VeriFL: Communication-Efficient and Fast Verifiable Aggregation for Federated Learning

Xiaojie Guo, Zheli Liu, Jin Li, *Senior Member, IEEE*, Jiqiang Gao, Boyu Hou, Changyu Dong, and Thar Baker, *Member, IEEE*

***Abstract*—Federated learning (FL) enables a large number of clients to collaboratively train a global model through sharing their gradients in each synchronized epoch of local training. However, a centralized server used to aggregate these gradients can be compromised and forge the result in order to violate privacy or launch other attacks, which incurs the need to verify the integrity of aggregation. In this work, we explore how to design communication-efficient and fast verifiable aggregation in FL. We propose VeriFL, a verifiable aggregation protocol, with $O(N)$ (dimension-independent) communication and $O(N+d)$ computation for verification in each epoch, where N is the number of clients and d is the dimension of gradient vectors. Since d can be large in some real-world FL applications (e.g., 100K), our dimension-independent communication is especially desirable for clients with limited bandwidth and high-dimensional gradients. In addition, the proposed protocol can be used in the FL setting where secure aggregation is needed or there is a subset of clients dropping out of protocol execution. Experimental results indicate that our protocol is efficient in these settings.**

***Index Terms*—Federated learning, verifiable aggregation, linearly homomorphic hash, commitment, machine learning.**

I. Introduction

FEDERATED learning (FL) [1]–[3] has become one of the most popular paradigms for distributed machine learning nowadays. It facilitates the usage of sensitive datasets distributed among a large number of clients, which may be mobile phones, other mobile devices, or sensors, without collecting their data. In FL, to train a global model, a subset of users are instructed to upload their local parameters in each synchronous epoch (Figure 1(a)). To update the global model, a centralized server is adopted to aggregate the parameters received from these users and sends back the updated global model to them. All users will update their local model according to the global one and such training process will continue until model convergence.

Manuscript received June 7, 2020; revised September 19, 2020; accepted November 25, 2020. Date of publication December 7, 2020; date of current version December 30, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 62032012, in part by the National Key Research and Development Program of China under Grant 2020YFB1005700, and in part by CNKLSTISS. The associate editor coordinating the review of this manuscript and approving it for publication was Mr. Frederik Armknecht. *(Corresponding author: Zheli Liu.)*

Xiaojie Guo, Zheli Liu, Jiqiang Gao, and Boyu Hou are with the College of Cyber Science, the College of Computer Science, and the Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin 300071, China (e-mail: xiaojieg.veda@gmail.com; liuzheli@nankai.edu.cn; jiqiang.gao@mail.nankai.edu.cn; houboyu@dbis.nankai.edu.cn).

Jin Li is with the School of Computer Science, Guangzhou University, Guangzhou 510006, China, and also with the College of Cyber Science, Nankai University, Tianjin 300071, China (e-mail: jinli71@gmail.com).

Changyu Dong is with the School of Computing, Newcastle University, Newcastle upon Tyne NE1 7RU, U.K. (e-mail: changyu.dong@ncl.ac.uk).

Thar Baker is with the Department of Computer Science, College of Computing and Informatics, University of Sharjah, Sharjah, United Arab Emirates (e-mail: tshamsa@sharjah.ac.ae).

Digital Object Identifier 10.1109/TIFS.2020.3043139

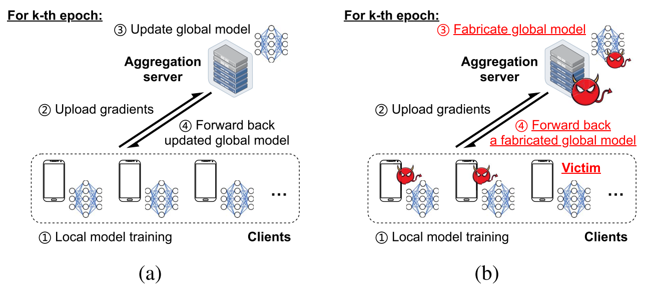


Fig. 1. Federated learning and a trivial attack on it. (a) The framework of federated learning. (b) The attack launched by the adversary corrupting the server and a subset of clients.

Despite of its appealing functionality, FL has been shown vulnerable to some attacks. For example, the works [4]–[6] show that the gradient vector uploaded by a client may leak sensitive information about its private dataset. To address this issue, in [7], the authors proposed a *secure aggregation* protocol that guarantees the privacy of gradients. However, the recent work [8] shows that, in addition to the gradient privacy, the integrity of aggregation should be protected as well. In particular, the server can easily become a single-point of failure in FL. Without integrity guarantee, once the server gets compromised, the adversary controlling the server can manipulate the global model and cause misclassification of any involved client at its specified data point (Figure 1(b)), which is similar to the consequence of backdoor attacks [9]–[12]. The lack of the integrity guarantee of aggregation may restrict the commercial application of FL. An example attack is that the corrupted server *re-trains* the global model updated in this epoch with some poisoned data and returns the re-trained model to honest clients, aiming to cause misclassification in these clients.

We note that attacks by modifying aggregation results (e.g., the aforementioned example attack) can be mitigated using verifiable aggregation protocols. In such protocols for FL, a considered adversary cannot convince an honest client to accept its forged aggregated gradient as a real one with an overwhelming probability (see Definition 3). Certainly, to resist inference attacks exploiting the "non-encrypted" gradients [4]–[6], the used verifiable aggregation protocol should be *secure* as well to guarantee input privacy of each client. Informally speaking, this means that an adversary in the aggregation protocol learns nothing about the gradient of each client from protocol transcripts (see Definition 4).

In this paper, we focus on how to realize a (secure) verifiable aggregation protocol in a communication-efficient and fast way on resource-constrained devices (e.g., the setting of mobile devices in Google's work [7]). Since the bandwidth in the resource-constrained setting is expensive and a large number of clients are required to train a global model through hundreds of iterations, communication becomes one of the most important considerations in the design of FL protocol. If the designed verifiable aggregation protocol costs a lot in terms of communication and computation, it will take a long period to finish the training process. There are many factors influencing the communication efficiency of a verifiable aggregation protocol, and the number of parameters in the model to be trained is particularly important. So, our first question is, *can we design a verifiable aggregation protocol in which the communication of verification is independent of the number of parameters in the model to be trained*? In fact, the communication efficiency of FL is always a research hotspot [1], [2], [7], [13].

To the best of our knowledge, there is no previous work that succeeds in designing a communication-efficient verifiable aggregation protocol. The works about communication-efficient FL [7], [13] do not consider how to guarantee the integrity of aggregation. The recent work that achieves the first secure verifiable aggregation [8] uses zero-knowledge proof to enforce honest aggregation. However, its communication cost for verification is *linearly* dependent on the dimension of gradient vectors. Its linearly-growing and high communication cost makes it impractical in its intended application scenarios, that is FL among mobile devices. Moreover, its computation cost for verification is unsatisfactory due to the use of zero-knowledge. For example, it takes 3.59 MB outgoing communication and 254964 ms computation per client to verify the aggregation of 20K-dimensional parameter vectors among 500 clients. Another trivial solution to achieve verifiable aggregation is using generic multiparty computation (MPC) approaches [14]–[21], but its communication and computational overhead are very large. So, in addition to communication efficiency, our second question is, *how to make our verifiable aggregation protocol computation-efficient*?

A. Challenge and Contribution

1) Challenge: To verify the integrity of aggregation, a straightforward idea is to use homomorphic hash to compute the "digest" of gradient vectors to be aggregated. A combined hash can be obtained from these hash values after aggregation, and each client compares it with the hash of the aggregation result (i.e., real hash) to verify the aggregation. Although this idea yields a desirable feature that the verification communication is independent of gradient dimension, we would like to show that this idea provides little security guarantee.

The security challenge comes from the fact that everyone in the above construction has access to the homomorphic hash function. In the simulation-based proof, the simulator did not know the input (i.e., the gradient vector) of each honest client by the time it was asked to simulate its hash value. To simulate it, the simulator has to use a dummy vector. However, this dummy value is different from the real one with an overwhelming probability. Since everyone (including the adversary) can call the homomorphic hash function with the aggregation result, an inconsistency between the combined and real hashes will be found after aggregation and this immediately yields an efficient distinguisher that fails the simulation. That is, the straightforward construction is *not* secure.

In [8], the authors solve this security challenge by combining homomorphic hash with zero-knowledge proof. However, due to the dimension-wise zero-knowledge proofs, their protocol leads to expensive computation and the communication caused by verification is still linear in the gradient dimension. We note that such dimension-dependence is depressing since the gradient dimension is large in real-world FL applications (e.g., 100K). We are motivated to find another way to achieve verifiable aggregation, overcoming the security challenge with good concrete efficiency in terms of communication and computation.

2) Contribution: In this paper, we present a communication-efficient and fast protocol, VERIFL, for verifiable aggregation in FL. As shown by experiment, for example, VERIFL achieves 33.24 KB (dimension-independent) outgoing communication and 8899 ms computation per client to verify the aggregation of 100K-dimensional parameter vectors among 500 clients, which outperforms the state-of-the-art work [8] by 110.6$\times$ in communication and 28.7$\times$ in computation even with 5$\times$ larger parameter vectors. More specifically, we make the following contributions:

- *(Secure) verifiable aggregation.* To verify the integrity of aggregation, we combine the linearly homomorphic hash with the commitment scheme to force the aggregation server to use the submitted gradients that are consistent with their previously broadcast hashes. Collision resistance of the hash scheme guarantees that the server cannot have an honest client accept its forged result. In addition, to overcome the aforementioned security issue, we observe that it sufficient to use equivocal commitment scheme to achieve the same security as [8] without using heavy zero-knowledge proof. Based on this observation, we develop a novel verifiable aggregation protocol. Notably, secure aggregation is also considered in our protocol and we show our protocol is composable with the secure protocol in [7] by using double-masking technique and adding extra rounds. The functionality of secure aggregation in our protocol is achieved in the sense that only the aggregation result computed by the server is revealed.

- *Dimension-independent communication overhead.* In our protocol, the communication overhead resulted from integrity verification is independent of the dimension of gradient vectors (i.e., the number of parameters in FL models), which significantly saves the bandwidth of clients. This is achieved by asking each client to commit only the hash of its gradient vector instead of the vector itself. These commitment strings, which facilitate our security proof, have a length independent of the gradient dimension. The linear homomorphism of the hash scheme ensures that the "sum" of hashes equals to the hash of the sum vector. It is helpful to think that linearly homomorphic hash can be used to compress high-dimensional gradient vectors while preserves the properties of addition.
- *Approximately-halved computation overhead.* We succeed in reducing the verification cost of our basic construction in terms of computation by using amortized verification. In our protocol, the most time-consuming operation is the two calls of linearly homomorphic hash that requires $O(d)$ expensive modular exponentiations, in which d is the dimension of gradient vectors. The former call is used to generate the necessary messages for future verification and is inevitable. However, the latter call that performs the integrity verification can be amortized. First, we draw a set of random coefficients to compute the linear combination of the hash aggregations in different epochs. Then, we check whether the combined hash is equal to the hash of the linear combination (using the same coefficients) of aggregation results in different epochs. In this way, we only performs the integrity verification once a batch and the cost of the latter call is amortized by a factor of the batch size. The total computation overhead can be approximately reduced to that of the former hash call.

II. Federated Learning With Secure Aggregation

A. Federated Learning

The basic framework of FL is summarized as follows. There are N users in FL, each having access to its private dataset $\mathcal{D}_i$ where $|\mathcal{D}_i| = s_i$. At the beginning of each epoch $k \in \{1, 2, \ldots\}$, the aggregation server will randomly select a subset of users S_k and send them the parameter vector $\mathbf{v}\langle k-1\rangle$ obtained in the previous epoch (note that $\mathbf{v}\langle 0\rangle$ is initialized randomly). Each selected user $\mathcal{P}_i$ will locally minimize the empirical loss over its dataset $\mathcal{D}_i$ to get its updated (local) model $\mathbf{v}_i\langle k\rangle$ and upload its gradient $\Delta_i\langle k\rangle \leftarrow \mathbf{v}_i\langle k\rangle - \mathbf{v}_i\langle k-1\rangle$ to the aggregation server. Upon receiving enough gradients from the users, typically, the server will take a weighted average of these gradients and obtain an updated global model, i.e., $\mathbf{v}\langle k\rangle \leftarrow \mathbf{v}\langle k-1\rangle + \sum_{i\in S_k} \beta_i \Delta_i\langle k\rangle$ where $\beta_i = s_i / \sum_{i\in S_k} s_i$. The training process will continue until model convergence.

B. Secure Aggregation

In [7], the authors build up a secure and dropout-tolerant aggregation protocol based on double-masking technique. More specifically, the double-masking of a gradient vector $\mathbf{v}_i$ includes two parts: the self-mask generated by the owner $\mathcal{P}_i$ itself, and the pairwise-mask generated between $\mathcal{P}_i$ and each other client. The double-masked gradient of $\mathbf{v}_i$ is denoted by $\mathbf{p}_i$:

$$\mathbf{p}_i = \mathbf{v}_i + \underbrace{\mathsf{PRG}(b_i)}_{\text{self}-\text{mask}} + \underbrace{\sum_{j\in\mathcal{U}, i\ <j} \mathsf{PRG}(\mathsf{mak}_{i,j}) - \sum_{j\in\mathcal{U}, i\ >j} \mathsf{PRG}(\mathsf{mak}_{i,j})}_{\text{pairwise}-\text{mask}} \ \mathbf{mod}\ B$$

where B is the modulus for aggregation, b_i is a secret seed sampled by the $\mathcal{P}_i$ and $\mathsf{mak}_{i,j} = \mathsf{mak}_{j,i}$ is a pairwise agreed value between $\mathcal{P}_i$ and $\mathcal{P}_j$ for each $\mathcal{P}_j \in \mathcal{U}$. Rounds of their protocol are briefly summarized in Figure 2.

Secure Aggregation Protocol

- **Round 0 (AdvertiseKeys):** Each client $\mathcal{P}_i$ generates two key pairs $(\mathsf{sk}_i, \mathsf{pk}_i)$, $(\mathsf{msk}_i, \mathsf{mpk}_i)$ and sends $(\mathsf{pk}_i, \mathsf{mpk}_i)$ to other clients. The server serves as a relay and appends each client that is alive at present to the set $\mathcal{U}_1$.
- **Round 1 (ShareKeys):** Each client $\mathcal{P}_i$ samples a secret value b_i and generates secret shares of (msk_i, b_i) to be distributed among other clients. For each other client $\mathcal{P}_j$, it derives pairwise symmetric encryption key $\mathsf{k}_{i,j}$ from sk_i and pk_j using key agreement. After that, it encrypts the shares for each $\mathcal{P}_j$ using $\mathsf{k}_{i,j}$ and sends the ciphertext to $\mathcal{P}_j$. The server serves as a relay and appends each client that is alive at present to the set $\mathcal{U}_2$.
- **Round 2 (MaskedInputCollection):** Each client $\mathcal{P}_i$ computes its double-masked gradient vector $\mathbf{p}_i$, in which the pairwise agreed $\mathsf{mak}_{i,j}$ is derived from msk_i and mpk_j using key agreement, and sends $\mathbf{p}_i$ to the server. The server appends each client that is alive at present to the set $\mathcal{U}_3$.
- **Round 3 (Unmasking):** The server asks the clients in $\mathcal{U}_3$ for: (i) all secret shares of b_i's for $i \in \mathcal{U}_3$, and (ii) all secret shares of msk_i for $i \in \mathcal{U}_2 \setminus \mathcal{U}_3$. The *aggregation result* $\mathbf{a} = \sum_{i\in\mathcal{U}_3} \mathbf{p}_i - \sum_{i\in\mathcal{U}_3} \mathsf{PRG}(b_i) + \sum_{i\in\mathcal{U}_2\setminus\mathcal{U}_3, j\in\mathcal{U}_3} \Delta_{i,j}\mathsf{PRG}(\mathsf{mak}_{i,j})$, in which b_i is directly derived from share reconstruction, $\mathsf{mak}_{i,j}$ is derived from the reconstructed msk_i and the public mpk_j, and $\Delta_{i,j} = 1$ if $i < j$ and $\Delta_{i,j} = -1$ if $i > j$ and $\Delta_{i,j} = 0$ if $i = j$.

Fig. 2. The semi-honest version protocol in [7].

C. Adversarial Model

In this paper, we consider the same adversary as in [8]. The adversary is semi-honest but with the additional power to instruct the corrupted server to forge the aggregation result arbitrarily with the knowledge of the transcripts it has seen before. The adversary can corrupt both the aggregation server and a subset of data parties (i.e., clients). In this semi-honest model, the corrupted parties will provide their gradients honestly. Using the notations to be introduced in the next section, all gradient vectors of parties belong to $\mathbb{Z}_R^d$ and the aggregation result should lie in $\mathbb{Z}_B^d$ for some bound $B \geq N \cdot R$, in which N is the number of parties.

In this adversarial model, the adversary may aim either (i) to infer the private gradients of honest parties, or (ii) to convince honest parties of its forged aggregation results. The goal of our protocol is to protect the privacy of each party's gradient while guarantees the integrity of aggregation. However, we do not consider the adversary that makes queries to the trained model to launch black-box statistical attacks [22]–[26] since it is known hard to prevent the leakage from the output of the functionality implemented by cryptographic protocols. Moreover, such attacks might not work well to precisely infer the sensitive information of honest parties, especially for deep neural networks that generalize well [6].

Note that, in FL, the private inputs of honest parties in the previous round may be approximate to the those in the current round when the model converges. To forge an *approximate* aggregation result, it suffices for the adversary to choose the inputs of corrupted parties according to the partial sum of the inputs of honest parties in the previous round. This observation can lead to a model replacement attack in [9], which bypasses secure aggregation protocols and is possible since some small perturbations will not significantly affect the behaviour of machine learning models. How to prevent such an attack in FL settings is another interesting and open problem, which is out of the scope of this paper. In this paper, like other works in the literature of multiparty computation, we only focus on how to prevent the adversary from forging an aggregation result different from the *exact* one. We do not consider the attacks (e.g., model replacement attack) where the adversary chooses its input according to the past protocol transcripts. This is known challenging in multiparty computation since it cannot be distinguished from the case where a corrupted party do use this as its input without some *apriori* knowledge of input.

III. VERIFL: EFFICIENT VERIFIABLE AGGREGATION

A. Notations

We use $[n]$ to denote the set $\{1, \ldots, n\}$ for some integer n. The set of integers is denoted by $\mathbb{Z}$. The quotient ring of integers modulo a positive integer q is written in $\mathbb{Z}_q$. When q is a prime, $\mathbb{Z}_q$ is a field denoted by $\mathbb{F}_q$. A vector is denoted by a bold lower case letter, e.g., $\mathbf{x}$, and its i-th entry is denoted by $\mathbf{x}[i]$ where the index i is counted from one. For some finite set $\mathcal{X}$, its cardinality is $|\mathcal{X}|$. If $\mathcal{A}$ is an algorithm, $a \leftarrow \mathcal{A}$ says that a is assigned to be the output of $\mathcal{A}$; otherwise, if $\mathcal{A}$ is a set, a is an element uniformly drawn from the set $\mathcal{A}$. The security parameter is denoted κ and we use $\mathsf{negl}(\kappa)$ to denote the negligible function in κ. We use $\mathcal{A}(x; y)$ to denote that an algorithm $\mathcal{A}$ is invoked with a public input x and a secret input y. We formulate pseudo-random generator as $\mathsf{PRG} : \{0, 1\}^* \mapsto \mathbb{Z}_B^d$ for some modulus B and vector dimension d.

B. Cryptographic Primitives

1) Linear Homomorphic Hash: A linearly homomorphic hash scheme consists of three polynomial-time algorithms, i.e., $\mathsf{LHH} = (\mathsf{LHH.HGen}, \mathsf{LHH.Hash}, \mathsf{LHH.Eval})$. We detail the above three algorithms as constructed in [27] by assuming the hardness of discrete logarithm.

- $\mathsf{LHH.HGen}(1^\kappa, 1^d)$: On input the security parameter κ and the dimension d, this algorithm outputs the public parameter LHHpp, including the description of a cyclic group $\mathbb{G}$ of *prime* order q, its generator $g \in \mathbb{G}$ and d distinct elements $g_1, \ldots, g_d \in \mathbb{G}$. For simplicity of presentation, this public parameter will be taken implicitly as the first parameter of $\mathsf{LHH.Hash}$ and $\mathsf{LHH.Eval}$.
- $\mathsf{LHH.Hash}(\mathbf{x})$: Taking a d-dimensional vector $\mathbf{x}$ as input, this algorithm outputs the linearly homomorphic hash of $\mathbf{x}$: $h \leftarrow \prod_{i\in[d]} g_i^{\mathbf{x}[i]} \in \mathbb{G}$.
- $\mathsf{LHH.Eval}(h_1, \ldots, h_\ell, \alpha_1, \ldots, \alpha_\ell)$: Taking ℓ hashes and ℓ coefficients of linear combination, this algorithm outputs the linear combination of these ℓ hashes: $h^* \leftarrow \prod_{i\in[\ell]} h_i^{\alpha_i}$.

Note that this construction of LHH satisfies the following definition of collision resistance with respect to the collision experiment $\mathsf{Expt}^{coll}_{\mathcal{A},\mathsf{LHH}}$.

$\mathsf{Expt}^{coll}_{\mathcal{A},\mathsf{LHH}}(1^\kappa, 1^d)$:
1) Call $\mathsf{LHHpp} \leftarrow \mathsf{LHH.HGen}(1^\kappa, 1^d)$.
2) Send LHHpp to the adversary and wait for its input $(\mathbf{x}_1, \mathbf{x}_2) \leftarrow \mathcal{A}(\mathsf{LHHpp})$ where $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{F}_q^d$ are two distinct vectors.
3) Output 1 iff $\mathsf{LHH.Hash}(\mathbf{x}_1) = \mathsf{LHH.Hash}(\mathbf{x}_2)$; otherwise output 0.

Definition 1 (Collision Resistance): LHH is said to be collision-resistant, if for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the advantage of $\mathcal{A}$

$$\mathsf{Adv}^{coll}_{\mathcal{A},\mathsf{LHH}}(\kappa) := \Pr\left[\mathsf{Expt}^{coll}_{\mathcal{A},\mathsf{LHH}}(1^\kappa, 1^d) = 1\right] \leq \mathsf{negl}(\kappa)$$

for the security parameter κ and the vector dimension d.

2) Commitment: Commitment is an "envelope" so that a party cannot change the value after they have committed to it while the committed value is kept secret to others before decommitment. The commitment scheme used in this paper is to make the security go through without leading to too much communication overhead for verification. In particular, to formally prove the security of our protocol, we need an *equivocal* commitment scheme. Roughly speaking, someone possessing the trapdoor of an equivocal commitment scheme can produce commitments that can be opened to different values. However, committers in real world, who have no idea about the trapdoor, can open them to only a single value. Such property enables the simulator in our proof to fool the distinguisher and achieves indistinguishability consequently (see Section IV).

A *non-interactive* equivocal commitment scheme is defined as a tuple of four polynomial-time algorithms $\mathsf{COM} = (\mathsf{COM.Setup}, \mathsf{COM.Commit}, \mathsf{COM.Decommit}, \mathsf{COM.Equiv})$:

- $\mathsf{COM.Setup}(1^\kappa)$: On input the security parameter κ, this algorithm outputs a public parameter COMpp and a trapdoor td. Note that the message space $\mathcal{M}$ and the commitment space $\mathcal{C}$ are also provided in COMpp. For simplicity of presentation, this public parameter will be taken implicitly as the first parameter of the other algorithms.
- $\mathsf{COM.Commit}(m; r)$: This algorithm is run by the committer and takes as input a message to be committed $m \in \mathcal{M}$ and a uniform randomness r and outputs a commitment string $c \in \mathcal{C}$, which is to be publicly

published. Note that the randomness r serves as the "decommitment string" to open the committed message and it should be kept secret until opening.

- $\mathsf{COM.Decommit}(c, m', r')$: This algorithm is run by the receiver and takes as input a commitment string $c \in \mathcal{C}$, a claimed committed message $m' \in \mathcal{M}$ and the claimed randomness r' that was used to commit m'. If $c = \mathsf{COM.Commit}(m'; r')$ then output 1; otherwise output 0.
- $\mathsf{COM.Equiv}(c, (m, r), m'; \mathsf{td})$: This algorithm takes as input a commitment string $c \leftarrow \mathsf{COM.Commit}(m; r) \in \mathcal{C}$, a desired arbitrary message $m' \in \mathcal{M}$ and the trapdoor td and outputs an valid randomness r' such that c decommits to m'.

The equivocality of COM is formally defined as follows.

Definition 2 (Equivocality): COM is said to be equivocal if for any $m \in \mathcal{M}$ and uniform randomness r, it holds that there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[b = 0 \;\middle|\; \begin{array}{l} (\mathsf{COMpp}, \mathsf{td}) \leftarrow \mathsf{COM.Setup}(1^\kappa), \\ c \leftarrow \mathsf{COM.Commit}(m; r), \\ m' \leftarrow \mathcal{M}, \\ r' \leftarrow \mathsf{COM.Equiv}(c, (m, r), m'; \mathsf{td}), \\ b \leftarrow \mathsf{COM.Decommit}(c, m', r') \end{array} \right] \leq \mathsf{negl}(\kappa)$$

for the security parameter κ.

3) Symmetric Encryption: An symmetric encryption scheme is defined to be a tuple $\mathsf{SE} = (\mathsf{SE.KeyGen}, \mathsf{SE.Enc}, \mathsf{SE.Dec})$.

- $\mathsf{SE.KeyGen}(1^\kappa)$: On input the security parameter κ, this algorithm outputs a secret symmetric key k.
- $\mathsf{SE.Enc}(m; \mathsf{k})$: This algorithm takes as input a message m to be encrypted and a symmetric key k. The output of this algorithm is the ciphertext c of m under the key k.
- $\mathsf{SE.Dec}(c; \mathsf{k}')$: This algorithm takes as input a ciphertext to be decrypted and its corresponding symmetric key k'. The output of this algorithm is the message m' such that $c \leftarrow \mathsf{SE.Enc}(m'; \mathsf{k}')$.

In this paper, we require SE to be IND-CPA secure.

4) Key Agreement: We use a key agreement scheme to (i) generate pairwise symmetric encryption keys, and (ii) generate pariwise seeds for PRG. A key agreement scheme is defined to be a tuple $\mathsf{KA} = (\mathsf{KA.Setup}, \mathsf{KA.KeyGen}, \mathsf{KA.Agree})$.

- $\mathsf{KA.Setup}(1^\kappa)$: On input the security parameter κ, this algorithm outputs a public parameter KApp. For simplicity of presentation, this public parameter will be taken implicitly as the first parameter of the other algorithms.
- $\mathsf{KA.KeyGen}()$: This algorithm generates a key pair $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{KA.Agree}(\mathsf{sk}_i, \mathsf{pk}_j)$: This algorithm takes as input a secret key sk_i and a public key pk_j and outputs a private agreed key $\mathsf{ak}_{i,j}$.

5) Secret Sharing: We use secret sharing to deal with dropout in our protocol and preserve input privacy. A secret sharing scheme is defined to be a tuple $\mathsf{SS} = (\mathsf{SS.Setup}, \mathsf{SS.Share}, \mathsf{SS.Combine})$.

- $\mathsf{SS.Setup}(1^\kappa)$: On input the security parameter κ, this algorithm outputs a public parameter SSpp, which includes the message space $\mathcal{M}$. For simplicity of presentation, this public parameter will be taken implicitly as the first parameter of the other algorithms.
- $\mathsf{SS.Share}(t, \mathcal{P}, s)$: This algorithms takes as input the threshold value t, the set of parties $\mathcal{P}$ which is of size $N \geq t$ and a secret $s \in \mathcal{M}$. The output of this algorithm is a set of secret shares, denoted by $\{[\![s]\!]_i\}_{\mathcal{P}_i \in \mathcal{P}}$, each of which is assigned to a distinct holder $\mathcal{P}_i \in \mathcal{P}$.
- $\mathsf{SS.Combine}(t, \{[\![s]\!]_i\}_{\mathcal{P}_i \in \mathcal{P}' \subseteq \mathcal{P}})$: This algorithm takes as input the threshold value t and the subset of shares $\{[\![s]\!]_i\}_{\mathcal{P}_i \in \mathcal{P}' \subseteq \mathcal{P}}$ of which the size is not less than t. The output of this algorithm is the original secret s.

C. How to Achieve Efficient Verifiable Aggregation?

1) Basic Verifiable Aggregation: The integrity of aggregation is achieved by adding the following two steps to the basic FL framework:

PREPARATION: This step is done by clients before they submit their gradient vectors to the server. In this step, each client is asked to generate: (i) the linearly homomorphic hash of its gradient vector, and (ii) the commitment string of this hash value. More formally, $\mathcal{P}_i$ generates

$$\begin{aligned} h_i &\leftarrow \mathsf{LHH.Hash}(\mathbf{v}_i), \\ c_i &\leftarrow \mathsf{COM.Commit}(h_i; r_i), \end{aligned}$$

in which h_i is the linearly homomorphic hash of $\mathbf{v}_i$, c_i is the commitment string and r_i is a uniformly random string secretly sampled by $\mathcal{P}_i$. Note that (h_i, r_i) serves as the *decommitment string* of $\mathcal{P}_i$. Note that, after this step, the commitment string c_i will be forwarded to other clients. $\mathcal{P}_i$ will not send its gradient vector to the server until it receives from all other clients their c_i's respectively.

VERIFICATION: This step is done by each client after it receives from the server the aggregation result $\mathbf{a}$. In this step, $\mathcal{P}_i$ asks each other client $\mathcal{P}_j$ for its decommitment string (h_j, r_j) and checks whether

$$1 \overset{?}{=} \mathsf{COM.Decommit}(c_j, h_j, r_j), \tag{1}$$

in which c_i was received after PREPARATION step. If the equality test (1) does not hold for some $j \in [N] \setminus \{i\}$, then the aggregation result $\mathbf{a}$ will be regarded as forged and $\mathcal{P}_i$ terminates with output $\perp$. Otherwise $\mathcal{P}_i$ will continue to check the equality of hashes:

$$\mathsf{LHH.Hash}(\mathbf{a}) \overset{?}{=} \mathsf{LHH.Eval}(h_1, \ldots, h_N, \underbrace{1, \ldots, 1}_{\text{N items}}). \tag{2}$$

If the equality test (2) holds, then the aggregation result $\mathbf{a}$ passes the verification and $\mathcal{P}_i$ will accept the result; otherwise the result will be regarded as forged and $\mathcal{P}_i$ terminates with output $\perp$.

2) Dimension-Independent Communication Overhead: The reduction in the communication overhead for verification comes from the usage of hash and commitment. Recall that the outgoing message for verification sent by each client $\mathcal{P}_i$

in each epoch consists of the commitment string c_i and the decommitment string (h_i, r_i). As long as we carefully instantiate COM with the one using commitment/decommitment strings of constant length, the outgoing communication for verification is constant and *independent* of the the dimension of gradient vectors. That is, in each epoch, the outgoing communication cost lead by verification can be made only $O(1)$ and the incoming communication cost is $O(N)$.

3) Approximately-Halved Computation Overhead: In our protocol, the most time-consuming operation for verification is running $\mathsf{LHH.Hash}$. To reduce the computation overhead for verification, we allow each client to do verification in an amortized manner. Similar to [14], [15], [28], we draw a set of random coefficients to compute the linear combination of the hash aggregations in different epochs. Then, we check whether the combined hash is equal to the hash of the linear combination (using the same coefficients) of aggregation results in different epochs. More formally, letting ℓ be the preset batch size and $\mathbf{a}\langle k\rangle$ be the aggregation result in the epoch $k \in [\ell]$, we replace the aforementioned VERIFICATION step with the following AMORTIZED-VERIFICATION step:

AMORTIZED-VERIFICATION: The equality test of commitment (1) remains unchanged for all commitments received in each epoch $k \in [\ell]$. However, for $k \in [\ell]$, $\mathcal{P}_i$ does the following:

$$\alpha_k \leftarrow \mathbb{F}_q,$$
$$h\langle k\rangle \leftarrow \mathsf{LHH.Eval}(h_1\langle k\rangle, \ldots, h_N\langle k\rangle, \underbrace{1, \ldots, 1}_{\text{N items}}),$$

in which q is the prime order provided in LHHpp and $h_i\langle k\rangle$ is the hash computed by client $\mathcal{P}_i$ in the k-th epoch. Then, it verifies the equality

$$\mathsf{LHH.Hash}\left(\sum_{k\in[\ell]} \alpha_k \mathbf{a}\langle k\rangle\right) \overset{?}{=} \mathsf{LHH.Eval}(h\langle 1\rangle, \ldots, h\langle \ell\rangle, \alpha_1, \ldots, \alpha_\ell) \quad (3)$$

instead of the equality (2). If the equality test (3) holds, then all the aggregation results $\mathbf{a}\langle 1\rangle, \ldots, \mathbf{a}\langle \ell\rangle$ pass the verification and $\mathcal{P}_i$ will accept these results; otherwise these results will be regarded as forged and $\mathcal{P}_i$ terminates with output $\perp$.

In this way, $\mathcal{P}_i$ does not need to call $\mathsf{LHH.Hash}$ in the equality test (2) every aggregation epoch. That is, the cost for $\mathsf{LHH.Hash}$ in the equality test (2) can be amortized by a factor ℓ. Since the cost for $\mathsf{LHH.Hash}$ is $O(d)$, the amortized computation cost for verification in each epoch can be reduced from $O(2d)$ to $O(d + d/\ell)$. When ℓ is large enough, the cost can be approximately halved.

4) Privacy Concern and Dropout-Tolerance: Our verifiable aggregation protocol (i.e., the basic FL framework with PREPARATION and AMORTIZED-VERIFICATION steps) is composable with the secure one Figure 2. The summarized secure verifiable aggregation protocol is given in Figure 3. We modify the $\mathsf{ShareKeys}$ round in [7] and the original secure protocol serves as the aggregation phase in this composite protocol. Notably, the dropout-tolerance of [7] is preserved and therefore the aggregation phase in our protocol is robust to client dropout. However, to deal with the potential dropout when clients report their decommitment strings in $\mathsf{Decommitting}$ round, an extra $\mathsf{DroppedDecommitting}$ round is introduced. This extra round guarantees that all clients that survive dropout are able to run AMORTIZED-VERIFICATION in $\mathsf{BatchChecking}$ round. Note that, to tolerate dropout, we require the additional $O(N)$ computation of secret sharing in $\mathsf{ShareMetadata}$ round. Therefore, the total verification cost in computation is $O(N + d)$.

VERIFL protocol

1) **Aggregation phase (for the epoch $k \in [\ell]$):**
- **Round 0 (AdvertiseKeys):** Identical to **Round 0** in Figure 2 with additional setups of LHH and COM.
- **Round 1 (ShareMetadata):** Identical to **Round 1** in Figure 2, except that each client $\mathcal{P}_i$ (i) additionally runs the PREPARATION step to get its $c_i\langle k\rangle$ and $(h_i\langle k\rangle, r_i\langle k\rangle)$, (ii) shares and encrypts $(h_i\langle k\rangle, r_i\langle k\rangle, \mathsf{msk}_i\langle k\rangle, b_i\langle k\rangle)$ instead of $(\mathsf{msk}_i\langle k\rangle, b_i\langle k\rangle)$, and (iii) sends $c_i\langle k\rangle$ to each other client along with the ciphertext.
- **Round 2 (MaskedInputCollection):** Identical to **Round 2** in Figure 2.
- **Round 3 (Unmasking):** Identical to **Round 3** in Figure 2, except that the server explicitly sends the aggregation result $\mathbf{a}\langle k\rangle$ to each client.

2) **Verification phase (of batch size ℓ):**
- **Round 0 (Decommitting):** For each $k \in [\ell]$, each client $\mathcal{P}_i$ sends its decommitment strings $(h_i\langle k\rangle, r_i\langle k\rangle)$ to each other client using the server as a relay. The server appends each client that is alive at present to the set $\mathcal{V}_1$.
- **Round 1 (DroppedDecommitting):** For each $k \in [\ell]$, each client $\mathcal{P}_i$ sends its shares of $h_j\langle k\rangle$ and $r_j\langle k\rangle$ to the server, in which $j \in \mathcal{U}_3\langle k\rangle \setminus \mathcal{V}_1$. The server reconstructs these $(h_j\langle k\rangle, r_j\langle k\rangle)$'s and sends them to each alive client.
- **Round 2 (BatchChecking):** Each client $\mathcal{P}_i$ runs the AMORTIZED-VERIFICATION step.

Fig. 3. Our verifiable aggregation protocol with secure aggregation functionality.

D. Full Version of VERIFL Protocol

The full version of VERIFL Protocol is presented in Figure 4 and Figure 5, where cryptographic primitives are defined in Section III-B.

IV. SECURITY ANALYSIS

In this section, we will show that our protocol ensures the integrity of the aggregation results (Definition 3) and the privacy of individual input (Definition 4). Recall that our protocol runs with a set $\mathcal{P}$ of N parties and an aggregation server $\mathcal{S}$, in which the building blocks are instantiated with the security parameter κ and the verification is performed in batch of size ℓ. The threshold for dropout is t (i.e., at least t parties survive dropout) and the set of corrupted parties is $\mathcal{C} \subseteq \mathcal{P} \cap \{\mathcal{S}\}$ such that $|\mathcal{C} \setminus \{\mathcal{S}\}| < t$. $\mathcal{U}_*\langle k\rangle$ is the set $\mathcal{U}_*$ in the k-th aggregation phase. Given fixed N, t, κ, ℓ and $\mathcal{C}$, we define $M_{\mathcal{C}}$ as the polynomial time algorithm for the "next-message" function of corrupted parties in $\mathcal{C}$. That is, given a party identifier $c \in \mathcal{C}$, a round index i a transcript T_i that has been sent and received so far by all corrupted parties in $\mathcal{C}$, and the joint randomness $r_{\mathcal{C}}$ for the execution of corrupted

Protocol Π_{VeriFL}

1) **Common reference string:** All parties are given $\mathsf{LHHpp} \leftarrow \mathsf{LHH.HGen}(1^\kappa, 1^d)$, the COMpp output by $\mathsf{COM.Setup}(1^\kappa)$, $\mathsf{SSpp} \leftarrow \mathsf{SS.Setup}(1^\kappa)$, and $\mathsf{KApp} \leftarrow \mathsf{KA.Setup}(1^\kappa)$. The security parameter κ, the threshold t, the number of parties N, the input domain $\mathbb{Z}_R^d$, the aggregation domain $\mathbb{Z}_B^d$ and the batch size ℓ are implicit.
2) **Aggregation phase (for the epoch $k \in [\ell]$):**
   - **Round 0 (AdvertiseKeys):**
     For each party $\mathcal{P}_i$ in parallel:
     - Generate $(\mathsf{sk}_i\langle k\rangle, \mathsf{pk}_i\langle k\rangle) \leftarrow \mathsf{KA.KeyGen}()$, $(\mathsf{msk}_i\langle k\rangle, \mathsf{mpk}_i\langle k\rangle) \leftarrow \mathsf{KA.KeyGen}()$.
     - Send $(\mathsf{pk}_i\langle k\rangle, \mathsf{mpk}_i\langle k\rangle)$ to the server and move to the next round.

     Server:
     - Collect messages from at least t parties in the previous round; otherwise abort. Let $\mathcal{U}_1\langle k\rangle$ be this set of parties.
     - Multi-send to all parties in $\mathcal{U}_1\langle k\rangle$ the set $\{(j, \mathsf{pk}_j\langle k\rangle, \mathsf{mpk}_j\langle k\rangle)\}_{j\in\mathcal{U}_1\langle k\rangle}$ and move to the next round.
   - **Round 1 (ShareMetadata):**
     For each party $\mathcal{P}_i$ in parallel:
     - Receive the set $\{(j, \mathsf{pk}_j\langle k\rangle, \mathsf{mpk}_j\langle k\rangle)\}_{j\in\mathcal{U}_1\langle k\rangle}$ from the server. Assert that $|\mathcal{U}_1\langle k\rangle| \geq t$, that all the public keys are different.
     - Compute the linearly homomorphic hash of its input in the k-th epoch, say $\mathbf{v}_i\langle k\rangle$, as well as the commitment of this hash: $h_i\langle k\rangle \leftarrow \mathsf{LHH.Hash}(\mathbf{v}_i\langle k\rangle)$, $c_i\langle k\rangle \leftarrow \mathsf{COM.Commit}(h_i\langle k\rangle; r_i\langle k\rangle)$, in which $r_i\langle k\rangle$ is uniformly drawn from randomness space.
     - Generate t-out-of-$(|\mathcal{U}_1\langle k\rangle| - 1)$ shares of $h_i\langle k\rangle$: $\{[\![h_i\langle k\rangle]\!]_j\}_{j\in\mathcal{U}_1\langle k\rangle\setminus\{i\}} \leftarrow \mathsf{SS.Share}(t, \mathcal{U}_1\langle k\rangle \setminus \{i\}, h_i\langle k\rangle)$.
     - Generate t-out-of-$(|\mathcal{U}_1\langle k\rangle| - 1)$ shares of $r_i\langle k\rangle$: $\{[\![r_i\langle k\rangle]\!]_j\}_{j\in\mathcal{U}_1\langle k\rangle\setminus\{i\}} \leftarrow \mathsf{SS.Share}(t, \mathcal{U}_1\langle k\rangle \setminus \{i\}, r_i\langle k\rangle)$.
     - Sample a random element $b_i\langle k\rangle \leftarrow \mathbb{F}_p$, in which the field $\mathbb{F}_p$ is described in SSpp.
     - Generate t-out-of-$(|\mathcal{U}_1\langle k\rangle| - 1)$ shares of $b_i\langle k\rangle$: $\{[\![b_i\langle k\rangle]\!]_j\}_{j\in\mathcal{U}_1\langle k\rangle\setminus\{i\}} \leftarrow \mathsf{SS.Share}(t, \mathcal{U}_1\langle k\rangle \setminus \{i\}, b_i\langle k\rangle)$.
     - Generate t-out-of-$(|\mathcal{U}_1\langle k\rangle| - 1)$ shares of $\mathsf{msk}_i\langle k\rangle$: $\{[\![\mathsf{msk}_i\langle k\rangle]\!]_j\}_{j\in\mathcal{U}_1\langle k\rangle\setminus\{i\}} \leftarrow \mathsf{SS.Share}(t, \mathcal{U}_1\langle k\rangle \setminus \{i\}, \mathsf{msk}_i\langle k\rangle)$.
     - For each other $\mathcal{P}_j \in \mathcal{U}_1\langle k\rangle \setminus \{i\}$, compute the pairwise symmetric key $\mathsf{k}_{i,j}\langle k\rangle \leftarrow \mathsf{KA.Agree}(\mathsf{sk}_i\langle k\rangle, \mathsf{pk}_j\langle k\rangle)$, and the ciphertext $C_{i,j}\langle k\rangle \leftarrow \mathsf{SE.Enc}((i, j, [\![h_i\langle k\rangle]\!]_j, [\![r_i\langle k\rangle]\!]_j, [\![b_i\langle k\rangle]\!]_j, [\![\mathsf{msk}_i\langle k\rangle]\!]_j); \mathsf{k}_{i,j}\langle k\rangle)$.
     - If any of the above operations fails, abort; otherwise send all tuples $(i, c_i\langle k\rangle, \{(j, C_{i,j}\langle k\rangle)\}_{j\in\mathcal{U}_1\langle k\rangle\setminus\{i\}})$ to the server.
     - Store all messages received and values generated in this round, and move to the next round at the end of the message delivery.

     Server:
     - Collect sets of ciphertexts from at least t parties. Let $\mathcal{U}_2\langle k\rangle \subseteq \mathcal{U}_1\langle k\rangle$ be this set of parties. Send to each party $\mathcal{P}_j \in \mathcal{U}_2\langle k\rangle$ all message for it: $\{(i, c_i\langle k\rangle, C_{i,j}\langle k\rangle)\}_{i\in\mathcal{U}_2\langle k\rangle\setminus\{j\}}$ and move to the next round.
   - **Round 2 (MaskedInputCollection):**
     For each party $\mathcal{P}_i$ in parallel:
     - Receive from the server the set of messages $\{(j, c_j\langle k\rangle, C_{j,i}\langle k\rangle)\}_{j\in\mathcal{U}_2\langle k\rangle\setminus\{i\}}$ and infer the set $\mathcal{U}_2\langle k\rangle$. If the set is of size $< t$, abort.
     - For each other party $\mathcal{P}_j \in \mathcal{U}_2\langle k\rangle \setminus \{i\}$, compute the agreed key for pairwise masking $\mathsf{mak}_{i,j}\langle k\rangle = \mathsf{KA.Agree}(\mathsf{msk}_i\langle k\rangle, \mathsf{mpk}_j\langle k\rangle)$.
     - Compute its masked input $\mathbf{p}_i\langle k\rangle \leftarrow \mathbf{v}_i\langle k\rangle + \mathsf{PRG}(b_i\langle k\rangle) + \sum_{j\in\mathcal{U}_2\langle k\rangle} \Delta_{i,j}\mathsf{PRG}(\mathsf{mak}_{i,j}\langle k\rangle)$ **mod** B, in which $\Delta_{i,j} = 1$ if $i < j$ and $\Delta_{i,j} = -1$ if $i > j$ and $\Delta_{i,j} = 0$ if $i = j$.
     - If any of the above operations fails, abort; otherwise send $\mathbf{p}_i\langle k\rangle$ to the server and move to the next round.

     Server:
     - Collect $\mathbf{p}_i\langle k\rangle$ from at least t parties. Let $\mathcal{U}_3\langle k\rangle \subseteq \mathcal{U}_2\langle k\rangle$ be this set of parties. Send to parties in $\mathcal{U}_3\langle k\rangle$ the set $\mathcal{U}_3\langle k\rangle$.
   - **Round 3 (Unmasking):**
     For each party $\mathcal{P}_i$ in parallel:
     - Receive from the server a set $\mathcal{U}_3\langle k\rangle$. Verify that $\mathcal{U}_3\langle k\rangle \subseteq \mathcal{U}_2\langle k\rangle$, that $|\mathcal{U}_3\langle k\rangle| \geq t$. If any of the above operations fails, abort.
     - For each other party $\mathcal{P}_j \in \mathcal{U}_2\langle k\rangle \setminus \{i\}$, decrypt the ciphertext $C_{j,i}\langle k\rangle$ (abort if decryption fails), which was received in the **Round 2**, to get $(j', i', c_{j'}\langle k\rangle, [\![h_{j'}\langle k\rangle]\!]_i, [\![r_{j'}\langle k\rangle]\!]_i, [\![b_{j'}\langle k\rangle]\!]_i, [\![\mathsf{msk}_{j'}\langle k\rangle]\!]_i) \leftarrow \mathsf{SE.Dec}(C_{j,i}\langle k\rangle; \mathsf{k}_{i,j}\langle k\rangle)$. Assert $j' = j$ and $i' = i$.
     - Send its secret seed for self-mask and two sets of shares to the server: $(b_i\langle k\rangle, \{[\![b_j\langle k\rangle]\!]_i\}_{j\in\mathcal{U}_3\langle k\rangle}, \{[\![\mathsf{msk}_j\langle k\rangle]\!]_i\}_{j\in\mathcal{U}_2\langle k\rangle\setminus\mathcal{U}_3\langle k\rangle})$.

     Server:
     - Collect message from at least t parties. Let $\mathcal{U}_4\langle k\rangle \subseteq \mathcal{U}_3\langle k\rangle$ be this set of parties. If $|\mathcal{U}_4\langle k\rangle| < t$, abort.
     - For each $\mathcal{P}_i \in \mathcal{U}_3\langle k\rangle \setminus \mathcal{U}_4\langle k\rangle$, reconstruct $b_i\langle k\rangle \leftarrow \mathsf{SS.Combine}(t, \{[\![b_i\langle k\rangle]\!]_j\}_{j\in\mathcal{U}_4\langle k\rangle})$. For each $\mathcal{P}_i \in \mathcal{U}_2\langle k\rangle \setminus \mathcal{U}_3\langle k\rangle$, reconstruct $\mathsf{msk}_i\langle k\rangle \leftarrow \mathsf{SS.Combine}(t, \{[\![\mathsf{msk}_i\langle k\rangle]\!]_j\}_{j\in\mathcal{U}_4\langle k\rangle})$ and therefore the $\mathsf{mak}_{i,j}\langle k\rangle$ for all $j \in \mathcal{U}_3\langle k\rangle$.
     - Compute the aggregation result $\mathbf{a}\langle k\rangle \leftarrow \sum_{i\in\mathcal{U}_3\langle k\rangle} \mathbf{p}_i\langle k\rangle - \sum_{i\in\mathcal{U}_3\langle k\rangle} \mathsf{PRG}(b_i\langle k\rangle) + \sum_{i\in\mathcal{U}_2\langle k\rangle\setminus\mathcal{U}_3\langle k\rangle, j\in\mathcal{U}_3\langle k\rangle} \Delta_{i,j}\mathsf{PRG}(\mathsf{mak}_{i,j}\langle k\rangle)$ and send $(\mathbf{a}\langle k\rangle, \mathcal{U}_4\langle k\rangle)$ to all parties in $\mathcal{U}_4\langle k\rangle$.

Fig. 4. Aggregation phase for privacy-preserving verifiable aggregation protocol.

parties, $M_{\mathcal{C}}(c, i, T_i, r_{\mathcal{C}})$ outputs the message for the party c in the round i.

Note that some common reference strings (CRS) are required by this protocol. As we will see, this protocol is secure in the CRS-hybrid model [29]. More specifically, according to the CRS functionality, the simulator is allowed to learn the trapdoor of the underlying commitment scheme. Therefore, in the CRS-hybrid model, the simulator of our protocol can obtain this trapdoor by simulating the CRS functionality and then use the trapdoor to make the simulated view consistent with the real one.

Definition 3 gives out the integrity of aggregation considered in this paper. In this definition, we say that integrity is achieved if an adversary who wants to forge the aggregation result after all clients having committed their homomorphic hash values respectively can be detected with an overwhelming probability.

Definition 3 (Integrity of Aggregation): In the k-th epoch, let $\boldsymbol{v}_H\langle k\rangle$ be the partially aggregation result of the inputs of honest parties in $\mathcal{U}_3\langle k\rangle$, and $\boldsymbol{v}_i\langle k\rangle$ be the well-formed input of some corrupted party $\mathcal{P}_i \in \mathcal{U}_3\langle k\rangle \cap \mathcal{C}$ whose hash was computed and committed in $\mathsf{ShareMetadata}$*. We say that the integrity of aggregation in a verification batch of size ℓ holds, if an adversary can have honest parties accept its forged aggregation result in some epochs of this batch with a negligible probability, i.e.,*

$$\Pr\left[\mathcal{P}_i \text{ outputs } \perp \;\middle|\; \begin{array}{l} \mathcal{P}_i \in \mathcal{V}_2 \setminus \mathcal{C}, \text{ for some } k \in \mathcal{K} \subseteq [\ell], \\ \boldsymbol{a}\langle k\rangle \leftarrow \boldsymbol{v}_H\langle k\rangle + \sum_{i\in\mathcal{U}_3\langle k\rangle\cap\mathcal{C}} \boldsymbol{v}_i\langle k\rangle, \\ \boldsymbol{a}\langle k\rangle \in \mathbb{Z}_B^d, \\ \boldsymbol{a}'\langle k\rangle \leftarrow M_{\mathcal{C}}(\mathcal{S}, k, T_k, r_{\mathcal{C}}), \\ \boldsymbol{a}'\langle k\rangle \in \mathbb{Z}_B^d,\ \boldsymbol{a}'\langle k\rangle \neq \boldsymbol{a}\langle k\rangle \end{array}\right] \geq 1 - \mathsf{negl}(\kappa).$$

Protocol Π_{VeriFL} (cont.)

1) **Batch verification phase (of batch size ℓ):**

- **Round 0 (Decommitting):**
  For each party $\mathcal{P}_i$ in parallel:
  - Send $(\{h_i\langle k\rangle, r_i\langle k\rangle\}_{k\in[\ell]})$ to the server.

  Server:
  - Collect messages from at least t parties. Let $\mathcal{V}_1 \subseteq \mathcal{U}_4\langle \ell\rangle$ be this set of parties (note that $\mathcal{U}_4\langle\ell\rangle$ is the latest set of surviving parties in this batch of size ℓ). If $|\mathcal{V}_1| < t$, abort; otherwise send the set $\{(\{h_i\langle k\rangle, r_i\langle k\rangle\}_{k\in[\ell]})\}_{i\in\mathcal{V}_1}$ to all parties in $\mathcal{V}_1$.
- **Round 1 (DroppedDecommitting):**
  For each party $\mathcal{P}_i$ in parallel:
  - Receive from the server the set $\{(\{h_i\langle k\rangle, r_i\langle k\rangle\}_{k\in[\ell]})\}_{i\in\mathcal{V}_1}$ and infer the set $\mathcal{V}_1$. If $|\mathcal{V}_1| < t$, abort.
  - Send two sets of shares to the server: $\{[\![h_j\langle k\rangle]\!]_i\}_{k\in[\ell], j\in\mathcal{U}_3\langle k\rangle\setminus\mathcal{V}_1}$ and $\{[\![r_j\langle k\rangle]\!]_i\}_{k\in[\ell], j\in\mathcal{U}_3\langle k\rangle\setminus\mathcal{V}_1}$.

  Server:
  - Collect sets of shares from at least t parties. Let $\mathcal{V}_2$ be this set of parties. If $\mathcal{V}_2 < t$, abort.
  - Reconstruct $h_i\langle k\rangle \leftarrow \mathsf{SS.Combine}(t, \{[\![h_i\langle k\rangle]\!]_j\}_{j\in\mathcal{V}_2})$ and $r_i\langle k\rangle \leftarrow \mathsf{SS.Combine}(t, \{[\![r_i\langle k\rangle]\!]_j\}_{j\in\mathcal{V}_2})$ for each $k\in[\ell]$ and each $\mathcal{P}_i \in \mathcal{U}_3\langle k\rangle \setminus \mathcal{V}_1$.
  - Send $\{(h_i\langle k\rangle, r_i\langle k\rangle)\}_{k\in[\ell], i\in\mathcal{U}_3\langle k\rangle\setminus\mathcal{V}_1}$ to all parties in $\mathcal{V}_2$.
- **Round 2 (BatchChecking):**
  For each party $\mathcal{P}_i$ in parallel:
  - Receive $\{(h_i\langle k\rangle, r_i\langle k\rangle)\}_{k\in[\ell], i\in\mathcal{U}_3\langle k\rangle\setminus\mathcal{V}_1}$ from the server. If, for all $k\in[\ell]$, the set indeed contains the decommitment strings of parties in $\mathcal{U}_3\langle k\rangle\setminus\mathcal{V}_1$, continue; otherwise abort.
  - Check that $\mathsf{COM.Decommit}(c_j\langle k\rangle, h_j\langle k\rangle, r_j\langle k\rangle) = 1$ holds for all $k\in[\ell]$ and $j\in\mathcal{U}_3\langle k\rangle\setminus\{i\}$; otherwise output $\perp$.
  - For $k\in[\ell]$, combine the linearly homomorphic hashes of different parties in the k-th epoch: $h\langle k\rangle \leftarrow \mathsf{LHH.Eval}(\{h_j\langle k\rangle\}_{j\in\mathcal{U}_3\langle k\rangle}, 1, \ldots, 1)$.
  - Draw ℓ uniformly random coefficients in the field $\mathbb{F}_q$ (which is provided in LHHpp) $\alpha_1, \ldots, \alpha_\ell$ and compute the linear combination of combined hashes of different epochs: $h^* \leftarrow \mathsf{LHH.Eval}(h\langle 1\rangle, \ldots, h\langle\ell\rangle, \alpha_1, \ldots, \alpha_\ell)$.
  - Compute the linear combination of the aggregation results of different epochs using the same coefficients: $\mathbf{a}^* \leftarrow \sum_{k\in[\ell]} \alpha_k \mathbf{a}\langle k\rangle \ \mathbf{mod}\ B$.
  - Check that $h^* = \mathsf{LHH.Hash}(\mathbf{a}^*)$. Output $(\mathbf{a}\langle 1\rangle, \ldots, \mathbf{a}\langle\ell\rangle)$ if it holds; otherwise output $\perp$.

Fig. 5. Verification phase for privacy-preserving verifiable aggregation protocol.

By Definition 4, we define input privacy of each client in the existence of the considered adversary. This definition aims to capture an adversary who corrupts the server and a subset of parties learns nothing from protocol transcript but the partially aggregation result of honest parties. We say that an adversary learns nothing if its view can be simulated by a simulator without any secret internal state of honest parties (e.g., the knowledge of the gradient of a client). More specifically, input privacy is defined in the sense that the view of an adversary corrupting less than t data parties can be simulated only given the partially aggregation result of honest parties.

Definition 4 (Input Privacy): We say that the input privacy of an honest client holds, if there exists a PPT simulator SIM such that for any set of parties $\mathcal{P}$, threshold t, security parameter κ, batch size ℓ, set of inputs $\{\boldsymbol{v}_i\}_{i\in\mathcal{P}}$, $\{\mathcal{U}_1\langle k\rangle, \mathcal{U}_2\langle k\rangle, \mathcal{U}_3\langle k\rangle, \mathcal{U}_4\langle k\rangle\}_{k\in[\ell]}$ and $\mathcal{V}_1$, $\mathcal{V}_2$ such that for all $j \in [3]$ and $k\in[\ell-1]$

$$\begin{aligned}\mathcal{P} &\supseteq \mathcal{U}_1\langle 1\rangle,\\ \mathcal{U}_j\langle k\rangle &\supseteq \mathcal{U}_{j+1}\langle k\rangle,\\ \mathcal{U}_4\langle k\rangle &\supseteq \mathcal{U}_1\langle k+1\rangle,\\ \mathcal{U}_4\langle \ell\rangle &\supseteq \mathcal{V}_1 \supseteq \mathcal{V}_2,\end{aligned}$$

and set of corrupted parties $\mathcal{C}$ such that $\mathcal{C} \subseteq \mathcal{P} \cap \{\mathcal{S}\}$ and $|\mathcal{C}\setminus\{\mathcal{S}\}| < t$, the output of SIM is computationally indistinguishable from the output of $\mathsf{REAL}_{\mathcal{C}}^{\mathcal{P},t,\kappa}$:

$$\begin{aligned}&\mathsf{REAL}_{\mathcal{C}}^{\mathcal{P},t,\kappa}(\{\boldsymbol{v}_i\}_{i\in\mathcal{P}}, \{\mathcal{U}_1\langle k\rangle, \mathcal{U}_2\langle k\rangle, \mathcal{U}_3\langle k\rangle, \mathcal{U}_4\langle k\rangle\}_{k\in[\ell]}, \mathcal{V}_1, \mathcal{V}_2)\\ &\approx_c \mathsf{SIM}_{\mathcal{C}}^{\mathcal{P},t,\kappa}(\{\boldsymbol{v}_i\}_{i\in\mathcal{C}}, \{\boldsymbol{z}\langle k\rangle\}_{k\in[\ell]}, \{\mathcal{U}_1\langle k\rangle, \mathcal{U}_2\langle k\rangle, \mathcal{U}_3\langle k\rangle,\\ &\qquad \mathcal{U}_4\langle k\rangle\}_{k\in[\ell]}, \mathcal{V}_1, \mathcal{V}_2),\end{aligned}$$

where

$$\boldsymbol{z}\langle k\rangle = \begin{cases} \sum_{i\in\mathcal{U}_3\langle k\rangle\setminus\mathcal{C}} \boldsymbol{v}_i, & \text{if } |\mathcal{U}_3\langle k\rangle| \geq t\\ \perp, & \text{otherwise.}\end{cases}$$

Lemma 1: Assume the hardness of discrete logarithm and the security of COM. In the AMOTIZED-VERIFICATION *step, an honest client will accept the aggregation results $\boldsymbol{a}\langle 1\rangle, \ldots, \boldsymbol{a}\langle\ell\rangle$ if and only if these results are honestly aggregated by the server with an overwhelming probability.*

Proof: Assume there exists a PPT adversary that succeeds in letting some honest party $\mathcal{P}_i$ output its forged aggregation results $\bar{\mathbf{a}}\langle k\rangle \neq \mathbf{a}\langle k\rangle = \sum_i \mathbf{v}_i\langle k\rangle$ for all $k\in\mathcal{K}\subseteq[\ell]$ in $\mathsf{BatchChecking}$. We first observe that, since $\mathcal{P}_i$ does not output $\perp$, the decommitment in $\mathsf{BatchChecking}$ should be done successfully. For decommitment strings $(h_j\langle k\rangle, r_j\langle k\rangle)$ where $k\in\mathcal{K}$ and $j\in\mathcal{U}_3\langle k\rangle\cap\mathcal{C}$, the adversary has the freedom whether to send correct decommitment strings in $\mathsf{Decommitting}$. If it chooses to send malformed decommitment strings, the decommitment in $\mathsf{BatchChecking}$ fails with a non-negligible probability since COM is binding. The same argument applies to the case where the adversary sends to $\mathcal{P}_i$ its incorrectly reconstructed decommitment strings of (honest) dropped parties in $\mathsf{DroppedDecommitting}$.

The above argument implies that the adversary cannot change the hash values it has committed in $\mathsf{ShareMetadata}$ on behalf of corrupted parties without having $\mathcal{P}_i$ output $\perp$. Recall that, by protocol specification, the honestly aggregation result $\mathbf{a}\langle k\rangle$ and the final hash h^* satisfy

$$\begin{aligned}h^* &= \mathsf{LHH.Hash}(\mathbf{a}^*) = \mathsf{LHH.Hash}\left(\sum_{k\in[\ell]}\alpha_k\mathbf{a}\langle k\rangle\right)\\ &= \prod_{j\in[d]} g_j^{\sum_{k\in[\ell]}\alpha_k\mathbf{a}\langle k\rangle[j]}\\ &= \left(\prod_{j\in[d]} g_j^{\sum_{k\in\mathcal{K}}\alpha_k\mathbf{a}\langle k\rangle[j]}\right)\cdot\left(\prod_{j\in[d]} g_j^{\sum_{k\in[\ell]\setminus\mathcal{K}}\alpha_k\mathbf{a}\langle k\rangle[j]}\right)\end{aligned}$$

Meanwhile, with unchanged hashes committed in $\mathsf{ShareMetadata}$, we have another linear combination

$\bar{\mathbf{a}} \leftarrow \sum_{k\in\mathcal{K}} \alpha_k \bar{\mathbf{a}}\langle k\rangle + \sum_{k\in[\ell]\backslash\mathcal{K}} \alpha_k \mathbf{a}\langle k\rangle$ and therefore

$$\begin{aligned}&\mathsf{LHH.Hash}(\bar{\mathbf{a}})\\ &= \mathsf{LHH.Hash}\left(\sum_{k\in\mathcal{K}} \alpha_k \bar{\mathbf{a}}\langle k\rangle + \sum_{k\in[\ell]\backslash\mathcal{K}} \alpha_k \mathbf{a}\langle k\rangle\right)\\ &= \left(\prod_{j\in[d]} g_j^{\sum_{k\in\mathcal{K}} \alpha_k \bar{\mathbf{a}}\langle k\rangle[j]}\right)\cdot\left(\prod_{j\in[d]} g_j^{\sum_{k\in[\ell]\backslash\mathcal{K}} \alpha_k \mathbf{a}\langle k\rangle[j]}\right).\end{aligned}$$

Since $\mathcal{P}_i$ does not output $\perp$, it should hold that $h^* = \mathsf{LHH.Hash}(\bar{\mathbf{a}})$, i.e.,

$$\prod_{j\in[d]} g_j^{\sum_{k\in\mathcal{K}} \alpha_k \mathbf{a}\langle k\rangle[j]} = \prod_{j\in[d]} g_j^{\sum_{k\in\mathcal{K}} \alpha_k \bar{\mathbf{a}}\langle k\rangle[j]}.$$

where $\mathbf{a}\langle k\rangle \neq \bar{\mathbf{a}}\langle k\rangle$ for $k \in \mathcal{K}$. Notice that the coefficients α_k for $k \in \mathcal{K}$ are uniformly drawn from $\mathbb{F}_q$, the event that $h^* = \mathsf{LHH.Hash}(\bar{\mathbf{a}})$ and

$$\sum_{k\in\mathcal{K}} \mathbf{a}\langle k\rangle[j] = \sum_{k\in\mathcal{K}} \bar{\mathbf{a}}\langle k\rangle[j]$$

in which $j \in [d]$ and $\mathbf{a}\langle k\rangle \neq \bar{\mathbf{a}}\langle k\rangle$ for $k \in \mathcal{K}$ happens with a probability $1/|\mathbb{F}_q|$, which is negligible in the security parameter κ. In other words, conditioned on $h^* = \mathsf{LHH.Hash}(\bar{\mathbf{a}})$, it is hold that

$$\sum_{k\in\mathcal{K}} \mathbf{a}\langle k\rangle[j] \neq \sum_{k\in\mathcal{K}} \bar{\mathbf{a}}\langle k\rangle[j]$$

for some $j \in [d]$, which induces a collision of LHH.

It is readily seen that such a PPT adversary can break either the binding property of COM or the collision resistance (Definition 1) of LHH with a non-negligible probability, which is infeasible under the assumption. This completes the proof. □

Theorem 1: Assume the hardness of discrete logarithm and the security of COM. VERIFL *achieves integrity of aggregation according to Definition 3.*

Proof: This theorem is straightforward from Lemma 1. □

Theorem 2: Assume the security of SE, KA, SS and COM. VERIFL *achieves input privacy according to Definition 4 in the CRS-hybrid model.*

Proof: The proof is identical to that for Theorem 6.3 in [7], except that we should additionally take care of the hashes being committed in ShareMetadata. Notice that SIM did not know the real inputs of honest parties by the time it was asked to compute the hash as well as the commitment. The strategy of SIM is to hash a dummy vector (e.g., the one filled with 0's) and commit this dummy hash value. Give that COM is hiding and the secret sharing scheme hides messages being shared, the joint view w.r.t. the *aggregation phase* of parties in $\mathcal{C}$ will be indistinguishable from that in $\mathsf{REAL}_{\mathcal{C}}^{\mathcal{P},t,\kappa}$.

It remains to show that the joint view w.r.t. the *verification phase* of parties in $\mathcal{C}$ can also be simulated by SIM. Note that the simulated commitment is committed to the dummy hash which is different (with an overwhelming probability) to that of the vector sampled by SIM after it obtains the partially aggregation result of honest parties by querying $\mathbf{z}\langle k\rangle$ for $k \in [\ell]$. Recall that COM is equivocal and, in the CRS-hybrid model, SIM has the trapdoor td which is associated with the COMpp output by COM.Setup. So, using the trapdoor td, SIM can equivocate these commitments to the hashes of inputs sampled by it on behalf of honest parties. The equivocality of COM (Definition 2) guarantees that these commitments are consistent with hashes of the simulated inputs of honest parties and therefore can be successfully decommitted with an overwhelming probability. Moreover, since $\mathcal{C}$ forms an unqualified set to reconstruct shared secrets by assumption, SIM can adjust its shares held on behalf of honest parties such that the reconstructed decomittment strings match those obtained from equivocation. That is, the *verification phase* can be simulated, which completes the proof. □

V. EVALUATION

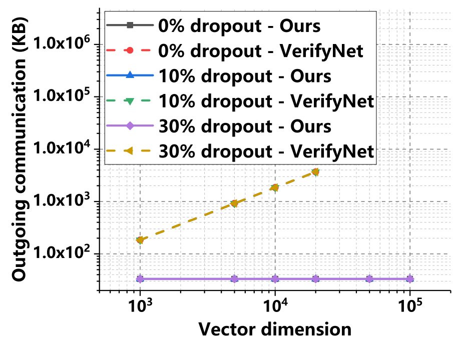
A. Experimental Setup

We show the performance of our protocol based on a prototype implementation. The prototype is written in Java while we use JNI to implement low-level cryptography algorithms. In particular, linearly homomorphic hash LHH is realized using elliptic curve over the NIST P-256 curve. For efficient equivocal commitment COM, we use folklore hash commitment scheme instantiated with SHA-256, which is proved secure in restricted programmable and observable global random oracle model [30]. For secret sharing scheme, we adopt standard t-out-of-N Shamir secret sharing. For key agreement, we use elliptic curve Diffie-Hellman over the NIST P-256 curve with SHA-384. The symmetric encryption is instantiated with AES-OFB mode with 256-bits key and 128-bits IV. For pseudo-random generator, we use AES-CTR mode. In addition, we fix two moduli $R = 2^{24}$ and $B = 2^{34}$ (i.e., the maximum number of clients is $B/R = 2^{10} = 1024$) and assume all clients will input honestly as in our adversarial model.

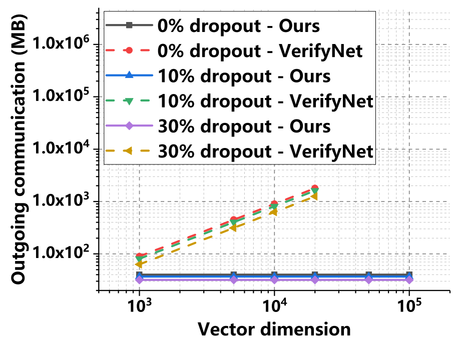
We simulate clients and the aggregation server on a 64-bits Ubuntu 16.04 LTS desktop equipped with Intel i7-7700 CPU (3.60 GHz) and 16 GB RAM. The simulation is single-threaded. Given that end-to-end networking time will not significantly influence the asymptotic computation complexity of our protocol and can be calculated using bandwidth, we omit this part in our evaluation.

1) Dropout Cases: For client dropout, we consider the following two cases of dropout in our protocol:

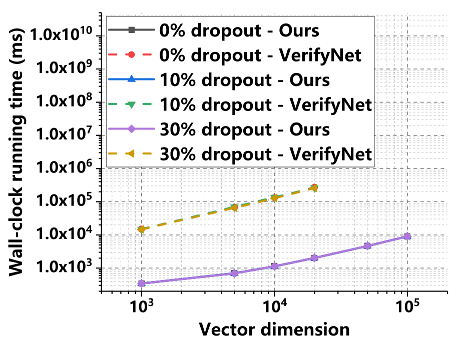
- Case I dropout: In this case, clients drop out of the protocol after sending their metadata to other clients via server in ShareMetadata but before sending their masked gradients to the server in MaskedInputCollection. As discussed in [7], this dropout case will lead to the worst performance of aggregation phase in our protocol for the expensive computation overhead to recover the pairwise-mask.
- Case II dropout: In this case, some clients drop out before reporting their decommitment strings to other clients via the aggregation server in Decommitting. The server has to recover these strings of dropped clients by asking all surviving clients to send their shares of these strings



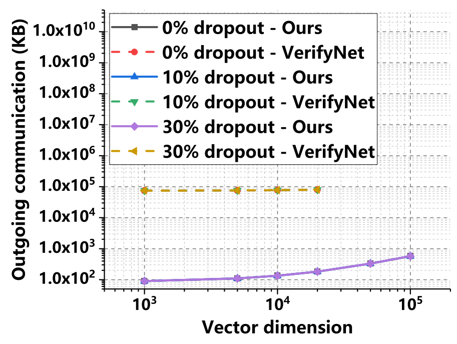
(a)



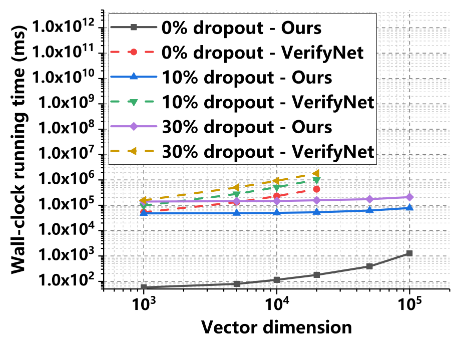
(b)



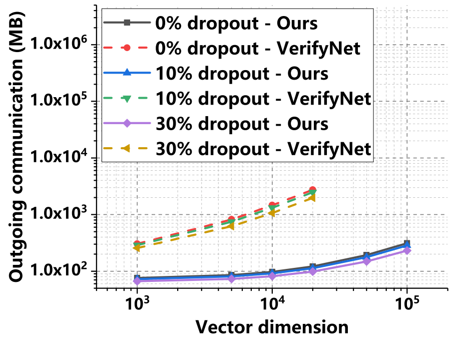
(c)



(d)



(e)



(f)

Fig. 6. Comparison between our protocol and VerifyNet [8] in terms of (i) outgoing communication overhead for verification, (ii) the total computation overhead, and (iii) the total outgoing communication overhead, as the dimension of gradient vectors increases. (a) Outgoing communication overhead for verification per client. (b) Outgoing communication overhead for verification of the server. (c) Computation overhead per client. (d) Outgoing communication overhead per client. (e) Computation overhead of the server. (f) Total outgoing communication overhead of the server.

and running secret reconstruction algorithm to recover them. This will lead to the most expensive computation overhead in our verification phase.

B. Comparison With VerifyNet [8]: Dimension-Independence

Given that the protocol in [8] does not allow Case II dropout and amortization, we consider only the Case I dropout and set the batch size to 1 in our comparison. In addition, we set the number of clients $N = 500$ and fix the threshold $t = \frac{1}{2}N$. All other parameters remain unchanged as above. It is readily seen from Figure 6(a) and Figure 6(b) that, in our protocol, the outgoing communication cost for verification of either each client or the server is independent of the dimension of gradients. However, in [8], the two metrics is linearly dependent on the dimension of gradients, which results in impractical performance when the dimension is large enough. A more comprehensive comparison with respect to the overall overhead is presented in Figure 6. It is easy to see that our protocol outperforms VerifyNet completely. A more encouraging result is that our protocol can be scaled to support high-dimensional gradient vectors (e.g., 100K) with even better performance than that achieved by VerifyNet in dealing with low-dimensional one.

C. Comparison With Secure Aggregation [7]

In Table I, Table II, Table III and Table IV, we give out the additional costs for verification in our protocol. The baseline protocol is [7] that addresses the input privacy of mobile clients. It is easy to see from underlined bold figures that our protocol does not introduce too much additional communication overhead to the baseline secure aggregation protocol per client. For example, when the dimension of gradient vectors is set to 100K, the communication for transferring a gradient is always 488.28 KB, which takes the largest proportion of the overall communication per client. Compared with that, the additional communication caused by verification protocol is insignificant. As for the computation overhead, the one caused by our protocol is also affordable compared with the one in [7]. In the real-world applications of FL, it is not uncommon that each local iteration of machine learning requires tens of seconds. When deploying our protocol to these applications where dropout is not severe (e.g., <1% occurred naturally as reported in [7]), the overall computation overhead will not blow up since our protocol has similar wall-clock running time with a local iteration of machine learning.

D. Other Experimental Results

1) Amortized Verification: In Figure 7, it is clear that, with the growth of batch size, the amortized verification overhead per epoch can be reduced nearly to that of a single LHH.Hash call. In other words, with a large enough batch size, we can approximately halve the computation overhead of a client in the sense of amortization. However, in real-world applications where there are dropouts in each epoch, the batch size cannot be set too large given that too large verification batch will lead

TABLE I

COMPUTATION OVERHEAD WITH RESPECT TO CASE I DROPOUT[1]

Num. Client	Dropout		Aggregation phase (Round #)					Verification phase (Round #)			Total
			0	1	2	3	Receive result	0	1	2	
1000	0.00%	Client	0 ms	391 + **3947** ms	1798 ms	0 ms	45 ms	**0** ms	**17** ms	**3931** ms	10129 ms
		Server	0 ms	0 ms	1007 ms	1363 ms	-	**1** ms	**0** ms	-	2371 ms
	10.00%	Client	0 ms	411 + **4089** ms	1834 ms	0 ms	45 ms	**0** ms	**15** ms	**3910** ms	10304 ms
		Server	0 ms	0 ms	930 ms	216205 ms	-	**0** ms	**0** ms	-	217135 ms
	30.00%	Client	0 ms	408 + **3930** ms	1791 ms	0 ms	45 ms	**0** ms	**12** ms	**3915** ms	10101 ms
		Server	1 ms	0 ms	703 ms	561420 ms	-	**0** ms	**0** ms	-	562124 ms
500	0.00%	Client	0 ms	128 + **3919** ms	896 ms	0 ms	44 ms	**0** ms	**8** ms	**3904** ms	8899 ms
		Server	0 ms	0 ms	501 ms	679 ms	-	**0** ms	**0** ms	-	1180 ms
	10.00%	Client	0 ms	123 + **3954** ms	909 ms	0 ms	45 ms	**0** ms	**7** ms	**3969** ms	9007 ms
		Server	1 ms	0 ms	466 ms	43921 ms	-	**0** ms	**0** ms	-	44388 ms
	30.00%	Client	0 ms	122 + **3943** ms	896 ms	0 ms	44 ms	**0** ms	**6** ms	**3906** ms	8917 ms
		Server	0 ms	0 ms	351 ms	107428 ms	-	**0** ms	**0** ms	-	107779 ms

[1]The dimension of gradient vectors is set to 100K and the batch size is 1. The underlined bold figures stand for the costs for verification in our protocol, compared with our implementation of [7] in the same experimental environment and parameter settings.

TABLE II

OUTGOING COMMUNICATION OVERHEAD WITH RESPECT TO CASE I DROPOUT[1]

Num. Client	Dropout		Aggregation phase (Round #)					Verification phase (Round #)			Total
			0	1	2	3	Receive result	0	1	2	
1000	0.00%	Client	0.07 KB	68.29 + **66.37** KB	488.28 KB	35.16 KB	-	**0.07** KB	**0.00** KB	-	658.24 KB
		Server	66.41 KB	68.29 + **97.56** KB	1.95 KB	488.28 KB	-	**66.41** KB	**0.00** KB	-	788.90 KB
	10.00%	Client	0.07 KB	68.29 + **66.37** KB	488.28 KB	35.16 KB	-	**0.07** KB	**0.00** KB	-	658.24 KB
		Server	66.41 KB	68.29 + **97.56** KB	1.76 KB	488.28 KB	-	**59.77** KB	**0.00** KB	-	782.06 KB
	30.00%	Client	0.07 KB	68.29 + **66.37** KB	488.28 KB	35.16 KB	-	**0.07** KB	**0.00** KB	-	658.24 KB
		Server	66.41 KB	68.29 + **97.56** KB	1.37 KB	488.28 KB	-	**46.48** KB	**0.00** KB	-	768.39 KB
500	0.00%	Client	0.07 KB	34.11 + **33.17** KB	488.28 KB	17.58 KB	-	**0.07** KB	**0.00** KB	-	573.28 KB
		Server	33.20 KB	34.11 + **48.73** KB	0.98 KB	488.28 KB	-	**33.20** KB	**0.00** KB	-	638.51 KB
	10.00%	Client	0.07 KB	34.11 + **33.17** KB	488.28 KB	17.58 KB	-	**0.07** KB	**0.00** KB	-	573.28 KB
		Server	33.20 KB	34.11 + **48.73** KB	0.88 KB	488.28 KB	-	**29.88** KB	**0.00** KB	-	635.09 KB
	30.00%	Client	0.07 KB	34.11 + **33.17** KB	488.28 KB	17.58 KB	-	**0.07** KB	**0.00** KB	-	573.28 KB
		Server	33.20 KB	34.11 + **48.73** KB	0.68 KB	488.28 KB	-	**23.24** KB	**0.00** KB	-	628.25 KB

[1]The dimension of gradient vectors is set to 100K and the batch size is 1. The entries of server are outgoing communication sent from the server to a single client. The underlined bold figures stand for the costs for verification in our protocol, compared with our implementation of [7] in the same experimental environment and parameter settings.

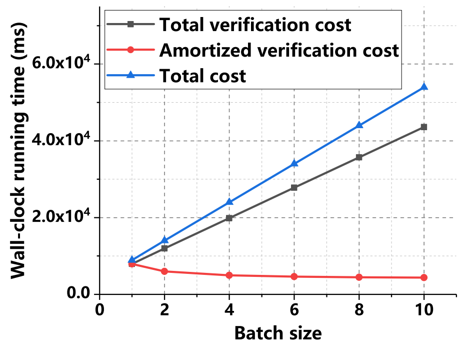


Fig. 7. Comparison between total computation cost for verification, amortized computation cost for verification and total computation cost per client. The dimension of gradient vectors is set to 100K and the number of clients is 500. Assume no dropout.

to severe accumulation of dropouts and therefore the expensive overhead to run secret reconstruction algorithm. We regard this phenomenon as a tradeoff and therefore the batch size should be fine-tuned in real-world applications.

2) Number of Clients: Both the computation and communication cost in client side grows linearly in the number of clients, which results from the fact that the number of secret shares generated in ShareMetadata is proportional to the number of clients. For the wall-clock running time of the server, it grows quadratically in the number of clients. The reason is that, in Unmasking, the server has to run secret reconstruction algorithm to recover the secret mask key of dropped clients and both the threshold t and the number of dropped clients are proportional to the number of clients N.

3) Dropout: As shown in Figure 8 and Figure 9, it is clear that, in client side, either the computation cost or the communication cost is independent of how many other clients drop out of the protocol execution. However, the two metrics in server side is influenced by the dropout rate, especially the wall-clock running time of the server. We note that the overhead suffered by the server is acceptable.

VI. RELATED WORK

In this section, we briefly discuss the works related to secure verifiable aggregation in federated learning.

TABLE III
COMPUTATION OVERHEAD WITH RESPECT TO CASE II DROPOUT[1]

Num. Client	Dropout		Aggregation phase (Round #)					Verification phase (Round #)			Total
			0	1	2	3	Receive result	0	1	2	
1000	0.00%	Client	0 ms	392 + **3946** ms	1798 ms	0 ms	45 ms	**0** ms	**17** ms	**3931** ms	10129 ms
		Server	0 ms	0 ms	1007 ms	1363 ms	-	**1** ms	**0** ms	-	2371 ms
	10.00%	Client	0 ms	405 + **3931** ms	1792 ms	0 ms	45 ms	**0** ms	**15** ms	**3927** ms	10115 ms
		Server	0 ms	0 ms	1004 ms	1358 ms	-	**0** ms	**172309** ms	-	174671 ms
	30.00%	Client	0 ms	408 + **3930** ms	1796 ms	0 ms	45 ms	**0** ms	**12** ms	**3925** ms	10116 ms
		Server	0 ms	0 ms	1011 ms	1369 ms	-	**0** ms	**517306** ms	-	519686 ms
500	0.00%	Client	0 ms	126 + **3921** ms	896 ms	0 ms	44 ms	**0** ms	**8** ms	**3904** ms	8899 ms
		Server	0 ms	0 ms	501 ms	679 ms	-	**0** ms	**0** ms	-	1180 ms
	10.00%	Client	0 ms	140 + **3917** ms	896 ms	0 ms	45 ms	**0** ms	**7** ms	**3912** ms	8917 ms
		Server	0 ms	0 ms	502 ms	680 ms	-	**0** ms	**21377** ms	-	22559 ms
	30.00%	Client	0 ms	136 + **3928** ms	899 ms	0 ms	45 ms	**0** ms	**6** ms	**3938** ms	8952 ms
		Server	1 ms	0 ms	507 ms	681 ms	-	**0** ms	**64462** ms	-	65651 ms

[1]The dimension of gradient vectors is set to 100K and the batch size is 1. The underlined bold figures stand for the costs for verification in our protocol, compared with our implementation of [7] in the same experimental environment and parameter settings.

TABLE IV
OUTGOING COMMUNICATION OVERHEAD WITH RESPECT TO CASE II DROPOUT[1]

Num. Client	Dropout		Aggregation phase (Round #)					Verification phase (Round #)			Total
			0	1	2	3	Receive result	0	1	2	
1000	0.00%	Client	0.07 KB	68.29 + **66.37** KB	488.28 KB	35.16 KB	-	**0.07** KB	**0.00** KB	-	658.24 KB
		Server	66.41 KB	68.29 + **97.56** KB	1.95 KB	488.28 KB	-	**66.41** KB	**0.00** KB	-	788.90 KB
	10.00%	Client	0.07 KB	68.29 + **66.37** KB	488.28 KB	35.16 KB	-	**0.07** KB	**6.84** KB	-	665.07 KB
		Server	66.41 KB	68.29 + **97.56** KB	1.95 KB	488.28 KB	-	**59.77** KB	**6.64** KB	-	788.90 KB
	30.00%	Client	0.07 KB	68.29 + **66.37** KB	488.28 KB	35.16 KB	-	**0.07** KB	**20.51** KB	-	678.75 KB
		Server	66.41 KB	68.29 + **97.56** KB	1.95 KB	488.28 KB	-	**46.48** KB	**19.92** KB	-	788.90 KB
500	0.00%	Client	0.07 KB	34.11 + **33.17** KB	488.28 KB	17.58 KB	-	**0.07** KB	**0.00** KB	-	573.28 KB
		Server	33.20 KB	34.11 + **48.73** KB	0.98 KB	488.28 KB	-	**33.20** KB	**0.00** KB	-	638.51 KB
	10.00%	Client	0.07 KB	34.11 + **33.17** KB	488.28 KB	17.58 KB	-	**0.07** KB	**3.42** KB	-	576.70 KB
		Server	33.20 KB	34.11 + **48.73** KB	0.98 KB	488.28 KB	-	**29.88** KB	**3.32** KB	-	638.51 KB
	30.00%	Client	0.07 KB	34.11 + **33.17** KB	488.28 KB	17.58 KB	-	**0.07** KB	**10.26** KB	-	583.53 KB
		Server	33.20 KB	34.11 + **48.73** KB	0.98 KB	488.28 KB	-	**23.24** KB	**9.96** KB	-	638.51 KB

[1]The dimension of gradient vectors is set to 100K and the batch size is 1. The entries of server are outgoing communication sent from the server to a single client. The underlined bold figures stand for the costs for verification in our protocol, compared with our implementation of [7] in the same experimental environment and parameter settings.

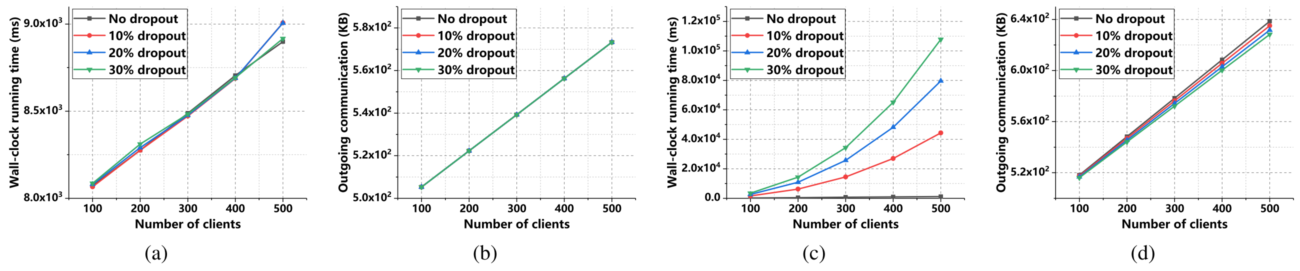


Fig. 8. Total computation and outgoing communication overhead, as the number of clients increases. (a) Computation overhead per client. (b) Outgoing communication overhead per client. (c) Computation overhead of the server. (d) Outgoing communication overhead of the server (for a single client). The dimension of gradient vectors is set to 100K and the batch size is 1. Assume Case I dropout.

A. Generic Maliciously-Secure Multiparty Computation

In general, FL involves an aggregation server and a set of clients and therefore can be regarded as a specific multiparty computation problem. Although there are piles of works [14]–[21], [31] guarantee the integrity of computation in the existence of a malicious (i.e., active) adversary, they are not suitable for FL settings. For garbled circuit-based protocols [19]–[21], [31], they deal with the malicious adversary at the cost of expensive communication overhead (e.g., that led by *commit-and-prove* [18] or *cut-and-choose* [21], [32] technique) and cannot be deployed on a large scale. For secret sharing-based protocols [14]–[16], each client has to divide each entry of its input into (additive) shares and

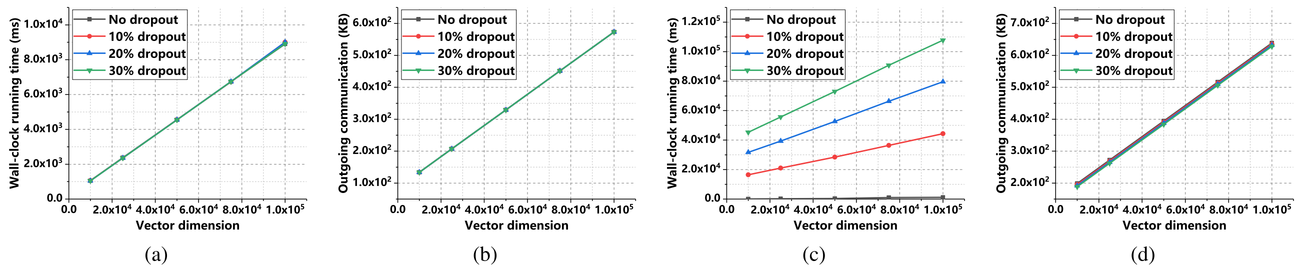


Fig. 9. Total computation and outgoing communication overhead, as the dimension of gradient vectors increases. (a) Computation overhead per client. (b) Outgoing communication overhead per client. (c) Computation overhead of the server. (d) Outgoing communication overhead of the server (for a single client). The number of clients is set to 500 and the batch size is 1. Assume Case I dropout.

send them to each other client, which results in a rather expensive communication overhead in $O(Nd)$. In addition, these protocols rely on homomorphic encryption to compute MACs of shares. Their distributed homomorphic decryption sub-protocols cannot tolerate dropout in a threshold manner directly and therefore are not suitable for FL settings where dropout is common.

B. Machine Learning Based on Cryptographic Protocols

There are two lines of the cryptographic research regarding privacy and verifiability issues in machine learning, i.e., secure model prediction and secure model training. For secure model prediction, the goal is to securely query a model without revealing model inputs and the information of the model. All works [33]–[37] require that the model being queried was well-trained in advance, which differs from the motivation of FL.

For secure model training, there are several works [38]–[40] being proposed to train a model while guarantee integrity of the derived model and the privacy of users. Unfortunately, these works will result in expensive overhead when applied to FL settings. For example, Mohassel *et al.* [38] proposed a framework to convert between any two kinds of secret shares and built up general protocols to train a neural network, where each client is supposed to share each entry of its gradient vector. The communication overhead will blow up as the dimension of vector increases. In addition, as the number of clients increases, to preserve the robustness against dropout, the size of the share held by each client blows up as well.

There are also some works [41], [42] based on homomorphic encryption (HE) which address input privacy only. These works rely on a stronger adversarial model where the secret decryption key of encrypted gradients is only known to all clients and these clients are honest. However, in our adversarial model (Section II-C), the adversary can acquire this key from controlled corrupted clients and decrypts all encrypted gradients, which undermines the security of [41], [42]. It is possible to secret share the decryption key (in a threshold manner, e.g., [43], [44]) among all clients to mitigate this issue. Unfortunately, such a construction complicates the decryption of encrypted gradients and, in the FL setting where dropout is common, will significantly increase the size of key shares and the communication between the server and a client as the number of clients increases. So, we believe the protocol based on double-masking is more efficient than that based on HE. In addition, HE is a primitive that can be used to address input privacy only. To achieve both input privacy and verifiability, some other cryptographic primitives (e.g., zero-knowledge proof) are required. This paper finds such a primitive that can be used to replace time-consuming zero-knowledge proof adopted by [8].

To the best of our knowledge, there is only one work [8] for secure verifiable aggregation in FL. However, this work adopts heavy zero-knowledge proof, which results in impractical performance and unaffordable communication overhead.

C. Differential Privacy

A notable fact is that privacy-preserving cryptographic protocols cannot fully prevent privacy disclosure in machine learning. It is known that privacy disclosure can be resulted from the statistical characteristics (e.g., confidence information, prediction outcome) of machine learning algorithms. For example, there are some *black-box* privacy attacks [22]–[26] leveraging these statistical characteristics. Treating cryptographic protocols as black-box oracles, such black-box attacks can be applied to the machine learning paradigms even if they are securely realized by cryptographic protocols. That is, as long as these protocols implement the functionality of machine learning, the statistical leakage is inevitable. A possible defense is to combine the cryptographic protocols with differential privacy [45]–[48].

Differentially private federated learning [49]–[53] has been extensively studied in the literature. A few of existing works [49], [53] consider how to combine differentially private mechanisms with multiparty computation to protect not only the statistical characteristics but also the raw inputs of *all* honest clients. However, these works do not address the verifiability issue in FL. In this work, we are devoted to fix this issue of FL with practical overhead and differential privacy is not the primary goal. A straightforward way to make our protocol differentially private is to add Gaussian or Laplacian noises to original gradients and then regard the noisy gradients as inputs to our protocol, although it might not achieve the best utility of the global model. An interesting direction for future work would be to explore how to design an optimal differentially private mechanism that is compatible with our practical secure verifiable aggregation protocol.

D. Byzantine-Robust Aggregation

A Byzantine client in FL can send arbitrary values to the aggregation server to influence model convergence. To deal with this issue, one popular mitigation is to use Byzantine-robust aggregation, which is mostly related to our aggregation protocol. There are several alternative aggregation mechanisms [54]–[57] to the standard federated averaging to ensure that convergence is not significantly influenced by Byzantine clients, in which [57] is for non-IID settings. As noted in [58], it is quite challenging to devise a Byzantine-robust aggregation mechanism for non-IID FL datasets.

Byzantine-robust aggregation can be regarded as a supplement to the secure verifiable aggregation based on cryptographic protocols since the latter guarantees only input privacy and the integrity of aggregation with respect to the used inputs. In other words, secure verifiable aggregation cannot prevent Byzantine clients to use malformed inputs to do harm to either the training process or the resulting global model.

However, how to combine Byzantine-robust aggregation [54]–[57] with secure verifiable aggregation is still an open problem. There are several technical challenges to be addressed by future work. First, these works adopt a completely different adversarial model to that of secure verifiable aggregation (e.g., Section II-C). That is, Byzantine aggregation assume only Byzantine clients that send arbitrary values to an honest aggregation server and does not consider the case where the server is corrupted. Second, these works implicitly assume the aggregation server has access to the plaintext stochastic gradients uploaded by clients, which violates input privacy. Since Byzantine-robust aggregation mechanisms usually adopt more complicated arithmetic (e.g., median or trimmed mean) than federated averaging, it is challenging to devise a tailored dropout-tolerant protocol for these mechanisms to achieve input privacy and verifiability additionally.

VII. CONCLUSION

In this paper, we studied how to realize verifiable aggregation in FL in a communication-efficient and fast way, and proposed a protocol named VERIFL. We show by experiments that VERIFL is capable of dealing with (i) high-dimensional gradient vectors, (ii) a large number of clients, and (iii) high dropout rate in real-world applications with practical performance. Notably, in VERIFL, the verification cost in terms of the outgoing communication is *independent* of the gradient dimension, resulting in $110.6\times$ improvement in communication even with $5\times$ larger gradient vectors in our experiments.

ACKNOWLEDGMENT

Xiaojie Guo, Jiqiang Gao, and Boyu Hou are supervised by Zheli Liu and have the same contribution to this paper.

REFERENCES

[1] H. Brendan McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016, *arXiv:1602.05629*. [Online]. Available: http://arxiv.org/abs/1602.05629

[2] J. Konečný, H. Brendan McMahan, F. X. Yu, P. Richtárik, A. Theertha Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*. [Online]. Available: http://arxiv.org/abs/1610.05492

[3] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," 2019, *arXiv:1902.01046*. [Online]. Available: http://arxiv.org/abs/1902.01046

[4] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 603–618.

[5] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 691–706.

[6] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 739–753.

[7] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1175–1191.

[8] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and verifiable federated learning," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 911–926, 2020.

[9] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," 2018, *arXiv:1807.00459*. [Online]. Available: http://arxiv.org/abs/1807.00459

[10] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," 2017, *arXiv:1712.05526*. [Online]. Available: http://arxiv.org/abs/1712.05526

[11] Y. Liu *et al.*, "Trojaning attack on neural networks," in *Proc. 25th Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, San Diego, CA, USA, Feb. 2018, pp. 1–17.

[12] J. Dumford and W. Scheirer, "Backdooring convolutional neural networks via targeted weight perturbations," 2018, *arXiv:1812.03128*. [Online]. Available: http://arxiv.org/abs/1812.03128

[13] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-i.i.d. data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020.

[14] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proc. 32nd Annu. Cryptol. Conf. Adv. Cryptol. (CRYPTO)*, Santa Barbara, CA, USA, Aug. 2012, pp. 643–662.

[15] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ great again," in *Proc. 37th Annu. Int. Conf. Theory Appl. Cryptograph. Techn. Adv. Cryptol. (EUROCRYPT), Part III*, Tel Aviv, Israel, Apr./May 2018, pp. 158–189.

[16] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "Spd$\mathbb{Z}_{2^k}$: Efficient mpc mod 2^k for dishonest majority," in *Proc. 38th Annu. Int. Cryptol. Conf. Adv. Cryptol. (CRYPTO), Part II*, Santa Barbara, CA, USA, Aug. 2018, pp. 769–798.

[17] I. Damgård, C. Orlandi, and M. Simkin, "Yet another compiler for active security or: Efficient MPC over arbitrary rings," in *Proc. Annu. Int. Cryptol. Conf.* Springer, 2018, pp. 799–829.

[18] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proc. 19th Annu. ACM Symp. Theory Comput.*, 1987, pp. 218–229.

[19] C.-H. Shen *et al.*, "Two-output secure computation with malicious adversaries," in *Proc. 30th Annu. Int. Conf. Theory Appl. Cryptograph. Techn. Adv. Cryptol. (EUROCRYPT)*, Tallinn, Estonia, May 2011, pp. 386–405.

[20] B. Kreuter, A. Shelat, and C.-H. Shen, "Billion-gate secure computation with malicious adversaries," in *Proc. 21st USENIX Secur. Symp. (USENIX Secur.)*, 2012, pp. 285–300.

[21] Y. Huang, J. Katz, and D. Evans, "Efficient secure two-party computation using symmetric cut-and-choose," in *Proc. 33rd Annu. Cryptol. Conf. Adv. Cryptol. (CRYPTO), Part II*, Santa Barbara, CA, USA, Aug. 2013, pp. 18–35.

[22] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Denver, CO, USA, Oct. 2015, pp. 1322–1333.

[23] Z. Yang, J. Zhang, E.-C. Chang, and Z. Liang, "Neural network inversion in adversarial setting via background knowledge alignment," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, London, U.K., Nov. 2019, pp. 225–240.

[24] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Jose, CA, USA, May 2017, pp. 3–18.

[25] L. Song, R. Shokri, and P. Mittal, "Membership inference attacks against adversarially robust deep learning models," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, San Francisco, CA, USA, May 2019, pp. 50–56.

[26] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff nets: Stealing functionality of black-box models," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 4954–4963.

[27] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography: The case of hashing and signing," in *Proc. 14th Annu. Int. Cryptol. Conf. Adv. Cryptol. (CRYPTO)*, Santa Barbara, CA, USA, Aug. 1994, pp. 216–233.

[28] M. N. Krohn, M. J. Freedman, and D. Mazieres, "On-the-fly verification of rateless erasure codes for efficient content distribution," in *Proc. IEEE Symp. Secur. Privacy*, May 2004, pp. 226–240.

[29] Y. Lindell, *How to Simulate It–A Tutorial on the Simulation Proof Technique*. Cham, Switzerland: Springer, 2017, pp. 277–346.

[30] J. Camenisch, M. Drijvers, T. Gagliardoni, A. Lehmann, and G. Neven, "The wonderful world of global random oracles," in *Proc. 37th Annu. Int. Conf. Theory Appl. Cryptograph. Techn. Adv. Cryptol. (EUROCRYPT), Part I*, Tel Aviv, Israel, Apr./May 2018, pp. 280–312.

[31] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *Proc. USENIX Secur. Symp.*, 2011, vol. 201, no. 1, pp. 331–335.

[32] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Proc. 26th Annu. Int. Conf. Theory Appl. Cryptograph. Techn. Adv. Cryptol. (EUROCRYPT)*, Barcelona, Spain, May 2007, pp. 52–78.

[33] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, 2018, pp. 1651–1669.

[34] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar, "Xonn: Xnor-based oblivious deep neural network inference," in *Proc. 28th USENIX Secur. Symp. (USENIX Secur.)*, Santa Clara, CA, USA, Aug. 2019, pp. 1501–1518.

[35] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 619–631.

[36] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 395–412.

[37] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. 33nd Int. Conf. Mach. Learn. (ICML)*, New York City, NY, USA, Jun. 2016, pp. 201–210.

[38] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 35–52.

[39] N. Agrawal, A. Shahin Shamsabadi, M. J. Kusner, and A. Gascón, "QUOTIENT: Two-party secure neural network training and prediction," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1231–1247.

[40] I. Damgard, D. Escudero, T. Frederiksen, M. Keller, P. Scholl, and N. Volgushev, "New primitives for actively-secure MPC over rings with applications to private machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 1102–1120.

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

[42] X. Sun, P. Zhang, J. K. Liu, J. Yu, and W. Xie, "Private machine learning classification based on fully homomorphic encryption," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 352–364, Jun. 2020.

[43] D. Boneh *et al.*, "Threshold cryptosystems from threshold fully homomorphic encryption," in *Proc. 38th Annu. Int. Cryptol. Conf.*, Santa Barbara, CA, USA, Aug. 2018, pp. 565–596.

[44] A. Jain, P. M. R. Rasmussen, and A. Sahai, "Threshold fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 257, Mar. 2017.

[45] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, nos. 3–4, pp. 211–407, 2014.

[46] C. Dwork, "Differential privacy: A survey of results," in *Proc. Theory Appl. Models Comput., 5th Int. Conf. (TAMC)*, Xi'an, China, Apr. 2008, pp. 1–19.

[47] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Vienna, Austria, Oct. 2016, pp. 308–318.

[48] F. McSherry and K. Talwar, "Mechanism design via differential privacy," in *Proc. 48th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, Providence, RI, USA, Oct. 2007, pp. 94–103.

[49] S. Truex *et al.*, "A hybrid approach to privacy-preserving federated learning," in *Proc. 12th ACM Workshop Artif. Intell. Secur. (AISec)*, 2019, pp. 1–11.

[50] K. Wei *et al.*, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3454–3469, 2020.

[51] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, Vancouver, BC, Canada, Apr./May 2018.

[52] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," 2017, *arXiv:1712.07557*. [Online]. Available: http://arxiv.org/abs/1712.07557

[53] T. Ryffel *et al.*, "A generic framework for privacy preserving deep learning," 2018, *arXiv:1811.04017*. [Online]. Available: http://arxiv.org/abs/1811.04017

[54] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 2, p. 44, 2017.

[55] D. Yin, Y. Chen, K. Ramchandran, and P. L. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, Stockholm, Sweden, Jul. 2018, pp. 5636–5645.

[56] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. Adv. Neural Inf. Process. Syst. Annu. Conf. Neural Inf. Process. Syst.*, Long Beach, CA, USA, Dec. 2017, pp. 119–129.

[57] L. He, S. P. Karimireddy, and M. Jaggi, "Byzantine-robust learning on heterogeneous datasets via resampling," 2020, *arXiv:2006.09365*. [Online]. Available: https://arxiv.org/abs/2006.09365

[58] P. Kairouz *et al.*, "Advances and open problems in federated learning," 2019, *arXiv:1912.04977*. [Online]. Available: http://arxiv.org/abs/1912.04977

Xiaojie Guo received the B.Eng. degree in information security and law from Nankai University, China, in 2018, where he is currently pursuing the master's degree with the College of Cyber Science. His research interests include cryptography, multi-party computation, and privacy-preserving machine learning.

Zheli Liu received the B.Sc. and M.Sc. degrees in computer science from Jilin University, China, in 2002 and 2005, respectively, and the Ph.D. degree in computer application from Jilin University in 2009. He joined the College of Computer and Control Engineering, Nankai University, in 2011, after a Post-Doctoral Fellowship at the same university. He currently works as an Associate Professor at Nankai University. His current research interests include applied cryptography and data privacy protection.

Jin Li (Senior Member, IEEE) received the B.S. degree in mathematics from Southwest University in 2002, and the Ph.D. degree in information security from Sun Yat-sen University in 2007. He currently works as a Professor at Guangzhou University. He has been selected as the "One of science and technology new star" in Guangdong province. His research interests include applied cryptography and security in cloud computing. He has published over 50 research papers in refereed international conferences and journals and has served as the program chair or program committee member in many international conferences.

Jiqiang Gao received the B.S. degree in information security from Nankai University in 2018, where he is currently pursuing the master's degree. His main research interests include protocol security about machine learning algorithm, distributed machine learning, and adversarial machine learning. His research interest includes the attack and defense on different machine learning algorithms.

Boyu Hou is currently pursuing the master's degree in computer science with the Database and Information System Laboratory, Computer Science Department, Nankai University. Her main research field is AI security, which includes secure distributed machine learning algorithms, adversarial machine learning, and attack and defense method on machine learning algorithms.

Changyu Dong received the Ph.D. degree from Imperial College London. He is currently a Senior Lecturer with the School of Computing, Newcastle University. He has authored over 30 publications in international journals and conferences. His research interests include applied cryptography, trust management, data privacy, and security policies. His recent work focuses on designing practical secure computation protocols. His application domains include secure cloud computing and privacy preserving data mining.

Thar Baker (Member, IEEE) received the Ph.D. degree in autonomic cloud applications from LJMU in 2010. He is currently a Senior Lecturer in Software Systems with the Department of Computer Science, Faculty of Engineering and Technology, University of Sharjah. Before that, he worked as a Lecturer with the Department of Computer Science, Manchester Metropolitan University (MMU), in 2011. He has published numerous refereed research articles in multidisciplinary research areas, including cloud computing, distributed software systems, big data, algorithm design, green and sustainable computing, and autonomic web science. He has been actively involved as a member of editorial board and review committee for a number peer reviewed international journals, and is on programme committee for a number of international conferences. He was appointed as an Expert Evaluator in the European FP7 Connected Communities CONFINE project (2012–2015).