Perfect Privacy for Discriminator-Based Byzantine-Resilient Federated Learning

Yue Xia, Christoph Hofmeister, Maximilian Egger, Rawad Bitar
School of Computation, Information and Technology, Technical University of Munich, Munich, Germany
{yue1.xia, christoph.hofmeister, maximilian.egger, rawad.bitar}@tum.de

***Abstract*—Federated learning (FL) shows great promise in large-scale machine learning but introduces new privacy and security challenges. We propose ByITFL and LoByITFL, two novel FL schemes that enhance resilience against Byzantine users while keeping the users' data private from eavesdroppers. To ensure privacy and Byzantine resilience, our schemes build on having a small representative dataset available to the federator and crafting a discriminator function allowing the mitigation of corrupt users' contributions. ByITFL employs Lagrange coded computing and re-randomization, making it the first Byzantine-resilient FL scheme with perfect Information-Theoretic (IT) privacy, though at the cost of a significant communication overhead. LoByITFL, on the other hand, achieves Byzantine resilience and IT privacy at a significantly reduced communication cost, but requires a Trusted Third Party, used only in a one-time initialization phase before training. We provide theoretical guarantees on privacy and Byzantine resilience, along with convergence guarantees and experimental results validating our findings.**

***Index Terms*—Byzantine Resilience, Federated Learning, Information-Theoretic Privacy, Private Aggregation**

I. Introduction

Federated learning (FL) [3] emerged as a promising paradigm enabling a central server (federator) to train neural networks on distributed private data stored at a large number of users. The training follows an iterative structure. Per iteration, the federator sends the current global model to the users, who compute local model updates based on their local data and return these updates. The federator aggregates the users' local model updates using a certain aggregation rule and uses this aggregate to update the global model. The process is repeated until the model achieves the desired performance.

Compared to centralized machine learning, FL tackles data privacy by allowing the users to keep their sensitive data locally and send only model updates to the federator. However, these local model updates, i.e., gradients or weights, still carry sensitive information about the users' data, which can be exploited, e.g., using model inversion attacks [4], [5], to learn more information about the users' private data. To address this risk, private[1] aggregation protocols [6]–[8] are introduced, ensuring that the federator only receives the aggregated local model update necessary for the progress of the learning algorithm; hence, guaranteeing privacy [8].

Beyond privacy, FL faces security threats from Byzantine users, named after the Byzantine generals problem [9], who maliciously manipulate the global model through corrupt local updates. The authors of [10] show the vulnerability of any simple linear aggregation rule, such as FedAvg [3], against Byzantine attacks. Even a single Byzantine user can take full control of the learning process, either causing convergence to a corrupt model or preventing convergence altogether [10]. To mitigate the effect of Byzantine users, numerous *robust aggregation rules* have been proposed, e.g., [10]–[15]. On a high level, robust aggregation rules compute some statistics of the local updates to identify the outliers. The outliers are then either excluded or given low weight on the final aggregation.

Therefore, in FL settings, there is an *inherent tension between privacy and Byzantine resilience.* On the one hand, resilience requires access to individual local updates to learn their statistics. On the other hand, privacy guarantees require concealing individual updates. This inherent tension makes developing methods that are jointly secure and private a challenging task, that we will address in this work.

Many existing schemes tackle privacy and security jointly. However, none of these schemes achieve the desired Information-Theoretic (IT) privacy guarantee with Byzantine resilience in the described FL setting. For instance, some schemes rely on clustering the users into smaller groups, and therefore compromise privacy [16], [17] by leaking aggregated gradients of clusters of users; some require multiple federators during the execution of the learning algorithm [18]–[24]; and many give different privacy guarantees, such as computational privacy [23]–[39] and differential privacy [40]–[43]. Among these works, the closest to this work are BREA [25] and ByzSecAgg [26]. Both schemes use the distance-based robust aggregation rule called Krum [10] and rely on secret sharing to make Krum privacy-preserving. The computational privacy guarantee in [25], [26] comes from the use of cryptographic commitments. Those commitments ensure that Byzantine users encode the same input local model into multiple shares given to all other users. Computationally breaking the commitments reveals the value of the input local model. In addition, due to the use of Krum, both schemes leak the pairwise distances between local updates to the federator. Furthermore, BREA leaks some extra information to the federator, through not re-randomizing secret sharings, cf. [1]. This is remedied in ByzSecAgg by incorporating such a re-randomization step.

While computational privacy and differential privacy have been the primary focus in most studies due to their efficiency, both come with their challenges. Computational privacy relies on computational hardness assumptions, which fail against computationally unbounded adversaries. Differential privacy (DP) obfuscates the users' local updates using noise inserted

This project has received funding from the German Research Foundation (DFG) under Grant Agreement Nos. BI 2492/1-1 and WA 3907/7-1. Part of this work was presented at IEEE ITW'24 [1] and FL-AsiaCCS'25 [2].

[1]Private aggregation is referred to as "secure aggregation" in the literature. We use the nomenclature private aggregation to avoid confusion between privacy and security (resilience) against malicious clients.

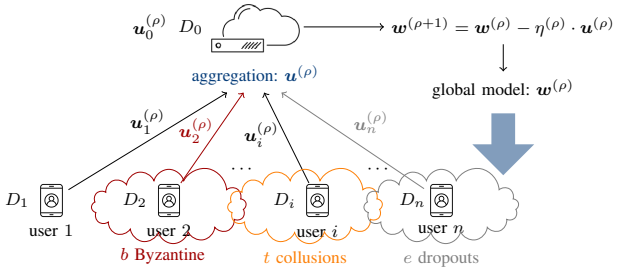

Fig. 1: Federated learning system.

by the users, which negatively affects[2] the utility of the model. Furthermore, IT privacy offers the strongest guarantee when only a limited number of entities collude to compromise privacy, withstanding even computationally unbounded adversaries and without compromising model accuracy.

We introduce ByITFL and LoByITFL, two Byzantine-resilient and IT private FL schemes. We utilize ideas from FLTrust [13] for Byzantine resilience, where the federator possesses a small representative root dataset to obtain a trustworthy model update as a reference. This enables the computation of a trust score (TS) for each user through a *discriminator function*. To enable IT privacy, we embed each user's update in a finite field and use secret sharing. To this end, we restrict the scope to polynomial discriminator functions, which are compatible with secret sharing.

II. System Model and Preliminaries

For an integer n, we denote by $[n]$ the set of positive integers $\{1, \cdots, n\}$. For a real number x, we denote by $\lfloor x \rfloor$ the largest integer less than or equal to x. Vectors are denoted by bold lowercase letters, e.g., $\boldsymbol{x}$. The dot product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is denoted by $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$. Given $x_1, \cdots, x_n$ and a set $\mathcal{S} \subseteq [n]$ of indices, we define $x_{\mathcal{S}} \triangleq \{x_i : i \in S\}$. For two random variables X and Y, $H(X)$ and $I(X;Y)$ denote the entropy of X and the mutual information of X and Y, respectively. The base of the logarithm will be clear from the context.

A. Federated Learning Model

We consider the FL setting with an honest-but-curious federator orchestrating the learning algorithm on the data of n users, as illustrated in Fig. 1. Among the users, up to b can be Byzantine, and up to t can collude with each other (and with the federator) to infer information about other users' data. In addition, in every iteration, up to e users can drop out and be unresponsive. Each user $i \in [n]$ holds a local dataset $D_i \in \mathbb{R}^{o_i \times d}$ drawn according to a (possibly distinct) distribution $\mathcal{D}_i$. As in [13], we assume that the federator possesses a small root dataset D_0 representing[3] the union of the users' data $D = \bigcup_{i=1}^{n} D_i$. The goal is to find a model $\boldsymbol{w}$ that best represents the users' private data, i.e., given a loss function $F(\boldsymbol{w}, D_i)$ parametrized by $\boldsymbol{w}$, the federator wants to solve the optimization problem

$$\min_{\boldsymbol{w}} \sum_{i=1}^{n} F(\boldsymbol{w}, D_i). \tag{1}$$

The federator employs the iterative stochastic gradient descent algorithm that starts with an initial global model $\boldsymbol{w}^{(0)}$ and local models $\boldsymbol{w}_1^{(0)}, \cdots, \boldsymbol{w}_n^{(0)}$ held by the users. At each global iteration ρ, the federator broadcasts the current global model $\boldsymbol{w}^{(\rho)}$ to all users. Each user $i \in [n]$ initializes its local model as the current global model, i.e., $\boldsymbol{w}_i^{(\rho,0)} = \boldsymbol{w}^{(\rho)}$, and updates it for $C \geq 1$ local iterations as

$$\boldsymbol{w}_i^{(\rho,c+1)} = \boldsymbol{w}_i^{(\rho,c)} - \eta_i \cdot \nabla F(\boldsymbol{w}_i^{(\rho,c)}; \tilde{D}_i),$$

where η_i is the local learning rate, c, $0 \leq c \leq C-1$, is the local iteration index, and $\nabla F(\boldsymbol{w}_i^{(\rho,c)}; \tilde{D}_i)$ is the gradient of the loss function evaluated at a subset of user i's data $\tilde{D}_i \subseteq D_i$. Upon finishing local training, users return their model updates $\boldsymbol{u}_i^{(\rho)} = \boldsymbol{w}_i^{(\rho,C)} - \boldsymbol{w}^{(\rho)}$ to the federator.

Meanwhile, the federator runs the same computations on D_0 to obtain a federator model update $\boldsymbol{u}_0^{(\rho)}$. Upon receiving the local updates, the federator aggregates them according to some aggregation rule AGG, i.e., it computes $\boldsymbol{\nu}^{(\rho)} = \text{AGG}(\boldsymbol{u}_0^{(\rho)}, \cdots, \boldsymbol{u}_n^{(\rho)})$. The federator obtains the global update $\boldsymbol{u}^{(\rho)} = \boldsymbol{\nu}^{(\rho)}$. Consider η as the global learning rate, the federator updates the global model[4]

$$\boldsymbol{w}^{(\rho+1)} = \boldsymbol{w}^{(\rho)} - \eta \cdot \boldsymbol{u}^{(\rho)}.$$

B. Threat Model and Defense Goals

We denote by $\mathcal{T} \subset [n]$, $|\mathcal{T}| = t$, the set of colluding users, by $\mathcal{B} \subset [n]$, $|\mathcal{B}| = b$, the set of Byzantine users and by $\mathcal{H} \triangleq [n] \setminus \mathcal{T} \cup \mathcal{B}$ the set of benign users.

1) IT Privacy: We do not assume curious entities to have limited computing resources. Our schemes guarantee IT privacy of the benign users' local updates in each iteration.

We consider a set of $\mathcal{T}$ colluding users. The federator is assumed to be honest-but-curious, i.e., it tries to infer private data about the benign users, but it honestly conducts the protocol, and does not proactively leak data that would compromise the system or give an advantage to adversaries.

Since we allow the federator to collude with the set $\mathcal{T}$ of colluding users. We require that the colluding users and the federator cannot learn further information about the benign users' local model updates $\boldsymbol{u}_{\mathcal{H}}^{(\rho)}$ beyond what can inherently be inferred from *(a)* the colluding users' local model updates $\boldsymbol{u}_{\mathcal{T}}^{(\rho)}$ and the federator model update $\boldsymbol{u}_0^{(\rho)}$; *(b)* the colluding users' local datasets $D_{\mathcal{T}}$ and the the root dataset D_0; *(c)* the current global model $\boldsymbol{w}^{(\rho)}$; and *(d)* the current aggregation $\boldsymbol{\nu}^{(\rho)}$; all of which are required by the learning algorithm and may give information about the other users' local datasets. For global iteration ρ, the privacy guarantee is formulated as:

$$I(\boldsymbol{u}_{\mathcal{H}}^{(\rho)}; M_{\mathcal{T}}^{(\rho)}, M_0^{(\rho)} \,|\, \boldsymbol{u}_{\mathcal{T}}^{(\rho)}, \boldsymbol{u}_0^{(\rho)}, D_{\mathcal{T}}, D_0, \boldsymbol{w}^{(\rho)}, \boldsymbol{\nu}^{(\rho)}) = 0, \tag{2}$$

where $M_{\mathcal{T}}^{(\rho)}$ is the set of potential messages[5] received by

[2] IT private aggregation can be combined with DP to allow for higher privacy guarantees for a fixed model utility. The reason is that less noise would be needed since the federator only observes the aggregation of the local updates, cf. [44].

[3] In practice, this small root data set can be obtained, e.g., through manually labelling a public dataset, see [13] for details.

[4] While presented in the simple gradient descent setting, our scheme does not depend on the exact update rule and is applicable to, e.g., momentum, higher order methods, and adaptive learning rate schedules.

[5] We will consider interactive algorithms, where the clients and the federator exchange messages to mitigate the effect of Byzantine users.

the colluding users during the execution of the algorithm at iteration ρ, and $M_0^{(\rho)}$ denotes the messages received by the federator during the execution of the algorithm at iteration ρ. We omit the global iteration index ρ in the rest for brevity.

2) Byzantine Resilience: We consider Byzantine users arbitrarily deviating from the protocol and having access to all users' datasets. Specifically, Byzantine users may: *(a)* corrupt their local updates; and *(b)* perform the secret sharing and all other computations dishonestly. A federated learning scheme is said to be Byzantine resilient if it converges, i.e., solves the optimization problem in Eq. (1), even if up to b Byzantine users behave maliciously.

3) Dropout Tolerance: Users can drop out during the protocol. This may be due to users experiencing delays, network congestion or similar reasons. A dropout tolerant federated learning scheme converges even if up to e users drop out per iteration of the algorithm.

4) Desired Schemes: Our goal is to design a FL scheme that simultaneously satisfies the following: *(a)* IT privacy against an honest-but-curious federator and against any collusion of up to t users; *(b)* Byzantine resilient against b Byzantine users; and *(c)* dropout tolerance of up to e users.

C. Preliminaries

1) Threshold Secret Sharing: We rely on the powerful tool of (n, k, t)*-threshold secret sharing*. Consider a secret vector $\boldsymbol{s}$ drawn from a finite alphabet, usually a finite field $\mathbb{F}_p^d$, partitioned into $m = k - t$ sub-vectors[6], $\boldsymbol{s}_i \in \mathbb{F}_p^{d/m}$, i.e., $\boldsymbol{s}^T = [\boldsymbol{s}_1^T, \cdots, \boldsymbol{s}_m^T]$. An (n, k, t)-threshold secret sharing (SecShare) scheme encodes $\boldsymbol{s}$ into n secret shares denoted by $\boldsymbol{s}[i]$ for $i \in [n]$. The encoding utilizes a degree-$(k-1)$ polynomial $f(x)$ encoding the m sub-vectors with t vectors drawn independently and uniformly at random from the finite field $\mathbb{F}_p^{d/m}$. An example of such a polynomial is given in the sequel. Each secret share is an evaluation of the encoding polynomial at a distinct non-zero evaluation point $\alpha_i \in \mathbb{F}_p$, i.e., $\boldsymbol{s}[i] = f(\alpha_i)$ for $i \in [n]$. Any $k \leq n$ or more shares allow full reconstruction of $\boldsymbol{s}$, while any $t < k$ or less shares are statistically independent of $\boldsymbol{s}$. We call k the threshold of the SecShare scheme. Instantiations of a (n, k, t)-threshold SecShare scheme can be Shamir SecShare [45], McEliece-Sarwate SecShare [46], Lagrange coded computing [47] or additive SecShare (see e.g. [48]).

The SecShare schemes we consider in this paper are linear, i.e., homomorphic with addition and scaling by a constant, however they are not inherently homomorphic with multiplication. Thus, additional steps are needed for multiplication on the secret shares, i.e., either by re-randomizing the coefficients of the product of the polynomials and requiring more users for decoding [49], [50], or by unlocking the multiplicative homomorphism through converting multiplication to linear computations using Beaver triples [51].

In our setting, users secret-share some information with each other that will be further processed. Verifying that all the shares come from the same encoding can be done through the use of cryptographic commitments to the coefficients of the encoding polynomial, e.g., [25], [26]. However, this verification step weakens the privacy to computational privacy guarantees. To guarantee IT privacy, IT verifiable secret sharing (ITVSS) [50], [52] is needed. ITVSS uses a bivariate polynomial $\boldsymbol{S}(x, y)$ to share the secret, where each user receives two univariate polynomials $f^i(x) = \boldsymbol{S}(x, \alpha_i)$ and $\boldsymbol{g}^i(x) = \boldsymbol{S}(\alpha_i, y)$ as its share, where $f(x)$ will be the encoding polynomial of the actual secret sharing and $\boldsymbol{g}(x)$ will be the authentication information to guarantee the use of the identical encoding polynomial $f(x)$. The secret share is a valid share if $f^i(\alpha_j) = \boldsymbol{S}(\alpha_j, \alpha_i) = \boldsymbol{g}^j(\alpha_i)$, which can be verified with the help of all other users [50], [52].

2) Lagrange Coded Computing: Lagrange Coded Computing (LCC) [47] is one instantiation of the threshold SecShare schemes. In LCC, a secret vector $\boldsymbol{s}$ is partitioned and secret-shared among n users using the degree-$(m+t-1)$ polynomial

$$f_{\boldsymbol{s}}(x) = \sum_{j\in[m]} \boldsymbol{s}_j \cdot \prod_{l\in[m+t]\setminus\{j\}} \frac{x-\beta_l}{\beta_j-\beta_l} + \sum_{j\in[t]} \boldsymbol{r}_j \cdot \prod_{l\in[m+t]\setminus\{m+j\}} \frac{x-\beta_l}{\beta_{m+j}-\beta_l}, \forall j \in [n], \tag{3}$$

where $\beta_1, \cdots, \beta_{m+t}$ are $m+t$ distinct non-zero elements from $\mathbb{F}_p$ and the $\boldsymbol{r}_j$'s are chosen independently and uniformly at random from $\mathbb{F}_p^{d/m}$. Note that $f_{\boldsymbol{s}}(\beta_1) = \boldsymbol{s}_1, \cdots, f_{\boldsymbol{s}}(\beta_m) = \boldsymbol{s}_m$. Secret shares are computed by evaluating $f_{\boldsymbol{s}}(x)$ at n distinct non-zero values $\{\alpha_l\}_{l\in[n]}$, which are selected from $\mathbb{F}_p$ such that $\{\alpha_l\}_{l\in[n]} \cap \{\beta_l\}_{l\in[m]} = \varnothing$, i.e. $\boldsymbol{s}[i] = f_{\boldsymbol{s}}(\alpha_i)$. The reconstruction of $\boldsymbol{s}$ follows by interpolating $f_{\boldsymbol{s}}(x)$ and evaluating it at $x = \beta_1, \cdots, \beta_m$.

It is worth mentioning that for an arbitrary degree-τ polynomial h, LCC allows the computation of $h(\boldsymbol{s})$ performed over the secret shares of $\boldsymbol{s}$. Specifically, each user $i \in [n]$, holding $\boldsymbol{s}[i] = f_{\boldsymbol{s}}(\alpha_i)$, computes $h(\boldsymbol{s}[i]) = h(f_{\boldsymbol{s}}(\alpha_i))$ locally. This gives an evaluation of the polynomial $h(f_{\boldsymbol{s}}(x))$ at the point α_i. Upon having more than $(m+t-1)\tau+1$ correct evaluations from the users, $h(f_{\boldsymbol{s}}(x))$ can be interpolated. Obtaining $h(\boldsymbol{s})$ is done by evaluating $h(f_{\boldsymbol{s}}(x))$ at $\{\beta_l\}_{l\in[m]}$, i.e., $h(\boldsymbol{s}) = [h(\boldsymbol{s}_1)^T, \cdots, h(\boldsymbol{s}_m)^T]^T = [h(f_{\boldsymbol{s}}(\beta_1))^T, \cdots, h(f_{\boldsymbol{s}}(\beta_m))^T]^T$.

3) Beaver Triples: To unlock the multiplicative homomorphism of threshold SecShare schemes, we use *Beaver triples*. A Beaver triple consists of two random variables γ and ω drawn independently and uniformly at random from a finite field $\mathbb{F}_p$ and their product $\kappa = \gamma\omega$. The values of γ, ω and κ are secret-shared independently to n users using an (n, k, t)-threshold SecShare scheme.

Assume we have two secrets s and v and a Beaver triple that are secret-shared with n users using an identical threshold SecShare scheme. Each user i holding the shares of the Beaver triple $\gamma[i]$, $\omega[i]$, $\kappa[i]$, and the shares of the secrets $s[i]$ and $v[i]$, wants to compute a secret share of the product sv without knowing the values of s nor v. To that end, each user: *(a)* computes $s[i] - \gamma[i]$ and $v[i] - \omega[i]$; *(b)* sends the results to one dedicated user (in our case, the federator), who reconstructs $s - \gamma$ and $v - \omega$ and publicly announces their values; and *(c)* computes its secret share of the product as $(sv)[i] \triangleq (s-\gamma)(v-\omega) + (s-\gamma)\omega[i] + (v-\omega)\gamma[i] + \kappa[i]$, which only involves addition and scaling of secret shares. Reconstructing from enough shares $(sv)[i]$ gives $sv = (s-\gamma)(v-\omega) + (s-\gamma)\omega + (v-\omega)\gamma + \kappa$.

[6]We assume $m \mid d$. Otherwise, this can be ensured through zero-padding.

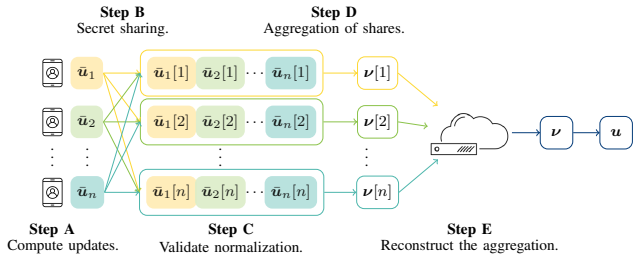

Fig. 2: The main steps in each training iteration of ByITFL and LoByITFL.

Therefore, using Beaver triples, we convert the multiplication on the secret shares to linear computations, which can be trivially solved due to the additive homomorphism of the chosen threshold SecShare scheme.

III. Main Results

We introduce ByITFL and LoByITFL, two Byzantine-resilient, dropout tolerant, and IT private FL schemes. The former has a high communication cost incurred by the joint privacy and Byzantine resilience requirements. The latter uses a trusted third party only during an initialization phase to reduce communication costs. Each training iteration of ByITFL and LoByITFL consists of the following main steps:

A. Users **compute, normalize and quantize** their local **model updates**. Similarly, the federator computes, normalizes and quantizes the federatot model update.
B. Each user's **update** is **secret-shared** among all users.
C. **Validation of the normalization** based on the received secret shares.
D. Users **compute a secret representation** of the aggregation of the validated updates.
E. The federator receives shares of the aggregation from the users to **reconstruct the aggregated updates**.

The main steps are illustrated in Fig. 2. To guarantee Byzantine resilience, our schemes utilize ideas from FLTrust [13]. Specifically, a TS is computed for each user using a discriminator function. Users' local updates are scaled by their TSs during aggregation. To simultaneously enable IT privacy, we design a new discriminator function based on polynomials that can be computed on the secret shares. The main challenge is designing the discriminator function such that it both (*a*) is computable over secret shares in a private manner; and (*b*) prevents the malicious gradients from affecting the convergence of the algorithm. Next, we provide the main ingredients and theoretical guarantees for each of our schemes.

A. ByITFL

Each local update vector is partitioned into m sub-vectors and secret-shared among all users using LCC [47], which is a threshold SecShare scheme and is described in Section II-C2. The shares are checked for corruptions using an ITVSS scheme [50], [52]. Users compute the aggregation on the secret shares using a carefully designed discriminator function, and re-randomize [49] before sending the result to the federator to maintain perfect privacy. Treating the erroneous computations sent by Byzantine users as errors and the dropouts as erasures in a Reed-Solomon (RS) code, cf. [46], the federator decodes the error-free aggregation result.

Theorem 1. *Consider a federated learning setting with n users, out of which b are Byzantine, t are curious and e may drop out at any iteration with $n \geq 2b + (\tau + 2) \cdot (m + t - 1) + e + 1$, where m is the number of sub-vectors into which each local update is partitioned and τ is the degree of the discriminator polynomial. ByITFL guarantees the following:*
1) IT privacy against any t users and the federator according to Eq. (2) in each iteration;
2) Resilience against b Byzantine users and robustness against e dropouts;
3) Communication cost of $O(\frac{d}{m}n^3 + n^4)$ scalars per user and $O(\frac{d}{m}n + n^2)$ for the federator, per iteration. Computation cost of $O\left((\frac{d}{m}n^3 + n^4)\log^2 n \log\log n\right)$ per user and $O\left((\frac{d}{m}n + n^2)\log^2 n \log\log n\right)$ at the federator; and
4) Convergence as shown in Theorem 3.

Proof. The proof is given in Appendix A. □

B. LoByITFL

We introduce LoByITFL to reduce the high communication and computation costs incurred by ByITFL, while still ensuring Byzantine resilience and IT privacy. This reduction comes at the expense of requiring a Trusted Third Party (TTP), which, however, is only used in an initialization phase to distribute Beaver triples [51] to the users and the federator. Beaver triples enable a multiplicative homomorphism of SecShare schemes, allowing polynomial computations on shared updates without re-randomization and without increasing the degree of the encoding polynomial of the secret sharing. Additionally, an additively homomorphic Message Authentication Code (MAC) is utilized to ensure integrity, preventing malicious users from performing corrupt polynomial computation [48].

Theorem 2. *Consider a federated learning setting with a TTP and n users, out of which b are Byzantine, t are curious, and e may drop out at any iteration with $n \geq b+m+t+e$, where m is the number of sub-vectors into which each local update vector is partitioned. LoByITFL guarantees the following:*
1) IT privacy against any t users and the federator according to Eq. (2) in each iteration;
2) Resilience against b Byzantine users and robustness against e dropouts;
3) Communication cost of $O\left((\frac{d}{m} + \tau)n\right)$ scalars per user and $O\left((\frac{d}{m} + \tau)n^2\right)$ for the federator, per iteration; Computation cost of $O\left((\frac{d}{m} + \tau)n^2 \log^2 n \log\log n\right)$ per user and $O\left((\frac{d}{m} + \tau)n\right)$ operations at the federator; and
4) Convergence as shown in Theorem 3.

Proof. The proof is given in Appendix B. □

C. Convergence for ByITFL and LoByITFL

We denote by $F(\boldsymbol{w}) = \mathbb{E}[F(\boldsymbol{w}, D)]$ the expectation of the loss on the union of the users' datasets. We make the same assumptions as in [13, Assumptions 1, 2 and 3]: (*a*) The expected loss $F(\boldsymbol{w})$ is μ-strongly convex and differentiable with L-Lipschitz continuity, and the loss function $F(\boldsymbol{w}, D_i)$

TABLE I: Complexity Analysis with respect to the total number of users n, the dimension of the model updates d, the partitioning parameter m, and the degree τ of the discriminator polynomial.

	Per-User		Federator	
	Computation	Communication	Computation	Communication
BREA [25]	$O(dn\log^2 n + dn^2)$	$O(dn+n^2)$	$O((dn+n^3)\log^2 n \log\log n)$	$O(dn+n^3)$
ByzSecAgg [26]	$O(\frac{d}{m}n\log^2 n + \frac{d}{m}n^2)$	$O(\frac{d}{m}n+n^2)$	$O((\frac{d}{m}n+n^3)\log^2 n\log\log n)$	$O(\frac{d}{m}n+n^3)$
ByITFL	$O((\frac{d}{m}n^3+n^4)\log^2 n\log\log n)$	$O(\frac{d}{m}n^3+n^4)$	$O((\frac{d}{m}n+n^2)\log^2 n\log\log n)$	$O(\frac{d}{m}n+n^2)$
LoByITFL	$O((\frac{d}{m}+\tau)n)$	$O((\frac{d}{m}+\tau)n)$	$O((\frac{d}{m}+\tau)n^2\log^2 n\log\log n)$	$O((\frac{d}{m}+\tau)n^2)$

is L_1-Lipschitz probabilistically with parameter δ. (b) The gradient $\nabla F(\boldsymbol{w}, D_i)$ of the loss function at the optimal global model, and the gradient difference between the loss with the optimal global model and the loss with any $\boldsymbol{w}$ are bounded. Specifically certain inner products with an arbitrary unit vector are sub-exponential with parameters σ_1 and Λ_1 in the former case, and σ_2 and Λ_2 in the latter case. (c) Each user's local dataset D_i for $i \in n$ and the root dataset D_0 are sampled independently from the same distribution.

Theorem 3. *Given the assumptions above, ByITFL and LoByITFL converge, i.e., the difference between the global model $\boldsymbol{w}^{(\rho)}$ obtained by the federator after ρ global iterations and the optimal global model $\boldsymbol{w}^\star \triangleq \arg\min F(\boldsymbol{w})$ obtained in the case where all users would have been honest is bounded by*

$$\|\boldsymbol{w}^{(\rho)} - \boldsymbol{w}^\star\| \le (1-\Gamma)^\rho \|\boldsymbol{w}^{(0)} - \boldsymbol{w}^\star\| + 12\eta\Delta_1/\Gamma,$$

where η is the global learning rate,

$$\Gamma = 1 - (\sqrt{1-\mu^2/4L^2} + 24\eta\Delta_2 + 2\eta L),$$

$$\Delta_1 = \sigma_1\sqrt{\frac{2}{|D_0|}\left(d\log 6 + \log(\frac{3}{\delta})\right)},$$

$$\Delta_2 = \sqrt{\frac{2\sigma_2^2}{|D_0|}\left(d\log\frac{18L_2}{\sigma_2} + \frac{d}{2}\log\frac{|D_0|}{d} + \log\frac{6\sigma_2^2 z\sqrt{|D_0|}}{\Lambda_2\sigma_1\delta}\right)},$$

$|D_0|$ is the size of the root dataset, $L_2 = \max\{L, L_1\}$ and z is a positive integer satisfying $\|\boldsymbol{w}-\boldsymbol{w}^*\| \le z\sqrt{d}$ for all $\boldsymbol{w} \in \mathbb{R}^d$.

Proof. The proof is given in Appendix C. □

In Section IV and Section V, we present ByITFL and LoByITFL in detail, respectively. We compare the communication and computation complexity of ByITFL with respect to n, d and m to BREA [25] and ByzSecAgg [26] in Table I.

IV. ByITFL

We explain the main steps of ByITFL followed in each global iteration. In this description, we omit the current global iteration index ρ for clarity of presentation.

A. Normalization and Quantization

To defend against Byzantine attacks performed on the magnitude of the local update vector, the federator, and all users first normalize $\boldsymbol{u}_i$ to a unit vector $\tilde{\boldsymbol{u}}_i$,

$$\tilde{\boldsymbol{u}}_i \triangleq \frac{\boldsymbol{u}_i}{\|\boldsymbol{u}_i\|}, \forall i \in \{0,1,\cdots,n\}.$$

This prevents Byzantine users from influencing the model by sending extremely large/small local updates.

Since the training process is performed in the real domain and LCC (like every IT private SecShare) works over finite fields, it is essential to transfer $\tilde{\boldsymbol{u}}_i \in \mathbb{R}^d$ to vectors in a finite field $\bar{\boldsymbol{u}}_i \in \mathbb{F}_p^d$, where p is a large prime or power of a prime. Therefore, users apply an element-wise stochastic quantizer $Q_q(x)$ with $2q+1$ quantization intervals as in [25], [26]:

$$Q_q(x) = \begin{cases} \frac{\lfloor qx \rfloor}{q} & \text{with prob. } 1-(qx-\lfloor qx\rfloor), \\ \frac{\lfloor qx\rfloor+1}{q} & \text{with prob. } qx-\lfloor qx\rfloor, \end{cases}$$

where q is the number of quantization levels. The relation between p and q is explained later. Note that the stochastic quantization is unbiased, i.e., $\mathbb{E}_Q[Q_q(x)] = x$. Let $\Phi(x) \triangleq x \mod p$ be the function mapping integers to values in $\mathbb{F}_p$. The quantization is defined as $\bar{\boldsymbol{u}}_i \triangleq \Phi(q \cdot Q_q(\tilde{\boldsymbol{u}}_i))$.

B. Sharing of the Normalized Model Updates

The normalized update vectors are partitioned into smaller sub-vectors and secret-shared using LCC and ITVSS. The partition of $\bar{\boldsymbol{u}}_i$ into m smaller subvectors is done as

$$\bar{\boldsymbol{u}}_i = [\bar{\boldsymbol{u}}_{i1}^T, \bar{\boldsymbol{u}}_{i2}^T, \cdots, \bar{\boldsymbol{u}}_{im}^T]^T, \forall i \in \{0,1,\cdots,n\},$$

where each sub-vector is of size $\frac{d}{m}$ and $m \le \frac{n-e-1}{\tau+2} - b - t + 1$.

We assume $\boldsymbol{u}_0$ does not require privacy. The federator broadcasts $\bar{\boldsymbol{u}}_0$ to the users. Each user i secret-shares $\bar{\boldsymbol{u}}_i$ with all users by LCC with the degree-$(m+t-1)$ encoding polynomial

$$f_i(x) = \sum_{j\in[m]} \bar{\boldsymbol{u}}_{ij} \cdot \prod_{l\in[m+t]\setminus\{j\}} \frac{z-\beta_l}{\beta_j-\beta_l} + \sum_{j\in[t]} \boldsymbol{r}_{ij} \cdot \prod_{l\in[m+t]\setminus\{m+j\}} \frac{z-\beta_l}{\beta_{m+j}-\beta_l}, \forall i \in [n], \tag{4}$$

where $\beta_1, \cdots, \beta_{m+t}$ are $m+t$ distinct non-zero elements from $\mathbb{F}_p$ and $\boldsymbol{r}_{ij}$'s are chosen independently and uniformly at random from $\mathbb{F}_p^{d/m}$. Note that the finite field size p should be large enough to avoid any wrap-around as we describe in Section IV-E. Each user $j \in [n]$ receives a secret share of $\bar{\boldsymbol{u}}_i$ from user $i \in [n]$, i.e., $\bar{\boldsymbol{u}}_i[j] = f_i(\alpha_j)$ where $\alpha_1, \cdots, \alpha_n$ are distinct non-zero element from $\mathbb{F}_p$ that satisfy $\{\alpha_l\}_{l\in[n]} \cap \{\beta_l\}_{l\in[m]} = \varnothing$. We leverage the ITVSS protocol from [52] to prevent Byzantine users from sending corrupt shares in the secret sharing step.

C. Validation of Normalization

Byzantine users may misbehave during the normalization. Thus, upon receiving a secret share, user $i \in [n]$ computes $\langle \bar{\boldsymbol{u}}_j[i], \bar{\boldsymbol{u}}_j[i]\rangle$ for each $j \in [n]$. Due to the linearity of secret sharing, this dot-product also corresponds to a secret share of the norm of $\bar{\boldsymbol{u}}_j$, i.e., $\|\bar{\boldsymbol{u}}_j\|_2^2[i] = \langle \bar{\boldsymbol{u}}_j[i], \bar{\boldsymbol{u}}_j[i]\rangle$.

If each user i sends the share $\|\bar{\boldsymbol{u}}_j\|_2^2[i]$ to the federator, the latter can decode $\|\bar{\boldsymbol{u}}_j\|_2^2$ and verify whether it is equal to one. However, the encoding polynomial of those shares $\|\bar{\boldsymbol{u}}_j\|_2^2[i]$ includes additional information about $\bar{\boldsymbol{u}}_j$. Therefore, simply sending the shares violates the IT privacy requirement. To that end, the users *re-randomize* [49], [50] the share $\|\bar{\boldsymbol{u}}_j\|_2^2[i]$ before sending it to the federator, which involves sub-sharing the users' secret shares using ITVSS [52], and linearly combining to construct the re-randomized secret shares. Upon receiving enough re-randomized shares $\|\bar{\boldsymbol{u}}_j\|_2^2[i]$, the federator utilizes error correction decoding of the underlying RS code to reconstruct $\|\bar{\boldsymbol{u}}_j\|_2^2$ for each $j \in [n]$ and checks if it is within a certain interval, i.e.,

$$\left|\|\bar{\boldsymbol{u}}_j\|_2^2 - \Phi(q \cdot Q_q(1))^2\right| < \varepsilon \cdot q^2,$$

where ε is a predefined threshold and can be set empirically. The interval is caused by the accuracy loss due to quantization. If any user does not pass the normalization check, the federator marks them as Byzantine and excludes them from future computations. Note that decoding the RS code for error correction requires $n \geq 2b + 2(m + t - 1) + e + 1$.

D. Users Secure Computation

Users compute the discriminator function on the secret shares of the model updates, and construct secret shares of the aggregation result.

This step is based on the non-private scheme FLTrust [13], which assigns to each user i a trust score determined by the cosine similarity between the federator and the local model update, i.e., $\mathrm{TS}_i = \mathrm{ReLU}\left(\cos(\theta_i)\right)$, where θ_i is the angle between $\boldsymbol{u}_i$ and $\boldsymbol{u}_0$. ReLU is the discriminator function in FLTrust. The federator then scales the local model updates by their trust scores and averages them for aggregation.

Making FLTrust IT private is not straightforward, which is why we replace ReLU by a degree-τ polynomial $h(x) = h_0 + h_1 x + \cdots + h_k x^\tau$ as the discriminator function. The choice of the discriminator function will be discussed later in Section VI-A. Therefore, the trust score for each user becomes $\mathrm{TS}_i = h(\cos(\theta_i)) = h(\langle \bar{\boldsymbol{u}}_0, \bar{\boldsymbol{u}}_i \rangle), \forall i \in [n]$, and the aggregation is

$$\boldsymbol{\nu} = \frac{1}{\sum_{i\in[n]} \mathrm{TS}_i} \cdot \sum_{i\in[n]} \left(\mathrm{TS}_i \cdot \bar{\boldsymbol{u}}_i\right) \triangleq \frac{\boldsymbol{\Sigma}_2}{\Sigma_1}, \tag{5}$$

with $\Sigma_1 \triangleq \sum_{i\in[n]} h(\langle \bar{\boldsymbol{u}}_0, \bar{\boldsymbol{u}}_i \rangle)$ and $\boldsymbol{\Sigma}_2 \triangleq \sum_{i\in[n]} \left(h(\langle \bar{\boldsymbol{u}}_0, \bar{\boldsymbol{u}}_i \rangle) \cdot \bar{\boldsymbol{u}}_i\right)$.

The federator needs to compute $\boldsymbol{\nu}$ in a private manner without learning individual users' private information beyond this quotient. The colluding users should learn nothing about other honest users' data during the computation. Both Σ_1 and $\boldsymbol{\Sigma}_2$ are polynomial functions of the model updates $\bar{\boldsymbol{u}}_0$ and $\bar{\boldsymbol{u}}_i$ for $i \in [n]$, where $\bar{\boldsymbol{u}}_0$ is known to all users and $\bar{\boldsymbol{u}}_i$'s are secret-shared among the users using LCC. Since LCC allows the computation of an arbitrary polynomial h over its secret, as described in Section II-C2, users compute the polynomials Σ_1 and $\boldsymbol{\Sigma}_2$ on their secret shares, such that each of them obtains an evaluation of Σ_1 and $\boldsymbol{\Sigma}_2$. This guarantees that any set of up to t users are not able to learn anything from the shares.

Privacy against the federator has not yet been guaranteed: if the federator were to reconstruct Σ_1 and $\boldsymbol{\Sigma}_2$ directly, some information about $\bar{\boldsymbol{u}}_1, \cdots, \bar{\boldsymbol{u}}_n$ would leak to the federator. LCC, like other threshold SecShare schemes, is additively homomorphic, but not multiplicatively. We follow the re-randomization from [49], [50] to construct the re-randomized secret shares.

The users now hold the re-randomized shares $\Sigma_1[i]$ and $\boldsymbol{\Sigma}_2[i]$. It remains to ensure that the federator obtains the quotient $\boldsymbol{\Sigma}_2/\Sigma_1$ without gaining any additional information about Σ_1 and $\boldsymbol{\Sigma}_2$. To this end, each user i: *(a)* chooses an independent value λ_i uniformly at random from $\mathbb{F}_p$ and secret-shares it by LCC and ITVSS among all users; *(b)* adds the shares of λ_j's from all user j and obtains its share of[7] $\lambda = \sum_{j=[n]} \lambda_j$; and *(c)* multiplies $\Sigma_1[i]$ and $\boldsymbol{\Sigma}_2[i]$ by $\lambda[i]$ and performs re-randomization to obtain $\lambda\Sigma_1[i]$ and $\lambda\boldsymbol{\Sigma}_2[i]$. Users send the secret shares $\lambda\Sigma_1[i]$ and $\lambda\boldsymbol{\Sigma}_2[i]$ to the federator.

E. Private Aggregation

The federator receives secret shares of the aggregation from the users to reconstruct the private aggregation by decoding an error correcting code and updates the global model.

The federator receives $(\lambda\Sigma_1)[i]$ and $(\lambda\boldsymbol{\Sigma}_2)[i]$, for which the degree of the encoding polynomial is $(\tau+1)(m+t-1)$ and $(\tau+2)(m+t-1)$, respectively. With sufficient number of users returning evaluations, the federator is able to leverage the error correction capability of RS codes [46] to decode $\lambda\Sigma_1$ and $\lambda\boldsymbol{\Sigma}_2$. Therefore, we require $n \geq 2b+(\tau+2)\cdot(m+t-1)+e+1$.

Upon decoding the correct values, the federator computes $\boldsymbol{\nu} = \frac{\lambda\boldsymbol{\Sigma}_2}{\lambda\Sigma_1}$, converts it from the finite field back to the real domain through de-quantizing by $Q_q(x)^{-1}$ and demapping by Φ^{-1}. The federator then checks whether $\boldsymbol{\nu}$ and $\boldsymbol{u}_0$ point to the same general direction, computes the model update

$$\boldsymbol{u} = \begin{cases} \|\boldsymbol{u}_0\| \cdot \frac{\boldsymbol{\nu}}{\|\boldsymbol{\nu}\|}, & \text{if } \langle \boldsymbol{\nu}, \boldsymbol{u}_0 \rangle \geq 0, \\ -\|\boldsymbol{u}_0\| \cdot \frac{\boldsymbol{\nu}}{\|\boldsymbol{\nu}\|}, & \text{if } \langle \boldsymbol{\nu}, \boldsymbol{u}_0 \rangle < 0, \end{cases} \tag{6}$$

and updates the global model for the next iteration. To ensure the correctness of the result, none of the computations should cause a wrap-around in the finite field. Each entry of the normalized gradient is in the range $-q$ to q, hence the dot product is in the range $-dq^2$ to dq^2. Thus, we require $p \geq 2nd^\tau q^{2\tau+1} + 1$, where n is the total number of users, d is the dimension of the model updates, q is the quantization parameter and τ is the degree of the discriminator function.

V. LoByITFL

Considering the high communication overhead required by ByITFL, we propose LoByITFL, a Byzantine-resilient scheme with IT privacy and low communication cost. IT privacy is obtained by the use of threshold SecShare schemes. For clarity of exposition and without loss of generality, we consider an $(n, k = t + 1, t)$-SecShare scheme in the following, i.e., the vectors are not partitioned into sub-vectors and $m = 1$.

Before the training phase, we require an *initialization phase* with a TTP. The TTP generates a sufficient number of Beaver triples γ, ω, κ and sends the corresponding shares

[7]The case $\lambda = 0$ can be avoided by minor changes, omitted for brevity.

$\gamma[i], \omega[i], \kappa[i]$ to the users, enabling multiplication of secret-shared messages in the training phase. In addition, it also samples sufficiently many vectors $\mathbf{r}^1, \ldots, \in \mathbb{F}_p^d$ and values $\lambda^1, \ldots, \in \mathbb{F}$ independently and uniformly at random. The $\mathbf{r}$'s make the secret sharing step more efficient, as will be explained later. The random values λ's are to compute $\boldsymbol{\nu}$, as in ByITFL. We omit the current global iteration index ρ for clarity. The numbers of random values and Beaver triples required for the training phase will be given later. The TTP sends $\mathbf{r}_i$, $\mathbf{r}_j[i]$ for $j \in [n]$, and $\lambda[i]$ to user $i \in [n]$.

a) Message Authentication Code: To prevent Byzantine users from providing corrupt computation results during the protocol, the TTP also assigns a one-time MAC [48] for each secret share of the generated randomness. Suppose $s[i]$ is a secret share of a symbol $s \in \mathbb{F}_p$. A MAC takes a key pair (α, β) where α and β are drawn independently and uniformly at random from $\mathbb{F}_p$, and is computed as $\mathrm{MAC}_{\alpha,\beta}(s[i]) \triangleq \alpha \cdot s[i] + \beta$. In our scheme, we keep α globally fixed and sample a new β independently for each MAC. As a result, the MAC is additive homomorphic and linear operations f can be performed on it, e.g., $f(s[i])$. Consider two parties A (in our case, the user) and B (in our case, the federator). Party A holds the secret share $s[i]$ and the corresponding MAC $\mathrm{MAC}_{\alpha,\beta}(s[i])$. It computes the linear operation $f(s[i])$ and $f(\mathrm{MAC}_{\alpha,\beta}(s[i]))$, and sends the computations to party B. Party B holds the MAC key (α, β) and wants to verify whether $f(s[i])$ is correctly computed. Upon receiving $f(s[i])$ and $f(\mathrm{MAC}_{\alpha,\beta}(s[i]))$, party B computes $\mathrm{MAC}_{\alpha,\beta}(f(s[i]))$ and checks if it is consistent with $f(\mathrm{MAC}_{\alpha,\beta}(s[i]))$, which prevents the corrupted computations. Thus, at the initialization phase, the TTP sends all the MAC keys (α, β)'s to the federator, enabling integrity check of each computation on the secret shares. The federator marks users who do not pass the integrity check as Byzantine and exclude them from future computation. We omit α and β later in the text for brevity.

We denote $\{s[i]\}$ as the secret shares of s accompanied by the corresponding MAC, i.e. $\{s[i]\} = \{s[i], \mathrm{MAC}(s[i])\}$. Hence, after the initialization, each user $i \in [n]$ receives a sufficient number of $\mathbf{r}_i$, $\{\mathbf{r}_j[i]\}$ for $j \in [n]$, $\{\lambda[i]\}$, and Beaver triples $\{\gamma[i]\}$, $\{\omega[i]\}$, $\{\kappa[i]\}$, while the federator receives all MAC keys (α, β)'s used for generating the MACs of the random values.

The *training phase* of LoByITFL also consists of the aforementioned main steps, as in ByITFL.

A. Normalization and Quantization

Users and the federator compute, normalize and quantize their model updates as described in Section IV-A. The relation between p and q will be detailed later.

B. Sharing of the Normalized Model Updates

The federator broadcasts $\bar{\boldsymbol{u}}_0$ to all users. Each user $i \in [n]$ secret-shares $\bar{\boldsymbol{u}}_i$ to all other users with a threshold SecShare scheme, thus keeping it IT private while available for computations on the shares. The secret sharing of $\bar{\boldsymbol{u}}_i$ is supported by the uniformly random vector $\mathbf{r}_i$ distributed in the initialization phase. Specifically, user i knows the plain value $\mathbf{r}_i$ and each user $j \in [n]$ has $\mathbf{r}_i[j]$. User i broadcasts $\bar{\boldsymbol{u}}_i - \mathbf{r}_i$ to all other users and user j computes

$$\{\bar{\boldsymbol{u}}_i[j]\} = (\bar{\boldsymbol{u}}_i - \mathbf{r}_i) + \{\mathbf{r}_i[j]\}. \tag{7}$$

Note that this constitutes an $(n, t+1, t)$-threshold SecShare of $\bar{\boldsymbol{u}}_i$ among users $j \in [n]$ as detailed in the following. The uniformly random vector $\mathbf{r}_i$ is independent of $\bar{\boldsymbol{u}}_i$ and thus perfectly hides it by Shannon's one-time-pad [53]. Once $\bar{\boldsymbol{u}}_i - \mathbf{r}_i$ is public, gaining information about $\bar{\boldsymbol{u}}_i$ implies gaining information about $\mathbf{r}_i$, which is only possible if the adversary has access to more than t shares $\bar{\boldsymbol{u}}_i[j]$. Conversely, from any $t+1$ shares $\mathbf{r}_i[j]$, it is straightforward to decode $\mathbf{r}_i$ and thus $\bar{\boldsymbol{u}}_i$. The corresponding $\mathrm{MAC}(\bar{\boldsymbol{u}}_i[j])$ is obtained by the additive homomorphism of the MAC. To summarize, each user i obtains a secret share and the corresponding MAC of the local model update of every user $j \in [n]$, i.e. $\{\bar{\boldsymbol{u}}_j[i]\}$.

C. Validation of Normalization

To prevent Byzantine users from incorrectly normalizing their updates, validation is needed. Each user $i \in [n]$ computes $\{\|\bar{\boldsymbol{u}}_j\|_2^2[i]\} \in \mathbb{F}_p^d$ by consuming d Beaver triples, and sends $\{\|\bar{\boldsymbol{u}}_j\|_2^2[i]\}$ to the federator. The federator performs an integrity check on each $\|\bar{\boldsymbol{u}}_j\|_2^2[i]$, reconstructs $\|\bar{\boldsymbol{u}}_j\|_2^2$ and checks if it lies within a certain interval. As the integrity check ensures the correctness of all computations, no error correction decoding [46] is needed, which is expensive in computation and requires a higher total number of users for decoding. Hence, we only require $n \geq b + m + t + e$. Users who fail the normalization check are excluded from future computations.

D. Users Secure Computation

To compute the aggregation in Eq. (5), user i, having $\bar{\boldsymbol{u}}_0$, $\{\bar{\boldsymbol{u}}_j[i]\}$ for $j \in [n]$, and $\lambda[i]$, computes $\{(\lambda\Sigma_1)[i]\}$ and $\{(\lambda\boldsymbol{\Sigma}_2)[i]\}$ by consuming Beaver triples and sends the computation results to the federator.

E. Private Aggregation

By receiving $\{(\lambda\Sigma_1)[i]\}$'s and $\{(\lambda\boldsymbol{\Sigma}_2)[i]\}$'s from the users, the federator performs the integrity check to guarantee the correctness of the computation and reconstructs $\lambda\Sigma_1$ and $\lambda\boldsymbol{\Sigma}_2$ by Lagrange interpolation, as long as it receives at least $m+t$ correct computations. Thus, we require $n \geq b+m+t+e$. After decoding, the federator computes $\boldsymbol{\nu} = \frac{\lambda\boldsymbol{\Sigma}_2}{\lambda\Sigma_1}$, converts $\boldsymbol{\nu}$ to the real domain and follows Eq. (6) to compute the global model for the next iteration. Similarly, to guarantee the correctness of the reconstructed results, we require $p \geq 2nd^\tau q^{2\tau+1} + 1$ to avoid any wrap-around in the finite field.

Remark 1. *The number of random values and Beaver triples required for the training phase (and distributed during the initialization phase) is given in Table II.*

VI. Experiments

A. Choice of the Discriminator Function

In FLTrust [13], the ReLU function is chosen as the discriminator function, whose use in IT privacy poses challenges. Specifically, the ReLU function requires comparison, which is

TABLE II: The number of random numbers and Beaver triples required per training iteration. We consider three different versions of Beaver triples: *a)* γ, ω, κ are used for the multiplication between two scalars; *b)* $\boldsymbol{\iota}, \boldsymbol{\phi}, \chi$ are used for computing the dot product of two d/m-dimensional vectors, where $\boldsymbol{\iota}, \boldsymbol{\phi} \in \mathbb{F}_p^{d/m}$ and χ is the dot product of $\boldsymbol{\iota}$ and $\boldsymbol{\phi}$; and *c)* $\zeta, \boldsymbol{\xi}, \boldsymbol{\psi}$ are used for computing the element-wise multiplication of a scalar and a d/m-dimensional vector, where $\boldsymbol{\xi}, \boldsymbol{\psi} \in \mathbb{F}_p^{d/m}$ and $\boldsymbol{\psi}$ is the component-wise multiplication of ζ and $\boldsymbol{\xi}$. Each vector Beaver triple contains d/m scalar Beaver triples. The operations over vector Beaver triples can be easily generalized from scalar Beaver triples.

	Notation	Amount	Purpose (for $i \in [n]$)
Random value	$\mathbf{r}$	n	secret share $\bar{\boldsymbol{u}}_i$
	λ	1	hide Σ_1 and $\boldsymbol{\Sigma}_2$
Beaver triple	$\boldsymbol{\iota}, \boldsymbol{\phi}, \chi$	n	compute $\|\bar{\boldsymbol{u}}_i\|_2^2$
	γ, ω, κ	$(\tau-1)n+1$	compute $h(\langle \bar{\boldsymbol{u}}_0, \bar{\boldsymbol{u}}_i \rangle)$ and $\lambda\Sigma_1$
	$\zeta, \boldsymbol{\xi}, \boldsymbol{\psi}$	$n+1$	compute $h(\langle \bar{\boldsymbol{u}}_0, \bar{\boldsymbol{u}}_i \rangle) \cdot \bar{\boldsymbol{u}}_i$ and $\lambda\boldsymbol{\Sigma}_2$

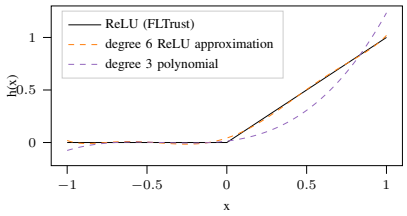

Fig. 3: Comparison of different discriminator functions $h(x)$.

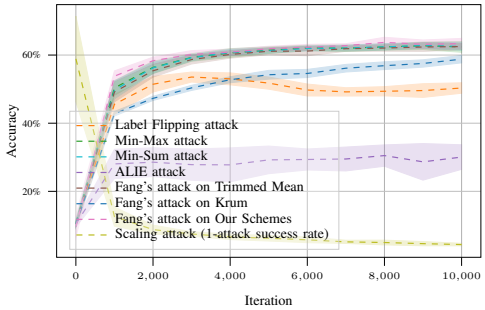

Fig. 4: Test accuracy (%) of our schemes on non-i.i.d CIFAR-10 over iterations. For scaling attack, we plot (1−attack success rate). Baseline is $63.9 \pm 0.6\%$.

difficult to compute under IT privacy guarantees. The value of the comparison must remain private, since it inherently leaks information about the relative direction of local updates with the federator update. For this reason, we replace the discriminator function by one that can be computed in a private manner. A candidate is a polynomial approximation of the ReLU [1]. However, while ReLU is plausible as a good discriminator function, there is no proof or principled reason to believe that the ReLU is the best discriminator function.

Therefore, we design our discriminator function to be a polynomial that satisfies the following core observation: The values of the discriminator function (TSs) for negative inputs need not be *exactly* zero; it is sufficient that updates in the opposite direction of the federator's update have small values. Further, even moderate negative values in the discriminator function for $x < 0$ cannot be exploited by the adversary. Consider some $x_0 > 0$ such that $h(-x_0) < 0 < h(x_0)$ and $|h(-x_0)| < |h(x_0)|$. Then, rather than sending a model update with cosine similarity $-x_0$ the adversary can always achieve a higher TS, and thus influence the aggregate more, by flipping the direction of the model update, achieving cosine similarity x_0. The direction of the model update weighted by the trust score is the same in either case. Hence, such corrupt updates cannot harm the learning process beyond what corrupt model updates with a positive cosine similarity are capable of.

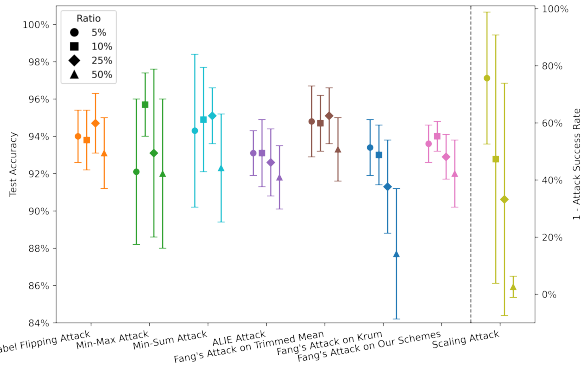

Fig. 5: Test accuracy of our schemes on MNIST with 40 users and i.i.d. setting. For scaling attack, we plot 1−attack success rate and separate it with a dashed line. Baseline is $96.2 \pm 0.1\%$.

Based on these observations, we find that it is not required to accurately approximate the ReLU function by high-degree polynomials as in [1], which is only satisfactory when the degree τ is greater than 6; instead, we carefully choose a degree-3 polynomial that mimics ReLU in the negative half. In the positive half, we choose a more conservative shape than ReLU by giving higher trust scores to local model updates that strongly point in the same direction as the federator model update. Uncertain local model updates are attenuated more aggressively. This is very similar to [34], where the authors use a quadratic function on the right, and a constant 0 on the left. As a by-product, the degree of our discriminator polynomial is decreased significantly to 3. The chosen degree-3 polynomial is given by $h(x) = 0.46897526\boldsymbol{x}^3 + 0.56578977\boldsymbol{x}^2 + 0.1860353\boldsymbol{x} + 0.01363545$, and plotted in Fig. 3 alongside with the degree-6 polynomial of [1] and the ReLU function.

B. Numerical Results

We demonstrate the convergence of our algorithms, using the above polynomial as a discriminator, and compare them to FedAvg and FLTrust. Our experiments are based on the implementation provided by [28]. We consider MNIST [54], Fashion MNIST [55] and CIFAR-10 [56] distributed across $n = 40$ users. On MNIST and Fashion MNIST, we train a three-layer dense neural network, and on CIFAR-10 a CNN model. The rectified linear unit function (ReLU) is used as the activation function. As loss function we choose Cross-Entropy. In each iteration, users randomly sample a minibatch of 64 samples from their local training dataset and perform local training on the minibatch. The learning rate for MNIST and Fashion MNIST is chosen to be 0.1, and 0.01 for CIFAR-10.

TABLE III: Test accuracy (%) of FLTrust [13] and our proposed schemes (ByITFL and LoByITFL) on MNIST, Fashion MNIST and CIFAR-10 in i.i.d. and non-i.i.d. setting. For scaling attack, we give both test accuracy and attack success rate (%). In the i.i.d. setting, baseline for MNIST is $96.2 \pm 0.1\%$, for Fashion MNIST is $88.6 \pm 0.1\%$, and for CIFAR-10 is $65.9 \pm 0.5\%$. In the non-i.i.d. setting, baseline is $96.2 \pm 0.1\%$, $88.6 \pm 0.2\%$, and $63.9 \pm 0.6\%$, respectively.

Attack Type	Algorithm	Dataset (i.i.d.)			Dataset (non-i.i.d.)		
		MNIST	Fashion MNIST	CIFAR-10	MNIST	Fashion MNIST	CIFAR-10
Label Flipping Attack	FLTrust	93.3 ± 0.8	87.6 ± 0.3	$\mathbf{60.4 \pm 0.9}$	92.8 ± 0.5	86.8 ± 0.4	$\mathbf{56.5 \pm 1.6}$
	proposed	$\mathbf{94.7 \pm 1.6}$	$\mathbf{88.5 \pm 0.3}$	53.1 ± 1.5	$\mathbf{94.1 \pm 1.3}$	$\mathbf{88.1 \pm 0.4}$	50.3 ± 1.5
Min-Max Attack	FLTrust	87.6 ± 8.5	86.5 ± 1.2	56.3 ± 13.9	87.5 ± 5.9	83.5 ± 3.1	59.9 ± 0.8
	proposed	$\mathbf{93.1 \pm 4.5}$	$\mathbf{88.7 \pm 0.3}$	$\mathbf{65.6 \pm 0.7}$	$\mathbf{95.1 \pm 1.6}$	$\mathbf{88.3 \pm 0.3}$	$\mathbf{62.4 \pm 1.6}$
Min-Sum Attack	FLTrust	88.2 ± 7.4	84.6 ± 3.9	60.3 ± 8.8	85.2 ± 7.3	84.8 ± 3.1	59.4 ± 0.9
	proposed	$\mathbf{95.1 \pm 1.5}$	$\mathbf{88.9 \pm 0.2}$	$\mathbf{65.0 \pm 1.7}$	$\mathbf{95.6 \pm 0.8}$	$\mathbf{88.5 \pm 0.4}$	$\mathbf{62.5 \pm 1.1}$
ALIE Attack	FLTrust	$\mathbf{92.7 \pm 0.8}$	$\mathbf{66.4 \pm 23.2}$	20.9 ± 12.0	92.6 ± 0.8	$\mathbf{42.4 \pm 16.0}$	25.0 ± 8.4
	proposed	92.6 ± 1.8	32.7 ± 3.8	$\mathbf{27.1 \pm 12.2}$	$\mathbf{92.8 \pm 1.7}$	28.9 ± 7.3	$\mathbf{30.1 \pm 3.6}$
Fang's Attack on Trimmed Mean	FLTrust	89.1 ± 4.1	84.2 ± 2.0	62.9 ± 0.7	91.6 ± 1.3	85.3 ± 1.5	59.8 ± 1.0
	proposed	$\mathbf{95.1 \pm 1.5}$	$\mathbf{86.7 \pm 3.6}$	$\mathbf{64.9 \pm 0.8}$	$\mathbf{92.9 \pm 4.1}$	$\mathbf{86.9 \pm 1.8}$	$\mathbf{62.4 \pm 0.9}$
Fang's Attack on Krum	FLTrust	$\mathbf{92.9 \pm 0.5}$	$\mathbf{86.8 \pm 0.6}$	$\mathbf{62.9 \pm 0.8}$	$\mathbf{93.4 \pm 0.4}$	86.4 ± 0.5	$\mathbf{59.1 \pm 1.2}$
	proposed	91.3 ± 2.5	86.7 ± 1.0	$62.7 \pm 0.9^\star$	91.6 ± 1.8	$\mathbf{86.8 \pm 0.6}$	$58.8 \pm 1.2^\star$
Fang's Attack on FLTrust/Our Schemes	FLTrust	90.8 ± 1.2	87.5 ± 0.3	62.2 ± 1.4	91.9 ± 2.0	87.0 ± 0.3	60.2 ± 1.3
	proposed	$\mathbf{92.9 \pm 1.2}$	$\mathbf{88.5 \pm 0.2}$	$\mathbf{65.6 \pm 1.8}$	$\mathbf{93.8 \pm 1.4}$	$\mathbf{88.6 \pm 0.3}$	$\mathbf{63.4 \pm 1.5}$
Scaling Attack (Test Accuracy)	FLTrust	93.0 ± 1.1	87.6 ± 0.3	65.2 ± 0.9	93.3 ± 0.9	87.2 ± 0.6	62.0 ± 1.2
	proposed	93.7 ± 1.6	89.0 ± 0.3	68.4 ± 0.9	93.5 ± 1.3	88.9 ± 0.3	66.4 ± 1.0
Scaling Attack (Attack Success Rate)	FLTrust	$\mathbf{61.7 \pm 36.6}$	99.9 ± 0.1	$\mathbf{92.0 \pm 1.1}$	$\mathbf{67.1 \pm 32.1}$	$\mathbf{99.8 \pm 0.1}$	$\mathbf{91.9 \pm 0.8}$
	proposed	66.8 ± 40.7	99.9 ± 0.1	95.2 ± 0.6	86.3 ± 27.0	99.9 ± 0.1	95.6 ± 0.4

* Results marked with a star are from a slightly modified version of our proposed scheme, i.e., not using Eq. (6), but using $\boldsymbol{\nu}$ directly as the model update. Using Eq. (6) causes the test accuracy to drop to $17.8 \pm 1.9\%$ and $17.9 \pm 2.9\%$ for i.i.d. and non-i.i.d. settings, respectively.

We assume 25% of the users are Byzantine ($b = 10$) and perform the Label Flipping attack, Min-Max attack [57], Min-Sum attack [57], a-little-is-enough (ALIE) attack [58], Fang's attack [59] on Trimmed Mean [11], Krum [10] and different discriminator functions, and Scaling attack. The label flipping attack follows the same setting as in [59]. Fang's attack [59] is an untargeted local model poisoning attack, optimized for different robust aggregation rules, such as Krum and Trimmed Mean. The scaling attack is equivalently known as the backdoor attack [60]. For scaling attack, we use attack success rate, i.e., the fraction of the attacker's target testing samples being predicted as the attacker's chosen label, to measure the performance of the attack, rather than using test accuracy. A lower attack success rate means a more robust defense. As in [13], we randomly split the users into 10 groups and a training example with label j is assigned to group j with probability $a > 0$ and to any other groups with probability $\frac{1-a}{9}$. Data are uniformly distributed to each user within the same group. Therefore, we set $a = 0.1$ for i.i.d. setting and set $a = 0.5$ for non-i.i.d. setting. We set $q = 1024$, $|D_0| = 100$ as in [13], and $\varepsilon = 0.02$ for the normalization validation.

The performance of our schemes using the aforementioned degree 3 polynomial on CIFAR-10 under different attacks is shown in Fig. 4, and the experiment results presented in Table III show that our schemes are comparable to FLTrust. This demonstrates that our approach maintains the original performance of the Byzantine resilience while incorporating IT privacy. Note that, for CIFAR-10, we found that a slight adjustment to our schemes significantly improved the resilience against Fang's attack on Krum. We skip the normalization and flipping step in Eq. (6), and directly use $\boldsymbol{\nu}$ as the model update, which is exactly the algorithm proposed in our previous work [2]. This variation is minor and does not affect generalizability, and we report the results of this variation in Fig. 4 and Table III. Fig. 5 shows the performance of our schemes under attacks with different ratios of Byzantine users.

VII. Conclusion

We proposed two robust aggregation schemes for FL that guarantee IT privacy. We require the federator to hold a root dataset for reference, used to scale the users' local updates during aggregation. To achieve IT privacy, we used a suitable polynomial as the discriminator function to compute trust scores and leverage secret sharing techniques to ensure privacy. We provided theoretical guarantees on privacy, resilience, convergence, and algorithmic complexity, along with empirical validation demonstrating convergence even in adversarial settings. While our primary contribution is to bring IT privacy with Byzantine resilience, we note that Byzantine resilience approaches based on cosine-similarity are still not fully understood, and a systematic and careful study remains an open problem. Consequently, the cosine-similarity may have additional weaknesses beyond the known weakness to backdoor attacks [34]. Establishing meaningful theoretical guarantees for robust aggregation with less restrictive assumptions, is yet unsolved. Future work includes extending our approach to wireless FL environments, as explored in [61], [62].

Appendix

A. Proof for Theorem 1

Proof. We start by proving the IT privacy of ByITFL.

1) IT Privacy: We first prove the IT privacy against any t colluding users and the federator according to Eq. (2):

$$\begin{aligned} &I(\boldsymbol{u}_{\mathcal{H}}; M_{\mathcal{T}}, M_0 \,|\, \boldsymbol{u}_{\mathcal{T}}, \boldsymbol{u}_0, D_{\mathcal{T}}, D_0, \boldsymbol{w}, \boldsymbol{\nu}) \\ =&H(\boldsymbol{u}_{\mathcal{H}} \,|\, \boldsymbol{u}_{\mathcal{T}}, \boldsymbol{u}_0, D_{\mathcal{T}}, D_0, \boldsymbol{w}, \boldsymbol{\nu}) \\ &- H(\boldsymbol{u}_{\mathcal{H}} \,|\, M_{\mathcal{T}}, M_0, \boldsymbol{u}_{\mathcal{T}}, \boldsymbol{u}_0, D_{\mathcal{T}}, D_0, \boldsymbol{w}, \boldsymbol{\nu}) \\ =&H(\boldsymbol{u}_{\mathcal{H}}|D_{\mathcal{T}},D_0,\boldsymbol{w},\boldsymbol{\nu}) - H(\boldsymbol{u}_{\mathcal{H}}|M_{\mathcal{T}},M_0,D_{\mathcal{T}},D_0,\boldsymbol{w},\boldsymbol{\nu}), \quad (8) \end{aligned}$$

where the last equation follows because $\boldsymbol{u}_{\mathcal{T}}$ is a deterministic function of $D_{\mathcal{T}}$ and $\boldsymbol{w}$, and $\boldsymbol{u}_0$ is a deterministic function of D_0 and $\boldsymbol{w}$. We then consider the exchanged messages $M_{\mathcal{T}}$ observed by the colluding users and M_0 observed by the federator in each step. $M_{\mathcal{T}}$ includes:

Step A and E: nothing, since step A only involves local computations and step E is performed by the federator;

Step B: shares $\bar{\boldsymbol{u}}_j[i]$ for $i \in \mathcal{T}, j \in [n]$;

Step C: $\|\bar{\boldsymbol{u}}_j\|_2^2[i]$ for $i \in \mathcal{T}, j \in [n]$ and sub-shares from the re-randomization when computing $\|\bar{\boldsymbol{u}}_j\|_2^2[i]$; and

Step D: shares $\Sigma_1[i], \boldsymbol{\Sigma}_2[i]$, $(\lambda\Sigma_1)[i]$, $(\lambda\boldsymbol{\Sigma}_2)[i]$, the random values λ_i, and $\lambda_j[i]$, for $j \in [n]$ and $i \in \mathcal{T}$, and sub-shares from the re-randomization step when computing shares of $\Sigma_1, \boldsymbol{\Sigma}_2, \lambda\Sigma_1$ and $\lambda\boldsymbol{\Sigma}_2$.

With regard to M_0, we need to consider:

Step A and B: nothing, since step A and B only involve computations among the users; and

Step C, D and E: shares $\|\bar{\boldsymbol{u}}_j\|_2^2[i]$, for $j \in [n]$, $(\lambda\Sigma_1)[i]$ and $(\lambda\boldsymbol{\Sigma}_2)[i]$ received from the users for $i \in [n]$, and the reconstruction of $\|\bar{\boldsymbol{u}}_j\|_2^2$, $\lambda\Sigma_1$ and $\lambda\boldsymbol{\Sigma}_2$.

Regarding $M_{\mathcal{T}}$, we leverage the privacy guarantees of LCC [47], the re-randomization step [49], [50] and ITVSS [52]. Since each secret-sharing is encoded by a degree $m+t-1$ polynomial, when considering at most t colluding users, the shares and the sub-shares observed by the colluding users are completely random and independent of $\boldsymbol{u}_{\mathcal{H}}, D_{\mathcal{T}}, D_0, \boldsymbol{w}$ and $\boldsymbol{\nu}$. Since the random values are drawn uniformly and independently from $\mathbb{F}_p$, λ_i's are independent of $\boldsymbol{u}_{\mathcal{H}}, M_0, D_{\mathcal{T}}, D_0, \boldsymbol{w}$ and $\boldsymbol{\nu}$. However, the shares $\|\bar{\boldsymbol{u}}_j\|_2^2[i]$, $(\lambda\Sigma_1)[i]$ and $(\lambda\boldsymbol{\Sigma}_2)[i]$ for $i \in \mathcal{T}, j \in [n]$ are not independent of M_0, rather, M_0 includes these shares. We denote these shares as $\hat{M}_{\mathcal{T}}$, thus, we obtain $H(\hat{M}_{\mathcal{T}} \,|\, M_0) = 0$. Hence,

$$\begin{aligned} H(\boldsymbol{u}_{\mathcal{H}}|M_{\mathcal{T}},M_0,D_{\mathcal{T}},D_0,\boldsymbol{w},\boldsymbol{\nu}) &= H(\boldsymbol{u}_{\mathcal{H}}|\hat{M}_{\mathcal{T}},M_0,D_{\mathcal{T}},D_0,\boldsymbol{w},\boldsymbol{\nu}) \\ &= H(\boldsymbol{u}_{\mathcal{H}}|M_0,D_{\mathcal{T}},D_0,\boldsymbol{w},\boldsymbol{\nu}), \end{aligned} \tag{9}$$

where the first equality holds because of the independence, and the second equality holds because $H(\hat{M}_{\mathcal{T}} \mid M_0) = 0$.

As for M_0, since the collection of all the shares from user $i \in [n]$ is informationally equivalent to the secret itself, we only need to consider the privacy of the reconstructions. Among the reconstructions: (*a*) $\|\bar{\boldsymbol{u}}_i\|_2^2$ lies within a certain range for all possible model updates (ideally equivalent to one); (*b*) Σ_1 is completely hidden due to the randomness λ; and (*c*) no information is leaked beyond $\boldsymbol{\nu} = \frac{\boldsymbol{\Sigma}_2}{\Sigma_1}$, i.e., $H(\lambda\boldsymbol{\Sigma}_2|\boldsymbol{\nu}) = 0$. We denote all the exchanged messages except $\lambda\boldsymbol{\Sigma}_2$ as $\hat{M}_0$, thus,

$$\begin{aligned} H(\boldsymbol{u}_{\mathcal{H}}|M_0, D_{\mathcal{T}}, D_0, \boldsymbol{w}, \boldsymbol{\nu}) &= H(\boldsymbol{u}_{\mathcal{H}}|\hat{M}_0, \lambda\boldsymbol{\Sigma}_2, D_{\mathcal{T}}, D_0, \boldsymbol{w}, \boldsymbol{\nu}) \\ &= H(\boldsymbol{u}_{\mathcal{H}}|D_{\mathcal{T}}, D_0, \boldsymbol{w}, \boldsymbol{\nu}), \end{aligned}$$

where the second equation holds since $\hat{M}_0$ is completely random and independent of other terms, and $H(\lambda\boldsymbol{\Sigma}_2|\boldsymbol{\nu}) = 0$. By substituting the above into Eq. (8) we prove Eq. (2), showing that ByITFL is IT private against t colluding users and the federator.

2) Resilience against Byzantine and Robustness against dropout: Byzantine users may present corrupt updates, and perform the required computations dishonestly. The effect of corrupt updates is mitigated by the robust aggregation rule, i.e. FLTrust [13]. Next, we guarantee the correct computation of $\lambda\Sigma_1[i]$ and $\lambda\boldsymbol{\Sigma}_2[i]$. For dishonest computations, Byzantine users may behave arbitrarily in any step during the computation. Particularly, user $i \in \mathcal{B}$ can: (*a*) incorrectly normalize local model updates $\tilde{\boldsymbol{u}}_i$ in the normalization step; (*b*) distribute invalid secret shares, i.e. secret shares encoded using different encoding polynomials, in the secret sharing step; and, (*c*) misbehave when computing $\|\bar{\boldsymbol{u}}_j\|_2^2[i]$, $(\lambda\Sigma_1)[i]$ and $(\lambda\boldsymbol{\Sigma}_2)[i]$, and send corrupt results to the federator.

Byzantine behavior in the first scenario can be detected and identified in the normalization validation step. For all honest users, the model updates presented and normalized lie within a certain range. Thus, even if malicious users present model updates with the largest squared l_2-norm that will be accepted by the normalization validation, it only has limited impact since ϵ is empirically set to be very small. For the second scenario, the attack can be detected by ITVSS [52], thus guaranteeing the correctness of the secret sharing scheme.

With regard to the third scenario, the correctness is ensured by the Reed-Solomon (RS) decoding algorithm [46]. As described in Section IV-C and Section IV-E, $\|\bar{\boldsymbol{u}}_j\|_2^2[i]$, $(\lambda\Sigma_1)[i]$ and $(\lambda\boldsymbol{\Sigma}_2)[i]$ can be viewed as evaluations of the polynomials with degree $2(m+t-1)$, $(\tau+1)\cdot(m+t-1)$ and $(\tau+2)\cdot(m+t-1)$, respectively. The decoding of $\|\bar{\boldsymbol{u}}_j\|_2^2$, $\lambda\Sigma_1$ and $\lambda\boldsymbol{\Sigma}_2$ can be treated as the decoding of Reed-Solomon codes with parameters

1) $[n, 2(m+t-1)+1, n-2(m+t-1)]_p$;
2) $[n, (\tau+1)\cdot(m+t-1)+1, n-(\tau+1)\cdot(m+t-1)]_p$;
3) $[n, (\tau+2)\cdot(m+t-1)+1, n-(\tau+2)\cdot(m+t-1)]_p$;

respectively. Note that, a $[n, k, n-k+1]_p$ RS code with e erasures is able to decode $\lfloor \frac{n-k-e}{2} \rfloor$ errors. By viewing the updates presented by Byzantine users as errors in a RS code and dropout users as erasures, we obtain three requirements on n given by the decoding of these three RS codes:

1) $b \leq \frac{n-(2(m+t-1)+1)-e}{2}$;
2) $b \leq \frac{n-((\tau+1)\cdot(m+t-1)+1)-e}{2}$;
3) $b \leq \frac{n-((\tau+2)\cdot(m+t-1)+1)-e}{2}$;

Hence, the federator can decode the correct computation results as long as $n \geq 2b+(\tau+2)\cdot(m+t-1)+e+1$, which is the largest n required by these three decodings.

3) Complexity: For each user sharing a single scalar, the computation complexity of encoding in LCC is $O(n\log^2 n \log\log n)$ [47] and that of ITVSS [52] is $O(n^2\log^2 n)$. Since the re-randomization step [49] involves sub-sharing each secret share using ITVSS and a linear combination of the sub-shares, the computation complexity is $O(n^2\log^2 n\log\log n + n^3\log^2 n)$. The communication cost for the ITVSS is $O(n^2)$, and $O(n^3)$ for re-randomization. With respect to the federator, the computation cost for decoding a single scalar using error-correction decoding takes $O(n\log^2 n\log\log n)$. The remaining proof follows from counting, here omitted for brevity. □

B. Proof for Theorem 2

Proof. We begin with the proof for IT privacy for LoByITFL.

1) IT Privacy: For privacy against any t colluding users and the federator according to Eq. (2), we want to prove Eq. (8). The exchanged messages $M_{\mathcal{T}}$ are:

Initialization Step: random vectors $\mathbf{r}_i$'s, shares of random values $\{\mathbf{r}_j[i]\}$'s for $j \in [n]$ and $\{\lambda[i]\}$'s, shares of Beaver triple $\{\gamma[i]\}$'s, $\{\omega[i]\}$'s, $\{\kappa[i]\}$'s, for $i \in \mathcal{T}$;

Step A and E: nothing, since step A only involves local computations and step E is performed by the federator;

Step B: the broadcasted values $\bar{\boldsymbol{u}}_j - \mathbf{r}_j$ and the shares $\{\bar{\boldsymbol{u}}_j[i]\}$, for $i \in \mathcal{T}, j \in [n]$;

Step C: shares $\{\|\bar{\boldsymbol{u}}_j\|_2^2[i]\}$ and the messages incurred when consuming Beaver triples to compute $\{\|\bar{\boldsymbol{u}}_j\|_2^2[i]\}$, for $i \in \mathcal{T}, j \in [n]$; and

Step D: shares $\{\Sigma_1[i]\}, \{\boldsymbol{\Sigma}_2[i]\}$, $\{(\lambda\Sigma_1)[i]\}$, $\{(\lambda\boldsymbol{\Sigma}_2)[i]\}$, and $\{\lambda_j[i]\}$ for $j \in [n]$ and $i \in \mathcal{T}$, and the messages incurred when consuming Beaver triples to compute shares of $\Sigma_1, \boldsymbol{\Sigma}_2, \lambda\Sigma_1$ and $\lambda\boldsymbol{\Sigma}_2$.

The exchanged messages M_0 include:

Initialization Step: the MAC keys used for generating the MACs of the secret shares;

Step A and B: nothing, since step A and B only involve computations among the users; and

Step C, D and E: shares $\{\|\bar{\boldsymbol{u}}_j\|_2^2[i]\}$ for $j \in [n]$, $\{(\lambda\Sigma_1)[i]\}$ and $\{(\lambda\boldsymbol{\Sigma}_2)[i]\}$ received from the user $i \in [n]$, the reconstruction of $\|\bar{\boldsymbol{u}}_j\|_2^2$, $\lambda\Sigma_1$ and $\lambda\boldsymbol{\Sigma}_2$, and the messages incurred when using Beaver triples to compute multiplications.

Note that, when computing $\{(sv)[i]\}$, colluding users get $\{s[i] - \gamma[i]\}$, $\{v[i] - \omega[i]\}$, $s - \gamma$ and $v - \omega$, for $i \in \mathcal{T}$, and the federator gets the values of $\{s[i] - \gamma[i]\}$, $\{v[i] - \omega[i]\}$ for $i \in [n]$ and the reconstructed values $s - \gamma$ and $v - \omega$.

We leverage the privacy guarantees of the SecShare scheme, the Beaver triple and the MAC scheme. Regarding the messages $M_{\mathcal{T}}$, when the number of colluding users is smaller than the threshold of the secret sharing scheme, i.e. t, we have: (*a*) the shares observed by the colluding users are completely random and independent of $\boldsymbol{u}_{\mathcal{H}}, D_{\mathcal{T}}, D_0, \boldsymbol{w}$ and $\boldsymbol{\nu}$, but partially included by M_0; (*b*) the random vectors $\mathbf{r}$'s are random and independent of $M_0, \boldsymbol{u}_{\mathcal{H}}, D_{\mathcal{T}}, D_0, \boldsymbol{w}$ and $\boldsymbol{\nu}$; (*c*) the broadcasted values $\bar{\boldsymbol{u}}_j - \mathbf{r}_j$'s are random because of one-time padding; (*d*) the MACs are also random because of the randomness of β; and (*e*) the messages incurred when consuming Beaver triples for multiplication are either secret shares or protected by one-time padding, thus, independent of $\boldsymbol{u}_{\mathcal{H}}, M_0, D_{\mathcal{T}}, D_0, \boldsymbol{w}$ and $\boldsymbol{\nu}$. Thus, we have Eq. (9) when considering the shares $\{\|\bar{\boldsymbol{u}}_j\|_2^2[i]\}$, $\{(\lambda\Sigma_1)[i]\}$ and $\{(\lambda\boldsymbol{\Sigma}_2)[i]\}$ for $j \in [n]$ and $i \in \mathcal{T}$ as $M_{\mathcal{T}}$.

As for M_0, the MAC keys sent from the TTP and the messages broadcasted during multiplication are random values that are independent of $\boldsymbol{u}_{\mathcal{H}}, D_{\mathcal{T}}, D_0, \boldsymbol{w}$ and $\boldsymbol{\nu}$. By following the same reasoning as in Appendix A1, we conclude the IT privacy proof for LoByITFL.

2) Resilience against Byzantine and Robustness against dropout: Byzantine users may present corrupt updates, and perform the required computations dishonestly. Same as in Appendix A2, the effect of corrupt updates is eliminated by the robust aggregation rule. We next guarantee the correct computation of $\lambda\Sigma_1[i]$ and $\lambda\boldsymbol{\Sigma}_2[i]$. For dishonest computations, Byzantine users $i \in \mathcal{B}$ may: (*a*) incorrectly normalize local model updates $\tilde{\boldsymbol{u}}_i$; (*b*) compute invalid secret shares $\{\bar{\boldsymbol{u}}_j[i]\}$ for any $j \in [n]$; and, (*c*) misbehave during the computation of $\{\|\bar{\boldsymbol{u}}_j\|_2^2[i]\}$, $\{(\lambda\Sigma_1)[i]\}$ and $\{(\lambda\boldsymbol{\Sigma}_2)[i]\}$.

As stated in Appendix A2, Byzantine attacks in the first scenario can be detected and identified in the normalization validation step. For Byzantine attacks in the remaining scenarios, the MACs are used to enable integrity check, i.e., for each linear operation performed on the secret shares, the users also perform the same computation on the corresponding MACs. Since the federator receives all the corresponding MAC keys (α, β)'s in the initialization phase, integrity check can be performed by the federator to guarantee the correctness of the computation. Therefore, the correctness of the computations entirely relies on the integrity check of the IT one-time MAC [48], where we need to consider the forgery probability of the MAC. Forgery probability is the probability that a malicious user guesses the MAC key used for generating the MAC of a specific secret share correctly, such that it can create a valid MAC for the computation result to fool the integrity check. Since in all MAC keys (α, β)s', α is a uniformly random consistent value and β's are chosen independently uniformly at random from the underlying prime field $\mathbb{F}_p$, which are used to protect the value of α when we reusing it in different MACs. The forgery probability for each scalar is the probability of guessing the β used for generating the MAC for a specific scalar, that is $1/\|\mathbb{F}_p\| = 1/p$, i.e. the inverse of the size of the underlying finite field. Since the finite field we choose is very large, i.e. $p \geq 2nd^{\tau}q^{2\tau+1}+1$, and the malicious users have to guess the β's used for all d dimensions to fool the integrity check, the forgery probability is small enough to guarantee the security and correctness of the scheme.

3) Complexity: In the training phase, the computation for each user sharing a scalar is, according to Eq. (7), the addition of a publicly known value and a secret share, thus $O(1)$, and that of one share multiplication is $O(1)$ as well by consuming one Beaver triple, which only involves addition and scaling of the secret shares. For the federator, the computation is to reconstruct the intermediate results during share multiplications and the final results, which is a Lagrange polynomial interpolation problem and costs $O(n \log^2 n \log\log n)$ per scalar. The communication cost for secret sharing and computations on secret shares are $O(1)$ for each user and $O(n)$ for the federator per scalar. The rest of the proof follows by counting, and is omitted here for the sake of brevity. □

C. Proof for convergence

For the proof of convergence, we need Lemma 1.

Lemma 1. *At each global iteration ρ, the distance between the federator's aggregation of the gradients $\boldsymbol{u}^{(\rho)}$ and the gradient $\nabla F(\boldsymbol{w}^{(\rho)})$ is bounded from above as:*

$$\|\boldsymbol{u}^{(\rho)} - \nabla F(\boldsymbol{w}^{(\rho)})\| \leq 3\|\boldsymbol{u}_0^{(\rho)} - \nabla F(\boldsymbol{w}^{(\rho)})\| + 2\|\nabla F(\boldsymbol{w}^{(\rho)})\|.$$

Proof. We omit the superscript (ρ) for ease of notation. Recall the definition of $\boldsymbol{\nu}$

$$\boldsymbol{\nu} \triangleq \frac{1}{\sum_{i\in[n]} h(\langle \bar{\boldsymbol{u}}_0, \bar{\boldsymbol{u}}_i \rangle)} \sum_{i\in[n]} h(\langle \bar{\boldsymbol{u}}_0, \bar{\boldsymbol{u}}_i \rangle) \cdot \bar{\boldsymbol{u}}_i.$$

Therefore, since Eq. (6), $\langle \boldsymbol{u}, \boldsymbol{u}_0 \rangle \geq 0$ always holds. We can bound $\|\boldsymbol{u} - \nabla F(\boldsymbol{w})\|$ as

$$\begin{aligned}
\|\boldsymbol{u} - \nabla F(\boldsymbol{w})\| &= \|\boldsymbol{u} - \boldsymbol{u}_0 + \boldsymbol{u}_0 - \nabla F(\boldsymbol{w})\| \\
&\overset{(a)}{\leq} \|\boldsymbol{u} - \boldsymbol{u}_0\| + \|\boldsymbol{u}_0 - \nabla F(\boldsymbol{w})\| \\
&\overset{(b)}{\leq} \|\boldsymbol{u} + \boldsymbol{u}_0\| + \|\boldsymbol{u}_0 - \nabla F(\boldsymbol{w})\| \\
&\overset{(c)}{\leq} \|\boldsymbol{u}\| + \|\boldsymbol{u}_0\| + \|\boldsymbol{u}_0 - \nabla F(\boldsymbol{w})\| \\
&= 2\|\boldsymbol{u}_0\| + \|\boldsymbol{u}_0 - \nabla F(\boldsymbol{w})\| \\
&= 2\|\boldsymbol{u}_0 - \nabla F(\boldsymbol{w}) + \nabla F(\boldsymbol{w})\| \\
&\quad + \|\boldsymbol{u}_0 - \nabla F(\boldsymbol{w})\| \\
&\overset{(d)}{\leq} 3\|\boldsymbol{u}_0 - \nabla F(\boldsymbol{w})\| + 2\|\nabla F(\boldsymbol{w})\|,
\end{aligned}$$

where (a), (b), and (d) follow from the triangle inequality, and (c) follows since $\langle \boldsymbol{u}, \boldsymbol{u}_0 \rangle \geq 0$. □

Using Lemma 1 and the same assumptions as in [13, Assumptions 1, 2 and 3], the proof of convergence of ByITFL follows the same steps as the proofs in Appendix A of [13].

REFERENCES

[1] Y. Xia, C. Hofmeister, M. Egger, and R. Bitar, "Byzantine-resilient secure aggregation for federated learning without privacy compromises," in *Proc. IEEE ITW*, 2024.

[2] ——, "Lobyitfl: Low communication secure and private federated learning," *Proc. FL-AsiaCCS*, 2025.

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

[4] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *Adv. Neural Inf. Process. Syst.*, 2019.

[5] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients: How easy is it to break privacy in federated learning?" *Adv. Neural Inf. Process. Syst.*, 2020.

[6] K. Bonawitz, V. Ivanov, B. Kreuter *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM CCS*, 2017.

[7] 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*, 2018.

[8] P. Kairouz, H. B. McMahan, B. Avent *et al.*, "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, 2021.

[9] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, 1982.

[10] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," *Adv. Neural Inf. Process. Syst.*, 2017.

[11] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. ICML*, 2018.

[12] R. Guerraoui, S. Rouault *et al.*, "The hidden vulnerability of distributed learning in byzantium," in *Proc. ICML*, 2018.

[13] X. Cao, M. Fang, J. Liu, and N. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," in *Proc. NDSS*, 2021.

[14] B. Zhao, P. Sun, T. Wang, and K. Jiang, "Fedinv: Byzantine-robust federated learning by inversing local model updates," in *Proc. AAAI*, 2022.

[15] R. Guerraoui, N. Gupta, and R. Pinot, *Robust Machine Learning*. Springer, 2024.

[16] R. K. Velicheti, D. Xia, and O. Koyejo, "Secure byzantine-robust distributed learning via clustering," *arXiv preprint arXiv:2110.02940*, 2021.

[17] M. Xhemrishi, J. Östman, A. Wachter-Zeh, and A. G. i Amat, "Fedgt: Identification of malicious clients in federated learning with secure aggregation," *IEEE Trans. Inf. Forensics Security*, 2025.

[18] L. He, S. P. Karimireddy, and M. Jaggi, "Secure byzantine-robust machine learning," *arXiv preprint arXiv:2006.04747*, 2020.

[19] M. Hao, H. Li, G. Xu, H. Chen, and T. Zhang, "Efficient, private and robust federated learning," in *Proc. ACSAC*, 2021.

[20] Z. Ma, J. Ma, Y. Miao, Y. Li, and R. H. Deng, "Shieldfl: Mitigating model poisoning attacks in privacy-preserving federated learning," *IEEE Trans. Inf. Forensics Security*, 2022.

[21] G. Xu, H. Li, Y. Zhang, S. Xu, J. Ning, and R. H. Deng, "Privacy-preserving federated deep learning with irregular users," *IEEE Trans. Dependable Secure Comput.*, 2022.

[22] Y. Zhong, W. Tan, Z. Xu, S. Chen, J. Weng, and J. Weng, "Wvfl: Weighted verifiable secure aggregation in federated learning," *IEEE Internet Things J.*, 2024.

[23] Y. Dong, X. Chen, K. Li, D. Wang, and S. Zeng, "Flod: Oblivious defender for private byzantine-robust federated learning with dishonest-majority," in *Proc. ESORICS*, 2021.

[24] Y. Miao, X. Yan, X. Li, S. Xu, X. Liu, H. Li, and R. H. Deng, "Rfed: Robustness-enhanced privacy-preserving federated learning against poisoning attack," *IEEE Trans. Inf. Forensics Security*, 2024.

[25] J. So, B. Güler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," *IEEE J. Sel. Areas Commun.*, 2020.

[26] T. Jahani-Nezhad, M. A. Maddah-Ali, and G. Caire, "Byzantine-resistant secure aggregation for federated learning based on coded computing and vector commitment," *arXiv preprints*, 2023.

[27] A. Roy Chowdhury, C. Guo, S. Jha, and L. van der Maaten, "Eiffel: Ensuring integrity for federated learning," in *Proc. ACM CCS*, 2022.

[28] T. Gehlhar, F. Marx, T. Schneider, A. Suresh, T. Wehrle, and H. Yalame, "Safefl: Mpc-friendly framework for private and robust federated learning," 2023.

[29] H. Fereidooni, S. Marchal, M. Miettinen *et al.*, "Safelearn: Secure aggregation for private federated learning," in *IEEE Security and Privacy Workshops (SPW)*, 2021.

[30] C. Dong, J. Weng, M. Li *et al.*, "Privacy-preserving and byzantine-robust federated learning," *IEEE Trans. Dependable Secure Comput.*, 2023.

[31] Y. Ben-Itzhak, H. Möllering, B. Pinkas *et al.*, "Scionfl: Efficient and robust secure quantized aggregation," in *Proc. IEEE SaTML*, 2024.

[32] G. Hu, H. Li, T. Wu, W. Fan, and Y. Zhang, "Efficient byzantine-robust and privacy-preserving federated learning on compressive domain," *IEEE Internet Things J.*, 2023.

[33] Z. Zhang and Y. Li, "Nspfl: A novel secure and privacy-preserving federated learning with data integrity auditing," *IEEE Trans. Inf. Forensics Security*, 2024.

[34] Z. Lu, S. Lu, X. Tang, and J. Wu, "Robust and verifiable privacy federated learning," *IEEE Trans. Artif. Intell.*, 2023.

[35] H. Lycklama, L. Burkhalter, A. Viand, N. Küchler, and A. Hithnawi, "Rofl: Robustness of secure federated learning," in *Proc. IEEE S&P*, 2023.

[36] Z. Alebouyeh and A. Jalaly Bidgoly, "Privacy-preserving federated learning compatible with robust aggregators," *Available at SSRN 4793556*, 2024.

[37] S. Hou, S. Li, T. Jahani-Nezhad, and G. Caire, "Priroagg: Achieving robust model aggregation with minimum privacy leakage for federated learning," *IEEE Trans. Inf. Forensics Security*, 2025.

[38] Z. Xing, Z. Zhang, Z. Zhang, J. Liu, L. Zhu, and G. Russello, "No vandalism: Privacy-preserving and byzantine-robust federated learning," *arXiv preprint arXiv:2406.01080*, 2024.

[39] A. Yazdinejad, A. Dehghantanha, H. Karimipour, G. Srivastava, and R. M. Parizi, "A robust privacy-preserving federated learning model against model poisoning attacks," *IEEE Trans. Inf. Forensics Security*, 2024.

[40] M. Naseri, J. Hayes, and E. De Cristofaro, "Local and central differential privacy for robustness and privacy in federated learning," *arXiv preprint arXiv:2009.03561*, 2020.

[41] X. Ma, X. Sun, Y. Wu, Z. Liu, X. Chen, and C. Dong, "Differentially private byzantine-robust federated learning," *IEEE Trans. Parallel Distrib. Syst.*, 2022.

[42] Y. Allouah, R. Guerraoui, N. Gupta, R. Pinot, and J. Stephan, "On the privacy-robustness-utility trilemma in distributed learning," in *Proc. ICML*, 2023.

[43] X. Gu, M. Li, and L. Xiong, "Dp-brem: Differentially-private and byzantine-robust federated learning with client momentum," *arXiv preprint arXiv:2306.12608*, 2023.

[44] W.-N. Chen, C. A. C. Choo, P. Kairouz, and A. T. Suresh, "The fundamental price of secure aggregation in differentially private federated learning," in *Proc. ICML*, 2022, pp. 3056–3089.

[45] A. Shamir, "How to share a secret," *Commun. ACM*, 1979.

[46] R. J. McEliece and D. V. Sarwate, "On sharing secrets and reed-solomon codes," *Commun. ACM*, 1981.

[47] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. AISTATS*, 2019.

[48] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *Proc. EUROCRYPT*, 2011.

[49] R. Gennaro, M. O. Rabin, and T. Rabin, "Simplified vss and fast-track multiparty computations with applications to threshold cryptography," in *Proc. ACM PODC*, 1998.

[50] G. Asharov and Y. Lindell, "A full proof of the bgw protocol for perfectly secure multiparty computation," *J. Cryptology*, 2017.

[51] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Adv. Cryptology—CRYPTO'91*, 1992.

[52] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proc. ACM STOC*, 1988.

[53] C. E. Shannon, "Communication theory of secrecy systems," *Bell Syst. Tech. J.*, 1949.

[54] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Process. Mag.*, 2012.

[55] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[56] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. of Toronto, Tech. Rep., 2009.

[57] V. Shejwalkar and A. Houmansadr, "Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning," in *Proc. NDSS*, 2021.

[58] G. Baruch, M. Baruch, and Y. Goldberg, "A little is enough: Circumventing defenses for distributed learning," *Adv. Neural Inf. Process. Syst.*, 2019.

[59] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *Proc. USENIX Sec. Symp.*, 2020.

[60] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proc. AISTATS*, 2020.

[61] M. Egger, C. Hofmeister, A. Wachter-Zeh, and R. Bitar, "Private aggregation in wireless federated learning with heterogeneous clusters," in *Proc. IEEE ISIT*, 2023.

[62] X. Zhang, Z. Li, K. Wan, H. Sun, M. Ji, and G. Caire, "Fundamental limits of hierarchical secure aggregation with cyclic user association," *arXiv preprint arXiv:2503.04564*, 2025.