and backdoor attacks [32] are complementary research areas and out the scope of this work.

5 Empowering Federated Learning with Secure and Efficient Aggregation

5.1 Overview

Our system is aimed at federated learning service with secure aggregation that is free of heavy cryptography. Meanwhile, our system should be able to work without a trusted third party for setting up keys for secure aggregation and/or assistance in the learning procedure. All these practical requirements preclude considerations on the use of expensive homomorphic encryption which is also confronted with key distributions issues as mentioned before. Recall that although the secure aggregation scheme in [7] allows lightweight encryption and aggregation without a trusted third party, it reveals the secret keys of drop-out clients in an aggregation round, which consequently prevents them from direct participation in future rounds unless a new key setup is conducted in a subsequent round.

In our system, we propose to efficiently protect the confidentiality of individual model updates in federated learning with a cherry-picked low overhead cryptographic aggregation protocol [10]. We resort to this protocol due to the observations that it does not involve heavy cryptographic operations (only lightweight hashing and arithmetic operations are needed), does not require a trusted third party, and will not reveal the secret keys of drop-out clients. With this protocol as a basis, we craft a thorough design for federated learning with secure aggregation of individual model updates. Later in Section 6 and Section 7, we will work through the refinements that promise boosted communication efficiency and security.

5.2 Protocol

We now present the protocol for federated learning with secure aggregation of model updates. Note that as required by the cryptographic computation, the values in the vector representing the local model update should integers, which, however, could be fractional values in practice. We adopt a common scaling factor trick where a large enough scaling factor is used to scale up a fractional value into an integer. Specifically, given a fractional value v and a scaling factor L, we can obtain an integer representation of v as $\overline{v} = \lfloor v \cdot L \rfloor$. The approximate v can be later reconstructed as $\overline{v}/L$ [33]. Applying such trick in our context, we only need to scale down the aggregate model update. To ensure that the scaling operation does not compromise the quality of computation result, the message space for the scaled integers should be large enough (say 2^{32} or 2^{64}). Our protocol, as shown in Algorithm 1, proceeds as follows.

Initialization. Suppose that there are totally S clients in the system. We write $\mathcal{S}$ to denote the set of clients, in which each client is uniquely indexed by an integer $i \in [1, S]$. For initialization, each client generates its own key pair and leverages the cloud server as a central hub for distribution of public keys. Let $\mathbb{G}$ be a cyclic group of prime order p, with generator g. Also, let $\mathsf{H}:\{0,1\}^* \rightarrow \mathbb{Z}_p$ be a cryptographic hash function mapping arbitrary-length strings to integers in $\mathbb{Z}_p$. Each client $\mathcal{C}_i$ generates a private key $SK_i = x_i \in \mathbb{Z}_p$ and a public key $PK_i = g^{x_i} \in \mathbb{G}$, and uploads PK_i to the cloud server. Then, each client $\mathcal{C}_i$ downloads other users' public keys and computes $CK_{i,j} = \mathsf{H}((PK_j)^{x_i})$, where $i, j \in \mathcal{S}$ and $j \neq i$.

Secure federated learning. Without loss of generality, we describe here the secure aggregation process for federated learning in one round. The cloud server first randomly selects a fraction η of the clients. We write $\mathcal{T}$ to denote the set of selected clients. This set $\mathcal{T}$ and the current global model vector $\mathbf{w}$ ($\mathbf{w}$ is an initialized model when it is the first round) is sent to the selected clients. Each selected client $\mathcal{C}_k$ ($k \in \mathcal{T}$) then performs local training and produces a model update $\mathbf{w}_k$. An obscured model update is then generated based on blinding factors. In particular, it first generates a blinding factor for each element in the model update vector $\mathbf{w}_k$ and produces a vector $\mathbf{r}_k$ of blinding factors.

Algorithm 1 Our Design for Secure Federated Learning

Initialization:
for each client $\mathcal{C}_i$ **do**
  $SK_i = x_i \in \mathbb{Z}_p$.
  $PK_i = g^{x_i} \in \mathbb{G}$.
  Upload PK_i to the cloud server.
  for each $j \in \mathcal{S}, j \neq i$ **do**
    Download PK_j from the cloud server.
    Compute $CK_{i,j} = \mathsf{H}((PK_j)^{x_i})$.
  end for
end for
Cloud server executes:
Initialize $\mathbf{w}^0$.
for each round $t = 1, 2, \cdots$ **do**
  Select a fraction of the clients and produce the set $\mathcal{T}^t$.
  for each client $\mathcal{C}_k$ in $\mathcal{T}^t$ **do**
    $\mathbf{w'}_k^t \leftarrow$ ClientUpdate($\mathbf{w}^{t-1}$,$\mathcal{T}^t$).
  end for
  $\mathbf{w}' \leftarrow \sum_{k \in \mathcal{T}^t} \mathbf{w'}_k^t \bmod M$.
  $\mathbf{w}^t \leftarrow \frac{1}{|\mathcal{T}^t|} \cdot \mathbf{w}'$.
end for
ClientUpdate($\mathbf{w}^{t-1}$,$\mathcal{T}^t$):
$\mathbf{w} \leftarrow \mathbf{w}^{t-1}$.
Split the dataset $\mathcal{D}_k$ into batches $\mathcal{B}$.
for each local epoch e from 1 to E **do**
  for each batch β in $\mathcal{B}$ **do**
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \nabla L(\mathbf{w}; \beta)$. // η is the learning rate.
  end for
end for
$\mathbf{w}_k^t \leftarrow \mathbf{w}$.
Generate the vector $\mathbf{r}_k^t$ of blinding factors. Each element $\mathbf{r}_k^t(b) = \sum_{n \in \mathcal{T}, n \neq k} (-1)^{k>n} \mathsf{H}(CK_{k,n}||b||t)$.
Compute $\mathbf{w'}_k^t \leftarrow \mathbf{w}_k^t + \mathbf{r}_k^t \bmod M$.
Return $\mathbf{w'}_k^t$ to the cloud server.

The basic idea for supporting correct aggregation over obscured model updates is that if a client adds randomness to its input for blinding while another client subtracts that randomness from its input, the randomness will be canceled out when the summation of clients' obscured inputs is formed. In particular, each blinding factor $\mathbf{r}_k(b)$ for the b-th element in $\mathbf{w}_k$ is generated as $\mathbf{r}_k(b) = \sum_{n \in \mathcal{T}, n \neq k} (-1)^{k>n} \mathsf{H}(CK_{k,n}||b||t)$, where $(-1)^{k>n} = -1$ if $k > n$ and 1 otherwise, and t is a round counter. It can be observed that the sum of all blinding factors $\sum_{k \in \mathcal{T}} \mathbf{r}_k(b)$ is 0 as the blinding factors are canceled out when the sum over the set $\mathcal{T}$ of selected clients is formed.

Given above, with a model update $\mathbf{w}_k$ where each element is assumed to lie in a message space M (say $M = 2^{32}$), each client $\mathcal{C}_k$ generates a vector $\mathbf{r}_k$ of blinding factors as introduced above, and computes $\mathbf{w}'_k = \mathbf{w}_k + \mathbf{r}_k \bmod M$, where the modulo operation is performed element-wise. Upon receiving the obscured model updates, the cloud server computes $\sum_{k \in \mathcal{T}} \mathbf{w}'_k \bmod M = \sum_{k \in \mathcal{T}} (\mathbf{w}_k + \mathbf{r}_k) \bmod M$ and obtains $\sum_{k \in \mathcal{T}} \mathbf{w}_k \bmod M$ as $\sum_{k \in \mathcal{T}} \mathbf{r}_k = 0$.

5.3 Security Guarantees

We now provide security analysis for our above design below.

***Definition* 1.** *(Computational Diffie-Hellman (CDH) Problem). Consider a cyclic group $\mathbb{G}$ of prime order p with generator g. The CDH problem is hard if, for any probabilistic polynomial time algorithm $\mathcal{A}$ and random a and b drawn from $\mathbb{Z}_p$: $Pr[\mathcal{A}(\mathbb{G}; p; g; g^a; g^b) = g^{ab}]$ is negligible.*

***Theorem* 1.** *Given the hardness of the CDH problem, our system ensures that the cloud server only learns the aggregate model update without knowing individual model updates. In case of collusion between the cloud server and a subset of clients, the model updates of honest clients are still protected.*

Proof. Each value in the model update of the client $\mathcal{C}_k$ is masked by a unique blinding factor generated by the secret keys $\{CK_{k,n} : \mathsf{H}((g^{x_n})^{x_k})\}$. So we need to show that the cloud server is oblivious to the secret key $CK_{k,n}$. In our system, the cloud server only has access to the public key g^{x_i} of each client in $\mathcal{S}$, as per the system setup. The CDH problem ensures that given g^a and g^b, it is computationally hard to compute the value g^{ab}. So, given access to the public keys g^{x_k} and g^{x_n}, the cloud server is not able to infer the secret key $CK_{k,n} = \mathsf{H}((g^{x_n})^{x_k})$ for the generation of blinding factors. Next, we analyze the case of passive collusion between a subset of selected clients with the cloud server, where the corrupted clients share all their secret materials with the cloud server.

Without loss of generality, we give the proof for a certain honest client denoted by $\mathcal{C}_i$. Let $\mathcal{E}$ denote the set of clients participating in the same aggregation round t, $\mathcal{E}_h$ denote the set of honest clients, and $\mathcal{E}_c$ the subset of clients colluding with the cloud server. Recall that in our system the data submitted by the honest client $\mathcal{C}_i$ is $\mathbf{w}_i(b) + \mathbf{r}_i(b) = \mathbf{w}_i(b) + \sum_{n \in \mathcal{E}, n \neq i} (-1)^{i>n} \mathsf{H}(CK_{i,n}||b||t)$, for the b-th element in $\mathbf{w}_i$. Given $\mathcal{E}_h$ and $\mathcal{E}_c$, $\sum_{n \in \mathcal{E}, n \neq i} (-1)^{i>n} \mathsf{H}(CK_{i,n}||b||t)$ can be expressed as two parts: $\sum_{n \in \mathcal{E}_h, n \neq i} (-1)^{i>n} \mathsf{H}(CK_{i,n}||b||t)$ and $\sum_{n \in \mathcal{E}_c} (-1)^{i>n} \mathsf{H}(CK_{i,n}||b||t)$. So what the cloud server receives from the (honest) client is $\mathbf{w}_i + \sum_{n \in \mathcal{E}_h, n \neq i} (-1)^{i>n} \mathsf{H}(CK_{i,n}||b||t) + \sum_{n \in \mathcal{E}_c} (-1)^{i>n} \mathsf{H}(CK_{i,n}||b||t)$. As the set $\mathcal{E}_c$ of clients colludes with the cloud server, they are able to reveal to the cloud server the part $\sum_{n \in \mathcal{E}_c} (-1)^{i>n} \mathsf{H}(CK_{i,n}||b||t)$. So given the collusion with the set $\mathcal{E}_c$ of clients, the cloud server can obtain $\mathbf{w}_i + \sum_{n \in \mathcal{E}_h, n \neq i} (-1)^{i>n} \mathsf{H}(CK_{i,n}||b||t)$ ultimately. It can be seen that the honest client $\mathcal{C}_i$'s model update is still protected by blinding factors generated from the mutual secret keys established between this honest client and other honest clients in $\mathcal{E}_h$. That said, the cloud server still observes a randomly masked model update from the honest client $\mathcal{C}_i$. This completes the proof. □

5.4 Practical Considerations

Fault tolerance. It is likely that a selected client in a round may not actually participate (i.e., submitting a model update), due to reasons like poor network connections, energy constraints, or temporary unavailability. We call such clients

Algorithm 2 Fault Tolerance against Client Drop-out

Cloud server sends $\mathcal{T}_d$ to the clients in $\mathcal{T}_o$.
for each client $\mathcal{C}_k$ in $\mathcal{T}_o$ **do**
  Initialize a vector $\mathbf{q}_k$ of size $|\mathbf{w}'_k|$.
  for the b-th element in $\mathbf{q}_k$ **do**
    $\mathbf{q}_k(b) \leftarrow \sum_{n \in \mathcal{T}_d} (-1)^{k>n} \mathsf{H}(CK_{k,n}||b||t) \bmod M$.
  end for
  Send $\mathbf{q}_k$ to the cloud server.
end for
Cloud server computes $\mathbf{q} = \sum_{k \in \mathcal{T}_o} \mathbf{q}_k \bmod M$.
Cloud server computes $(\sum_{k \in \mathcal{T}_o} \mathbf{w}'_k) - \mathbf{q} \bmod M$.

drop-out clients. In such case, the cloud server would not be able to directly obtain the correct aggregation result, because the sum of the blinding factors is not zero. Therefore, our system should be able to handle the drop-out clients in a certain round and ensure that the obscured model updates of the responding clients can still be correctly aggregated.

We assume that the selected clients in a round which have managed to submit the (obscured) model updates will be able to stay online for potential assistance. The rationale behind such assumption is that clients will typically train and participate when their devices are charging and on an unmetered network [1], [34]. In practice, adequate incentive mechanisms could also be developed to further motivate clients' active participation.

To handle drop-out clients, the main insight is that the blinding factors of the responding clients should be eventually canceled out. Let $\mathcal{T}_o$ denote the set of responding clients who have submitted the (masked) model updates and are able to stay online for potential assistance, and $\mathcal{T}_d$ the set of drop-out clients who fail to submit the masked model updates. At a high level, each online client $\mathcal{C}_k \in \mathcal{T}_o$ generates a vector consisting of blinding factors that are generated based on the shared key $\{CK_{k,n}\}_{n \in \mathcal{T}_d}$ with the set of drop-out clients. This vector of blinding vectors allows the server to eliminate the blinding factors corresponding to the drop-out clients in each masked model update, and thus the aggregate model update. We show in Algorithm 2 the fault tolerance mechanism for handling drop-out clients.

Client dynamics. After the initial system setup, new clients may join and some clients may be revoked later. We now introduce how our system can support such client dynamics. When a new client $\mathcal{C}_i$ joins, it generates a secret key x_i and uploads its public key $PK_i{=}g^{x_i}$ to the cloud server, which then broadcasts it to other current clients in the system. Upon receiving PK_i, each client $\mathcal{C}_j$ computes $CK_{j,i}{=}(PK_i)^{x_j}$ to ensure that its blinding factor will correctly be generated for later use. On another hand, when a client $\mathcal{C}_i$ leaves the system, the cloud server only needs to inform each client $\mathcal{C}_j$ to discard the key $CK_{j,i}$.

Group management. In each round of federated learning, the client-side computation complexity in secure aggregation scales linearly in the size of the set $\mathcal{T}$, as generating the blinding factors requires each client to perform $O(|\mathcal{T}|)$ hashing operations. When $|\mathcal{T}|$ becomes quite large, it would be beneficial to improve the client side efficiency. One immediate optimization is that in each round, the selected clients can be divided into groups with sizes smaller than $\mathcal{T}$. During the process of secure aggregation, within-group aggregation is first performed at the cloud server, followed by cross-group aggregation.

Suppose that the selected clients are divided into s groups with equal sizes, the computation complexity is reduced to $O(|\mathcal{T}|/s)$ for each client participating in a round of federated learning. For client grouping, different guidelines might be adopted, such as geographical proximity. Note that group management also benefits fault tolerance. This is because each group is independent so the faults in a certain group will not affect other groups. Only the clients in the same group will be involved to handle the faults. The impact of faults on the overall system is thus effectively mitigated. One trade-off here is that the cloud server learns aggregated model updates from clients in independent groups. Yet, individual clients' model updates in each group are still protected. The group size can be set in advance according to the agreement between the cloud server and the clients.

6 Leveraging Quantization for Boosted Communication Efficiency

Communication could be a bottleneck for federated learning [8], because the trained machine learning models are usually of large sizes, clients may have limited uplink bandwidth, and the number of clients could be large [1]. As in each round a selected client communicates the model update to the cloud server, reducing the model update size could be highly beneficial to improve the communication efficiency of the whole system. In the literature, researchers have studied various kinds of techniques for compressing model updates such as sparsification, subsampling, and quantization [8]. Among these techniques, our observation is that quantization delicately represents the values in a model update as integers, exhibiting compatibility with cryptographic aggregation. Therefore, we choose to leverage the advancements in quantization to support secure aggregation while achieving a reduction in communication cost, and thus kill two birds with one stone.

6.1 Quantization Technique

Quantization based communication efficiency optimization for distributed machine learning has received considerable traction in recent years (e.g., [35], [36], [37], to just list a few). However, most of existing quantization techniques are not secure aggregation friendly, as they require a de-quantization operation, which demands computation much more complicated than secure aggregation, to be performed before doing the aggregation. What we need here is thus a quantization scheme which can support aggregation directly over quantized values.

We make an observation that a newly developed quantization technique [6] (originally tailored for homomorphic encryption) suits our purpose and has good compatibility with secure aggregation in our system. This technique quantizes fractional values into r-bit signed integers. In order to enable the values with opposite signs to be canceled out during summation, it proposes to make the quantized range symmetrical with respect to the range of the fractional values. Specifically, the r-bit quantizer Q_r that we integrate in our system works as follows. We start with

Algorithm 3 Secure Federated Learning with Quantization

Cloud server executes:
Initialize $\mathbf{w}^0$.
for each round $t = 1, 2, \cdots$ **do**
  Select a fraction of the clients and produces the set $\mathcal{T}^t$.
  for each client $\mathcal{C}_k$ in $\mathcal{T}^t$ **do**
    $\mathbf{w'}_k^t \leftarrow$ ClientUpdate($\mathbf{w}^{t-1}$,$\mathcal{T}^t$).
  end for
  $\Delta^t \leftarrow \sum_{k \in \mathcal{T}^t} \mathbf{w'}_k^t \bmod 2^r$.
  $\mathbf{w}^t = \mathbf{w}^{t-1} + \frac{1}{|\mathcal{T}^t|} \cdot Q_r^{-1}(\Delta^t)$.
end for

ClientUpdate($\mathbf{w}^{t-1}$,$\mathcal{T}^t$):
Produce a locally trained model $\mathbf{w}_k^t$ as in Algorithm 1 (steps 22-29).
$\Delta_k^t = \mathbf{w}_k^t - \mathbf{w}^{t-1}$.
Generate the vector $\mathbf{r}_k^t$ of blinding factors.
Compute $\mathbf{w'}_k^t \leftarrow Q_r(\Delta_k^t) + \mathbf{r}_k^t \bmod 2^r$.
Return $\mathbf{w'}_k^t$ to the cloud server.

introducing some notations. For any scalar $v \in \mathbb{R}$, we write $\mathsf{sgn}(v) \in \{-1, 1\}$ to denote the sign of v, with $\mathsf{sgn}(0) = 1$. We write $\mathsf{abs}(v)$ to denote the absolute value of v, and $\mathsf{round}(v)$ to denote standard rounding over v. Given a value v in the range $[-B, B]$, the quantizer Q_r quantizes it to an integer in $[-(2^{r-1}-1), 2^{r-1}-1]$ via:

$$Q_r(v) = \mathsf{sgn}(v) \cdot \mathsf{round}(\mathsf{abs}(v) \cdot (2^{r-1}-1)/B) \quad (1)$$

Given a quantized value $u = Q_r(v)$, the de-quantized value is computed as follows:

$$Q_r^{-1}(u) = \mathsf{sgn}(u) \cdot (\mathsf{abs}(u) \cdot B/(2^{r-1}-1)) \quad (2)$$

6.2 Secure Federated Learning with Quantization

We adapt the above quantization technique for our system to simultaneously achieve lightweight cryptographic aggregation and high communication efficiency. Such delicate integration leads to our refined protocol for secure federated learning, which is shown in Algorithm 3. Compared with the process shown in Algorithm 1, the main difference is that in each round, after training a local model, each selected client sends an obscured quantized model update to the cloud server. The cloud server then performs aggregation over these obscured quantized model updates and produces an updated global model.

Note that following prior works [35], [38], the quantization in our system is performed over the difference between a locally trained model $\mathbf{w}_k^t$ and the current global model $\mathbf{w}^{t-1}$, i.e., $\Delta_k^t = \mathbf{w}_k^t - \mathbf{w}^{t-1}$, as the model difference is more amenable to compression, in contrast to the locally trained model. Note that the quantizer requires the input to be bounded in a range $[-B, B]$. This can be achieved by clipping the values in Δ_k^t based on the threshold B which can be pre-set based on a public calibration dataset [39]. Values greater than B are set to B, and to $-B$ if they are smaller than B. In what follows, we further address some practical considerations.

Preventing overflow in aggregating quantized values. As the aggregation is performed over the quantized values from multiple clients, preventing overflow is crucially important. We note that this can be achieved via applying a scaling mechanism on the input range for quantization [6]. In particular, to prevent overflow when the aggregation is performed over c clients, we can set the input range for quantization as $[-c \cdot B, c \cdot B]$, in contrast with the original range $[-B, B]$. The intuition here is that by scaling the input range, each quantized value is scaled down c times so overflow can be prevented when c quantized values are aggregated.

Identifying negative values in de-quantization. It is noted that as the aggregate model update Δ^t is derived modulo 2^r, the cloud server needs to know which values in the Δ^t should have been negative indeed before performing the de-quantization. This can be achieved as follows. The cloud server checks if a value a in Δ^t is greater than $2^{r-1}-1$. if yes, its raw value can be obtained via such transformation $a' = a - 2^r$. Otherwise, it should have been non-negative. After this check and the transformation applied for values deemed to be negative, the cloud server can then proceed to the de-quantization.

1: Each client $\mathcal{C}_i$ generates the public key g^{x_i} and signature $\sigma_{\mathcal{C}_i} = \mathsf{Sign}_{sk_{\mathcal{C}_i}}(g^{x_i})$.
2: Each client $\mathcal{C}_i$ sends to g^{x_i} and $\sigma_{\mathcal{C}_i}$ to $\mathcal{F}_{\mathsf{TEE}}$ as well as revealed to the cloud server.
3: The cloud server invokes $\mathcal{F}_{\mathsf{TEE}}$ on ("Compute", $(\{g^{x_i}\}_{i\in\mathcal{S}}, \{\sigma_{\mathcal{C}_i}\}_{i\in\mathcal{S}})$), and receives $\perp$ or updated state containing $(\mathsf{ctr}, \{g^{x_j}\}_{j\in\mathcal{S}, j\neq i}, \sigma)$ to be sent to each client $\mathcal{C}_i$, where $\mathsf{ctr} = 0$.
4: Each client $\mathcal{C}_i$ runs $\mathsf{Verify}_{vk_\mathsf{T}}((\mathsf{ctr}, \{g^{x_j}\}_{j\in\mathcal{S}, j\neq i}), \sigma)$, upon receiving $(\mathsf{ctr}, \{g^{x_j}\}_{j\in\mathcal{S}, j\neq i}, \sigma)$.
5: Each client continues to produce the keys used for blinding factor generation if the check passes, or aborts otherwise.

Fig. 2. Key setup in security-hardened federated learning in our system.

7 Hardening Secure Federated Learning

In our design above, we assume a passively adversarial cloud server that faithfully performs the designated computation. We now present a practical design which can provide security against an actively adversarial cloud server with minimal performance overhead (as demonstrated in the experiments later), ensuring the correctness of computation at the cloud server. We aim for a pragmatic solution and thus following the trend of using trusted hardware, with only minimal assumption on its computation integrity assurance.

7.1 Abstract Functionality from Trusted Hardware

Inspired by the prior work [13], [23], we define the abstract functionality $\mathcal{F}_{\mathsf{TEE}}$ assumed out of the trusted hardware, which we use to harden the secure federated learning service in our system. The functionality is parameterized by a secure signing scheme $\{\mathsf{Sign}_{sk}, \mathsf{Verify}_{vk}\}$ with a key pair (sk, vk). Let $\mathsf{Sign}_{sk}(m)$ denote signing a message m and and $\mathsf{Verify}_{vk}(\sigma, m)$ denote verifying a signature σ on m. The functionality allows users to provide a program prog using the "install" command. It then returns $\alpha = \mathsf{Sign}_{sk}(\mathsf{Hash}(\mathsf{prog}))$ as a token for this program, which allows public integrity verification given the program prog and the signature verification key vk. Subsequent invocation

of $\mathcal{F}_{\mathsf{TEE}}$ runs prog on given inputs inp and fresh randomness rnd using the "Compute" command, and produces some output outp. The program prog could be stateful and may receive successive inputs in different rounds as indexed by a counter ctr and produce corresponding output $\mathsf{outp}_{\mathsf{ctr}}$.

Let $\mathsf{state}_{\mathsf{ctr}}$ be an internal state maintained by $\mathcal{F}_{\mathsf{TEE}}$. Upon the initialization, the state_0 is empty (i.e., $\mathsf{state}_0 = \epsilon$). On each "Compute" command for the ctr-th round, the functionality produces the output $\mathsf{outp}_{\mathsf{ctr}}$ and a signature σ_{ctr} on $(\mathsf{outp}_{\mathsf{ctr}}, \mathsf{ctr})$. Then, $\mathsf{state}_{\mathsf{ctr}-1}$ is updated to $\mathsf{state}_{\mathsf{ctr}}$ by adding the tuple $\{\mathsf{ctr}, \mathsf{inp}_{\mathsf{ctr}}, \mathsf{outp}_{\mathsf{ctr}}, \sigma_{\mathsf{ctr}}, \mathsf{rnd}_{\mathsf{ctr}}\}$. The updated state $\mathsf{state}_{\mathsf{ctr}}$ is always given to the host of the trusted hardware. This reflects the assumption that the internal state of the trusted hardware can be observed by the host. Note that for the output from the functionality, any parties with the public vk can verify the integrity. In short, we only assume that the functionality can conduct computation and produce signed outputs.

7.2 Security-Hardened Federated Learning

Given the functionality $\mathcal{F}_{\mathsf{TEE}}$, we now describe how to make our above federated learning service secure against with the cloud server that may not correctly conduct the designated processing. The main idea is to have every message sent by the (TEE-enabled) cloud server in the semi-honest protocol be computed by $\mathcal{F}_{\mathsf{TEE}}$, where the signing key pair $(sk_{\mathsf{T}}, vk_{\mathsf{T}})$ is used by $\mathcal{F}_{\mathsf{TEE}}$. These messages can be verified by any party which knows vk_{T}. We assume that every participant in the system knows vk_{T} in a reliable manner. Later we will show how this can be achieved via the remote attestation mechanism in the widely popular trusted hardware Intel SGX. We also assume that each client has a signing key pair $(sk_{\mathcal{C}_i}, pk_{\mathcal{C}_i})$, where the pubic key $pk_{\mathcal{C}_i}$ is bound to each client's identity $\mathcal{C}_i$.

We now describe our protocol for security-hardened federated learning. Firstly, the requester sends to the cloud server the code prog. The cloud server invokes $\mathcal{F}_{\mathsf{TEE}}$ on ("Install ", prog) to receive $(\mathsf{state}_0, \alpha = \mathsf{Sign}_{sk_{\mathsf{T}}}(\mathsf{Hash}(\mathsf{prog})))$, which is also sent to the requester and all clients in the system. The requester and each client run $\mathsf{Verify}_{vk_{\mathsf{T}}}(\alpha, \mathsf{Hash}(\mathsf{prog}))$, and abort if the check fails. After $\mathcal{F}_{\mathsf{TEE}}$ is initialized, the key setup for secure aggregation is conducted as shown in Fig. 2. After the key setup is done, secure federated learning now proceeds as shown in Fig. 3.

7.3 Realization of the Ideal Functionality $\mathcal{F}_{\mathsf{TEE}}$

The above functionality $\mathcal{F}_{\mathsf{TEE}}$ can potentially be realized via any trusted hardware techniques that provide code attestation and signing. As an instantiation, we resort to Intel SGX due to its wide integration in commodity processors. SGX enables the execution of programs in secure enclaves that are isolated from all other applications on the same host. It also provides attestation mechanisms which assure the integrity of the program loaded into the enclave once it has been attested. In particular, a signed attestation report can be generated for the program being loaded, which allows anyone to verify based on a public key corresponding to Intel's report signing key. The signing key pair $(sk_{\mathsf{T}}, vk_{\mathsf{T}})$ of the functionality $\mathcal{F}_{\mathsf{TEE}}$ can be generated inside the enclave, and the verification key vk_{T} can be part of the payload of the signed attestation report so that it is made public reliably. This verification key vk_{T} can then be used to verify the signed outputs from the functionality throughout the computation procedure in federated learning.

1: In the beginning of the round ctr, the cloud server invokes $\mathcal{F}_{\mathsf{TEE}}$ on ("Compute", ctr) and receives $(\mathsf{ctr}, \mathcal{T}_{\mathsf{ctr}}, \mathbf{w}^{\mathsf{ctr}-1}, \sigma)$ as the updated state, which is also sent to each client $\mathcal{C}_k$ in $\mathcal{T}_{\mathsf{ctr}}$.
2: Each selected client in $\mathcal{T}_{\mathsf{ctr}}$ runs $\mathsf{Verify}_{\mathsf{vk}_{\mathsf{T}}}((\mathsf{ctr}, \mathcal{T}_{\mathsf{ctr}}, \mathbf{w}_{\mathsf{ctr}-1}), \sigma)$, and aborts if the verification fails.
3: Each selected client $\mathcal{C}_k$ does local training and produces an obscured (quantized) model update $\mathbf{w}'_k$.
4: The (ctr, $\mathcal{C}_k$, $\mathbf{w}'_k$, $\sigma_{\mathcal{C}_k}$) is sent to $\mathcal{F}_{\mathsf{TEE}}$ as well as revealed to the cloud server.
5: The cloud server invokes $\mathcal{F}_{\mathsf{TEE}}$ on ("Compute", ctr, $\{\mathcal{C}_k, \mathbf{w}'_k, \sigma_{\mathcal{C}_k}\}_{k\in\mathcal{T}_{\mathsf{ctr}}}$). It receives $(\mathsf{ctr}+1, \mathcal{T}_{\mathsf{ctr}+1}, \mathbf{w}^{\mathsf{ctr}}, \sigma)$ for the next round, or $\perp$ if verification any selected clients' signatures fails.

Fig. 3. Aggregation in our security-hardened federated learning design.

8 Experiments

8.1 Setup

We implement the client with Python. The PyTorch is used for model training. For the federated learning service, we use C++ since the official SGX SDK only has C++ interfaces. We adopt Thrift [40] to enable cross-language communication. That is, the federated learning service is implemented as a daemon on a Thrift server that provides the corresponding interface which can be invoked by a service client generated as Python scripts. Each client uses the service client to connect with the server to upload the model update. The daemon passes the model update to the enclave and gets the global model and its signature as the result. In our experiments, we manage to get rid of expensive EPC paging in the enclave by temporarily storing all received (obscured) model updates and their signatures in the untrusted memory of the host server. Aggregation in the enclave is then performed by reading in and verifying the (obscured) model updates one by one.

We use three popular datasets in our experiments, including the commonly used datasets MNIST and CIFAR-10, and the CelebA dataset from the LEAF federated learning benchmark framework [41]. The MNIST dataset contains images of 0-9 handwritten digits, with 60000 training examples and 10000 testing examples. The CIFAR-10 dataset contains 50000 training color images and 10000 testing color images, with 10 classes. The CelebA dataset contains 200,288 celebrity images, each with 40 attribute annotations. In this paper we use this dataset for the application of gender classification. For MNIST, we use a relatively simple CNN model with two 5x5 convolution layers, a dropout layer, and two fully connected layers ($21,840$ total parameters). For CIFAR-10, we rely on the popular AlexNet [42] with more sophisticated structures and larger sizes but use less conv. kernels ($23,272,266$ total parameters). For CelebA, we use ResNet-18 [43], a popular CNN model that is 18 layers deep ($13,962,562$ total parameters). Hereafter, to facilitate

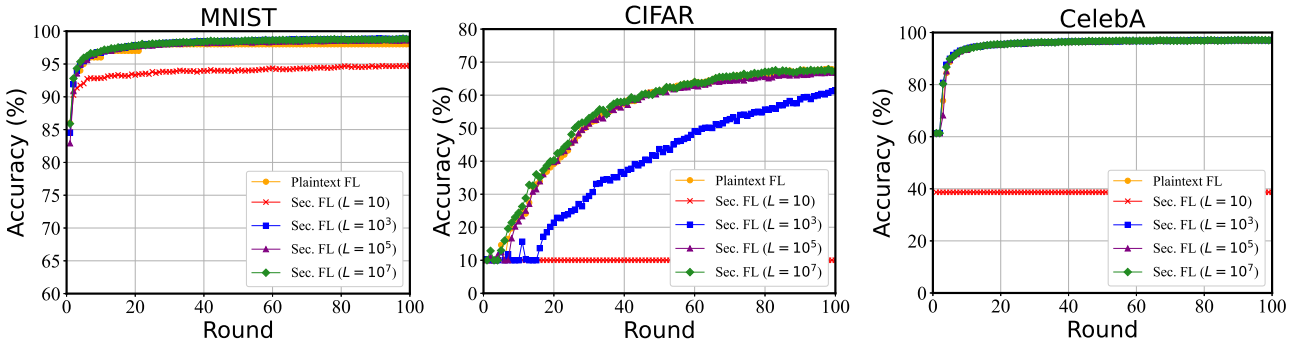

Fig. 4. Effect of the scaling factor L on accuracy in the proposed secure federated learning design over different datasets.

presentation, we will refer to the models over different datasets as the MNIST model, CIFAR model, and CelebA model, respectively. All evaluations are conducted on an SGX-enabled server equipped with Intel Xeon E 2288G 3.70GHz CPU (8 cores), 128GB RAM, and 3 RTX 3090 GPUs.

8.2 Accuracy Evaluation

We evaluate three cases: the plaintext federated learning with no privacy of model updates, our basic secure protocol without quantization, where a scaling factor is used to scale model parameters into integers for cryptographic computation, and our secure protocol extended with quantization.

For the datasets MNIST and CIFAR, we randomly shuffle the training examples and evenly distribute them across 100 clients. Each client receives 600 examples under the MNIST dataset, and 500 examples under the CIFAR-10 dataset respectively. For CelebA, we divide it into a training set and a test set with a 70/30 ratio, and distribute the training set to the clients, and each client holds about 2002 images. Such way of partitioning is referred to as IID data distribution [1]. The fraction of clients being selected in each round is set to 10%. Each selected client performs local training over 5 epochs, with the learning rate being 0.01 and batch size being 10. The threshold B related with quantization is set to 0.5 for the MNIST model and 0.1 for the CIFAR model and the CelebA model, respectively. For our secure protocol extended with quantization, we examine the cases where the quantization bit widths are 8 and 16 respectively.

To start, we evaluate the effect of varying scaling factors on the model accuracy in our secure federated learning design, of which the results are shown in Fig. 4. It is observed that the use of an appropriate scaling factor does not adversely affect the accuracy as the number of rounds grows. As long as a large scaling factor is used, the model accuracy is maintained with respect to the plaintext baseline. Such accuracy preservation is also consistent with the literature that uses the trick of scaling factor for cryptographic computation (e.g., [44], [45], [46], to just list a few). Hereafter, we set the scaling factor to 10^7 in all the remaining experiments.

In Fig. 5, we show the evolution of the testing accuracy under varying number of rounds over the MNIST dataset, CIFAR dataset, and CelebA dataset, respectively. It is noted that the legend "Plaintext FL" refers to the plaintext federated learning setting where the raw values of model updates in 32-bit floating-point representation are used. The legend

TABLE 3
Size of Model Updates (in MB)

Setting	MNIST	CIFAR	CelebA
Plaintext FL	0.083	88.78	53.26
Sec. FL (scaling)	0.083	88.78	53.26
Sec. FL (16-bit quantization)	0.042	44.49	26.63
Sec. FL (8-bit quantization)	0.021	22.19	13.32

"scaling" refers to the setting where a scaling factor (10^7) is used to scale up the fractional values in model updates into 32-bit integers to support computation in the cryptographic aggregation protocol. The legend "8/16-bit quantization" refers to the setting where a 8-/16-bit quantizer is applied over the fractional values in model updates, leading to 8-/16-bit integers for supporting communication efficiency optimization in the ciphertext domain. It can be observed that our secure protocols share similar behavior with the baseline and achieve comparable accuracy. The unique integration of quantization (with adequate bit width) does not adversely affect the quality of the trained model either.

In addition to the IID setting, following the seminal work [1] on federated learning, we further examine the non-IID setting over the MNIST dataset. The non-IID data distribution is set up by sorting the training images according to the digit labels, dividing them into shards of size 300, and randomly distributing two shards to each client. We show the accuracy evaluation result in Fig. 6, which demonstrates similar behavior and comparable accuracy to the plaintext baseline. Note that data heterogeneity is related to the federated learning paradigm itself and is independent of our security design. There are no ties between data heterogeneity and client drop-out. Client-dropout only affects the actual number of model updates being aggregated to produce the global model. Our system ensures the correctness of secure aggregation even in case of client-dropout.

8.3 Performance Evaluation

8.3.1 *Client-Side Performance*

We first examine the client-side computation performance. Recall that encrypting a model update requires generation of blinding factors that requires $O(|\mathcal{T}|)$ hashing operations per each, where $|\mathcal{T}|$ is the number of selected clients in a round. The encryption cost thus scales with $|\mathcal{T}|$ (and inherently the size of the model update). We show the client's

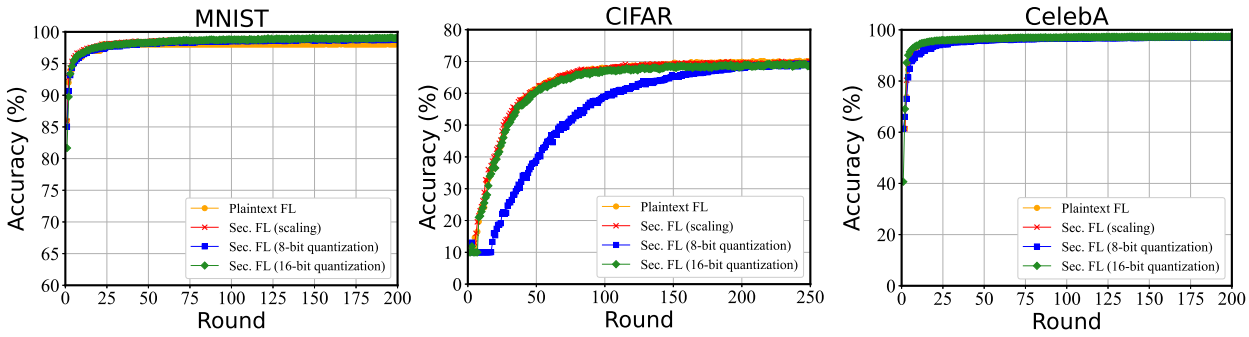

Fig. 5. Accuracy evolution of the models over different datasets.

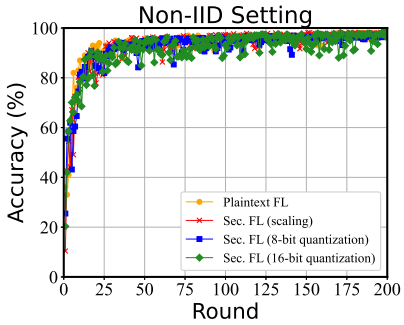

Fig. 6. MNIST model accuracy evolution under the non-IID setting.

running time of encrypting (quantized) model updates for varying fraction of clients being selected in a round, over different models in Fig. 7. For the smallest MNIST model, it is seen that the encryption cost is on the order of a few seconds. For the CIFAR and CelebA models, the running times are on the order of minutes due to their substantially larger model size ($23,272,266$ parameters and $13,962,562$ parameters as opposed to $21,840$ parameters of the MNIST model). However, it is worth noting that our system can flexibly support client grouping to largely limit the computation complexity of a client (i.e., independent of the fraction), as demonstrated in Fig. 7 where the selected clients are grouped with size 10.

We also examine the computation cost of a client in a recovery phase to handle dropouts, with the results are shown in Fig. 8, under varying dropout rates over different models. As expected, the running time of the client scales linearly with the dropout rate. It is also revealed that client grouping can greatly reduce the cost.

In Table 3, we report the size of a model update (i.e., sizes of model update parameters), for the plaintext case and our secure design under the settings of scaling, 8-bit quantization, and 16-bit quantization, respectively. Our secure protocol with scaling incurs no overhead on the model update size, as the bit precision remains the same. Our secure protocol can achieve $4\times$ reduction under 8-bit quantization and $2\times$ reduction under 16-bit quantization on the size of transferred model update in a round. Recall that our system exhibits similar behavior in accuracy evolution with regard to varying number of rounds. Given a target number of rounds, our system with quantization can lead to $2\times$ or $4\times$ reduction on the communication, with comparable accuracy to the plaintext baseline.

TABLE 4
Performance Complexity Comparison

Approach	Client Computation	Server Computation
Ours	$O(mn + md)$	$O(m(n-d))$
[7]	$O(n^2 + mn)$	$O(m(n-d) + md(n-d))$

8.3.2 Cloud Server-Side Performance

We now examine the costs of securely aggregating model updates to produce an updated global model under the semi-honest adversary setting and active-adversary setting respectively. The results are plotted in Fig. 9. As expected, the computation costs scales linearly with the fraction. It is revealed that our protocol with security against an active adversarial cloud server (i.e., computation integrity against the cloud server) incurs almost no overhead over the semi-honest setting ($1\times$, $1.005\times$, and $1.013\times$ over the MNIST, CIFAR, and CelebA models respectively). Such minimal performance overhead is promised as no paging is required.

8.3.3 Comparison with Prior Work

We now make comparison with the most related prior work [7] without heavy cryptographic operations. Firstly, we compare the computational complexity on the client side and on the cloud server side respectively. Suppose the dimension of each model update vector is m, and the number of clients being selected in a round of the federated learning procedure is n, and the number of dropped clients is d. Overall the secure aggregation approach in our system leads to $O(mn + md)$ computation on the client side and $O(m(n-d))$ computation on the cloud server side. In comparison, according to [7], their scheme leads to $O(n^2 + mn)$ computation on the client side and $O(m(n-d)+md(n-d))$ on the cloud server side. Table 4 summarizes the comparison of the asymptotic computational complexity.

We note that the work [7] does not present real machine learning based experiments. To have empirical performance comparison with [7], we test their scheme[1] over the MNIST model to measure the client-side and server-side runtime costs, with varying dropout rates and fraction of selected clients per around. Table 5 gives the comparison of the client-side cost. Our design is (up to $39\times$) more efficient than

1. Implementation used: https://github.com/55199789/PracSecure

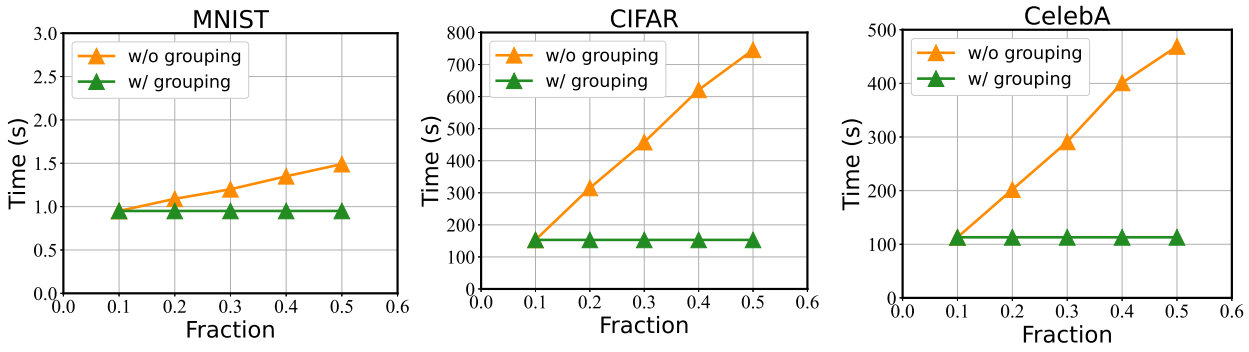

Fig. 7. Client's encryption cost with varying fraction of selected clients per round, over different models.

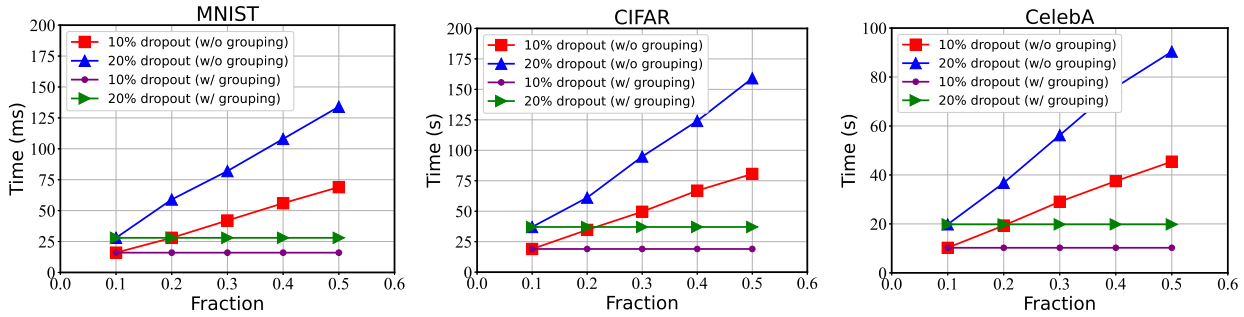

Fig. 8. Client's computation cost in dealing with different dropout rates, over different models.

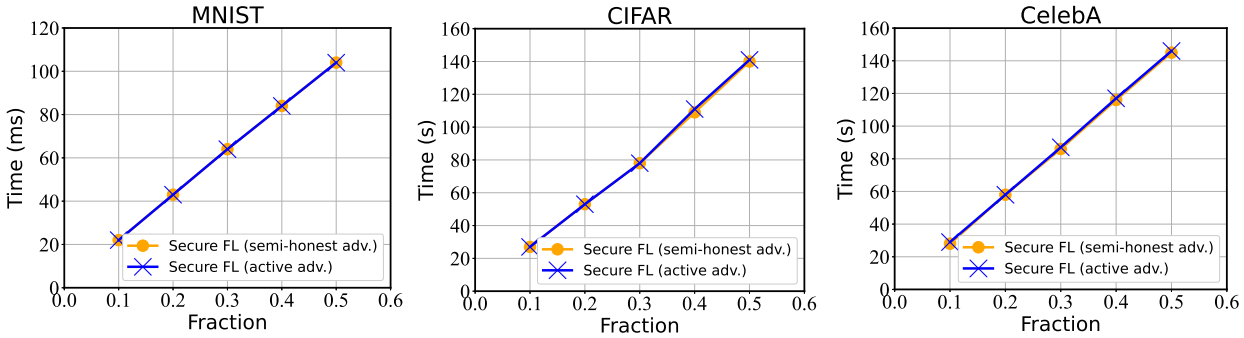

Fig. 9. Cost of aggregating model updates at the cloud server, under different adversary settings and over different models

.

TABLE 5
Client Cost Comparison with Prior Work [7]

Dropouts	Client Cost (ms)	Fraction				
		0.1	0.2	0.3	0.4	0.5
0%	Ours	0.95	1.09	1.2	1.35	1.49
	[7]	12.05	23.38	34.98	47.16	58.43
10%	Ours	16	28	42	56	69
	[7]	11.98	23.42	35.04	46.91	58.32
20%	Ours	28	59	82	108	134
	[7]	11.87	23.42	35.77	47.02	58.3

TABLE 6
Server Cost Comparison with Prior Work [7]

Dropouts	Server Cost (ms)	Fraction				
		0.1	0.2	0.3	0.4	0.5
0%	Ours	22	43	64	84	104
	[7]	5.74	11.26	16.32	22.59	29.66
10%	Ours	38	79	115	153	196
	[7]	13.53	41.15	83.53	141.98	220.5
20%	Ours	35	75	111	148	189
	[7]	19.83	64.17	138.89	238.24	366.59

the work [7] when the dropout rate is zero. As the dropout rate increases, our design has higher client cost (limited to 2.3×), as each online client assists by computing blinding factors scaling to the number of drop-out clients.

Regarding the server-side cost, it is observed that as the dropout rate and the fraction of selected clients increase, our design (semi-honest adv. setting) consumes less computation. This is because the server in their scheme

needs to perform reconstruction of secret keys over collected secret shares and re-compute masks to be subtracted from the aggregate sum over the online clients. We emphasize that unlike [7], our system does not reveal the secret keys of drop-out clients, so their direct participation in future rounds is not affected. Meanwhile, our system can also provide much stronger security (computation integrity) against the server with minimal overhead, as demonstrated above.

9 Conclusion

In this paper, we present a system design for federated learning, which allows clients to provide obscured model updates while aggregation can still be supported. Our system first departs from prior works by building on a cherry-picked cryptographic aggregation protocol, which promises the advantages of lightweight encryption and aggregation as well as the ability to handle drop-out clients without exposing their secret keys. For higher communication efficiency, our system also adapts the latest advancements in quantization techniques for compressing individual model updates. Furthermore, our system also provides security beyond the common semi-honest adversary setting, ensuring the computation integrity at the cloud server. We conduct an extensive evaluation over popular benchmark datasets, and the results validate the practical performance of our system.

Acknowledgement

This work was supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2021A1515110027, in part by the Shenzhen High-Level Talents Research Start-up Fund, in part by the Australian Research Council (ARC) Discovery Projects under Grants DP200103308 and DP180103251, in part by a Monash-Data61 collaborative research project (Data61 CRP43), in part by the Research Grants Council of Hong Kong under Grants CityU 11217819, 11217620, 11218521, N_CityU139/21, R6021-20F, and RFS2122-1S04, in part by Shenzhen Municipality Science and Technology Innovation Commission under Grant SGDX20201103093004019, and in part by the National Natural Science Foundation of China under Grant 61572412.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. of AISTATS*, A. Singh and X. J. Zhu, Eds., 2017.

[2] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Proc. of IEEE S&P*, 2019.

[3] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, "Hybridalpha: An efficient approach for privacy-preserving federated learning," in *Proc. of AISec*, 2019.

[4] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proc. of AISec*, 2019.

[5] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[6] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *Proc. of USENIX ATC*, 2020.

[7] 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 *Proc. of ACM CCS*, 2017.

[8] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[9] K. Kursawe, G. Danezis, and M. Kohlweiss, "Privacy-friendly aggregation for the smart-grid," in *Proc. of PETS*, 2011, pp. 175–191.

[10] L. Melis, G. Danezis, and E. D. Cristofaro, "Efficient private statistics with succinct sketches," in *Proc. of NDSS*, 2016, pp. 1–15.

[11] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proc. of Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

[12] Intel, "Intel software guard extensions," On line at: https://software.intel.com/en-us/sgx, 2020.

[13] F. Tramèr, F. Zhang, H. Lin, J. Hubaux, A. Juels, and E. Shi, "Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge," in *Proc .of IEEE EuroS&P*, 2017.

[14] J. So, B. Guler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *CoRR*, vol. abs/2002.04156, 2020.

[15] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramchandran, "Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning," *CoRR*, vol. abs/2009.11248, 2020.

[16] B. Choi, J. Sohn, D. Han, and J. Moon, "Communication-computation efficient secure aggregation for federated learning," *CoRR*, vol. abs/2012.05433, 2020.

[17] K. Mandal and G. Gong, "Privfl: Practical privacy-preserving federated regressions on high-dimensional data over mobile networks," in *Proc. of CCSW*, 2019.

[18] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "VC3: trustworthy data analytics in the cloud using SGX," in *Proc. of IEEE S&P*, 2015.

[19] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *Proc. of USENIX Security*, 2016.

[20] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A. Sadeghi, G. Scerri, and B. Warinschi, "Secure multiparty computation from SGX," in *Proc. of FC*, 2017.

[21] J. I. Choi, D. J. Tian, G. Hernandez, C. Patton, B. Mood, T. Shrimpton, K. R. B. Butler, and P. Traynor, "A hybrid approach to secure function evaluation using SGX," in *Proc. of AsiaCCS*, 2019.

[22] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *Proc. of ICLR*, 2019.

[23] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," *Proc. of IEEE S&P*, 2020.

[24] X. Zhang, F. Li, Z. Zhang, Q. Li, C. Wang, and J. Wu, "Enabling execution assurance of federated learning at untrusted participants," in *Prof. of IEEE INFOCOM*, 2020.

[25] H. Duan, Y. Zheng, Y. Du, A. Zhou, C. Wang, and M. H. Au, "Aggregating crowd wisdom via blockchain: A private, correct, and robust realization," in *Proc. of PerCom*, 2019.

[26] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "Verifynet: Secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2020.

[27] Y. Zheng, H. Duan, X. Tang, C. Wang, and J. Zhou, "Denoising in the dark: Privacy-preserving deep neural network-based image denoising," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 3, pp. 1261–1275, 2021.

[28] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "Medisc: Towards secure and lightweight deep learning as a medical diagnostic service," in *Proc. of ESORICS*, 2021.

[29] Q. Li, G. Cao, and T. F. L. Porta, "Efficient and privacy-aware data aggregation in mobile sensing," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 2, pp. 115–129, 2014.

[30] Y. Zhang, Q. Chen, and S. Zhong, "Efficient and privacy-preserving min and kth min computations in mobile sensing systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 1, pp. 9–21, 2017.

[31] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *Proc. of Usenix Security*, 2020.

[32] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proc. of AISTATS*, 2020.
[33] C. Wang, K. Ren, J. Wang, and Q. Wang, "Harnessing the cloud for securely outsourcing large-scale systems of linear equations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1172–1181, 2013.
[34] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecný, S. Mazzocchi, B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," in *Proc. of MLSys*, 2019.
[35] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization," in *Proc. of AISTATS*, 2020.
[36] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: communication-efficient SGD via gradient quantization and encoding," in *Proc. of NeurIPS*, 2017.
[37] N. Shlezinger, M. Chen, Y. C. Eldar, H. V. Poor, and S. Cui, "Federated learning with quantization constraints," in *Proc. of IEEE ICASSP*, 2020.
[38] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *Proc. of NIPS Workshop on Private Multi-Party Machine Learning*, 2016.
[39] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. of ACM CCS*, 2015.
[40] Apache, "Thrift," On line at: https://thrift.apache.org, 2020.
[41] P. W. T. L. J. K. H. B. M. V. S. Sebastian Caldas, Sai Meher Karthik Duddu and A. Talwalkar, "Leaf: A benchmark for federated settings," in *Prof. of Workshop on Federated Learning for Data Privacy and Confidentiality*, 2019.
[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of NIPS*, 2012.
[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of IEEE CVPR*, 2016.
[44] Y. Zheng, H. Duan, X. Yuan, and C. Wang, "Privacy-aware and efficient mobile crowdsensing with truth discovery," *IEEE Trans. Dependable Secur. Comput.*, vol. 17, no. 1, pp. 121–133, 2020.
[45] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proc. of ACM CCS*, 2017.
[46] Y. Zheng, H. Duan, and C. Wang, "Learning the truth privately and confidently: Encrypted confidence-aware truth discovery in mobile crowdsensing," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2475–2489, 2018.