Technische Universität München
Professur für Coding and Cryptography
Prof. Dr.-Ing. Antonia Wachter-Zeh

Master's Thesis

Byzantine-Resilient and Information-Theoretically Private Federated Learning

Vorgelegt von:
Yue Xia

München, May 2024

Betreut von:
Dr. Ph.D. Rawad Bitar
M.Sc. Maximilian Egger
M.Sc. Christoph Hofmeister

Master's Thesis an der
Professur für Coding and Cryptography (COD)
der Technischen Universität München (TUM)
Titel : Byzantine-Resilient and Information-Theoretically Private Federated Learning
Autor : Yue Xia

Yue Xia
Theresienstr. 90
80333 München
yue1.xia@tum.de

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderung entnommen wurde.

München, 28.05.2024

. . . .

Ort, Datum (Yue Xia)

Contents

List of Figures

List of Tables

Abstract

Federated learning (FL) emerged as a new paradigm enabling training neural networks on private data owned by multiple users. Users run local computations on their data and only share the computation results with the federator coordinating the learning process. FL faces many challenges such as guaranteeing privacy of the users' data and security against malicious users. It has been shown that, by only observing the computational results, the federator can infer information about the users' data. Secure aggregation is proposed in the literature where only the aggregate result is shared with the federator. Hence, guaranteeing privacy. Security is typically guaranteed by running statistics on the users' computations and pruning the outliers. Thus, a natural tradeoff arises between privacy (not learning the users' data) and security (requiring learning statistics on the users' data to prune the outliers). Proposed solutions for this challenge design secret sharing schemes to compute the desired statistics and prune the outliers in a private manner. However, those schemes guarantee computational privacy, i.e., assume eavesdroppers to have bounded computational power.

We design a private and secure federated learning scheme that guarantees perfect information-theoretic privacy and leaks as less information as possible about the users' data. We build our scheme on FLTrust. The federator computes a trust score (TS) for each users' computation result based on a small representative dataset at the federator and uses these TSs as weights for aggregation. We introduce a novel method to compute the aggregation by only observing a privatized version of the users' computation. We approximate the ReLU function used to compute TSs by a polynomial, then rely on the additive homomorphism of secret sharing. We present two different schemes. The first uses a trusted authority that distributes Beaver triples that unlock a multiplicative homomorphism property of the secret shares. The second instead uses Lagrange coded computing for evaluating polynomial functions of the secret shares. We compare our schemes to the literature and show that the first scheme outperforms all the existing ones in computation and communication complexity while achieving a better privacy guarantee. The only caveat is the need for a trusted authority, which is used in a pre-processing phase before the data is known, to distribute randomnesses to users.

1 Introduction

Federated learning (FL) emerged as a promising paradigm enabling the training of neural networks on private data possessed by a large amount of users, allowing users to train the model under the coordination of a central server, called federator, while keeping their private data localized. The training process is distributed across multiple devices. In each training iteration, the federator sends the current global model to the users, each user then updates it based on its local training data, creates a local model update and returns the model update to the federator. Upon receiving the local model updates, the federator uses an aggregation rules to combine them into the new global model for the next iteration. This repeats until the model attains certain converge criteria.

Compared to centralized machine learning, Federated learning addresses the data privacy problem by allowing users to keep their private data local and send the model updates to the federator instead. However, this introduces additional privacy concerns. The local model updates, gradients or weights, still contain sensitive information about the users' private local data. Once the federator receives the local model updates, it is possible to reconstruct users' private training data through inversion attacks [ZLH19], [GBDM20]. Private aggregation protocols [BIK+17] [AHW+17] are introduced to ensure the privacy of individual updates such that the federator only obtains the aggregation of the local model updates.

Moreover, federated learning introduces security concerns by allowing users sending corrupt updates that manipulate the global model directly. We call malicious users behaving arbitrarily Byzantine, who may send arbitrary vectors to the federator. [BEMGS17] shows that when employing a linear aggregation rule, such as FedAvg [MMR+17], even a single Byzantine worker can force the aggregation rule to select any arbitrary value as the output result. This may lead to convergence to sub-optimal models, or even divergence [KMA+21]. The primary countermeasure to defend against such Byzantine attack is identify and remove the outliers in users' updates, which are more likely to be vectors presented by Byzantine users. Many aggregation rules against Byzantine attacks have been proposed ([BEMGS17], [YCKB18], [GR+18], [FYB18], [XKG19], [CFLG20], [ZSWJ22]). Krum [BEMGS17] is a distance-based approach. With N being the number of total users in the system, assumption on the number of Byzantine workers B has to be made.

The local update having the minimum distance with its $N - B$ neighbors, where no Byzantine users are included by assumption, is chosen to be the aggregation result. However, the distance-based approach can be easily fooled when considering high dimensional data. Statistic-based approaches such as Trimmed Mean and Median are proposed in [YCKB18]. They perform coordinate-wise operations: for each coordinate, they either find the mean value after trimming off a certain percentage of the highest and lowest values or find the median value of this coordinate. [GR+18] is a combination of Krum and Trimmed Mean. [FYB18] use cosine similarity of identify malicious updates. FLTrust [CFLG20] also use cosine similarity as a measure, but assumes a small representative root dataset possessed by the federator, based on which it computes a trust score for each user and use the trust score to scale the users' updates during aggregation. [ZSWJ22] requries the federator to first inverse the local model update submitted by individual user to generate a corresponding dummy dataset, then compare the resulting dummy datasets for finding the outliers. However, all these approaches require the federator to learn the actual values of the local updates for learning the statistics of them, which is therefore not privacy preserving.

Hence, there is an inherent tension between security and privacy: for security, the aggregator needs access to the plain local updates in order to learn statistics of them and identify the outliers; for privacy reason, on the other hand, the true local updates must remain concealed. This tension between the objectives makes it challenging to develop methods that are jointly secure and private, which is the focus of this work.

Many Byzantine-resilient secure aggregation schemes [SGA20,JNMAC23,VXK21,HLX+21, MMM+22,XOWZiA23,GMS+23,FMM+21,DWL+23,XLZ+22,HLW+23,BIMP+24,ZTX+24, ZL24, LLTW23, AJB, DCL+21, MYL+24] are proposed, but are either based on clustering and compromise privacy or require two federators during execution or consider computational privacy. Among computational private schemes, the most related works are BREA [SGA20] and ByzSecAgg [JNMAC23]. Both use secret sharing schemes to make Krum [BEMGS17], a distance-based Byzantine-resilient aggregation rule, privacy-preserving. Both schemes are computationally private and leak the pairwise distances between local updates to the federator. Additionally, BREA leaks some extra information to the federator, through not re-randomizing secret sharings, cf. [XHEB24]. ByzSecAgg alleviates this problem by incorporating additional randomness.

While computational privacy has been the main focus in most studies due to its efficiency, all methods rely on computational hardness assumptions potentially vulnerable to attacks, such as those run by quantum computers. If only a limited number of entities collude to compromise privacy, information-theoretic (IT) privacy provides the strongest possible privacy guarantee, which withstands even computationally unbounded adversaries. Currently, no Byzantine-resilient scheme in the literature provides full IT-privacy and without privacy compromises in the literature.

For this reason, we propose two Byzantine-resilient and information-theoretically (IT) private secure aggregation schemes in this thesis. We name them ByITFL and LoByITFL respectively. ByITFL is an information-theoretically private solution without privacy compromises but suffers from high communication and computation costs, growing with the fifth power of the number of users. In LoByITFL, we reduce the communication and computation cost significantly, i.e. with total communication and computation cost quadratic in the number of users, by assuming the existence of a trusted third party distributing randomnesses before the start of the training algorithm. Both schemes are built on FLTrust [CFLG20] for Byzantine-resilience. The federator possesses a small representative root dataset to obtain a federator model update. This enables the computation of a trust score (TS) for each user through a discriminator function on the cosine similarity between the user's and federator's model update. To enable IT privacy, we use a polynomial as the discriminator function. The two solutions then use different methods to compute the aggregation in an IT private manner.

In ByITFL, each user's model update is embedded into a finite field through a stochastic quantizer, partitioned into M sub-vectors, secret shared with all users by Lagrange Coded Computing (LCC) [YLR$^+$19], and verified against corruptions using an IT verifiable secret sharing (ITVSS) scheme [BOGW88], [AL17]. Re-randomization [GRR98] is required before reconstructing the aggregation result to perfectly hide the local model updates. From the perspective of LCC, Byzantine users can be seen as errors and dropouts as erasures in a Reed-Solomon code, cf. [MS81]. The federator decodes the aggregate by Reed-Solomon error-correction decoding and updates the global model. With k the degree of the approximation polynomial and M the number of sub-vectors, the proposed scheme is resilient against any B Byzantine users, IT private against the federator and against any T colluding users, and robust against any P dropouts, as long as $N \geq 2B + (k+2) \cdot (M + T - 1) + P + 1$.

In LoByITFL, we require a Trusted Third Party (TTP) during initialization, which can be implemented through a trusted execution environment, to distribute Beaver triples [Bea92] *once* before the start of the training algorithm. Through an additively homomorphic secret sharing, each user's model update is shared with all other users. Beaver triples enable a multiplicative homomorphism, and hence, polynomial computations can be performed on shared model updates without increasing the degree of the encoding polynomial. Before reconstruction, an additively homomorphic Message Authentication Code (MAC) is used for an integrity check to prevent the malicious users from sending corrupt messages during the polynomial computation [BDOZ11]. The federator decodes the aggregation and updates the global model. The proposed scheme is resilient against any B Byzantine users, IT private against the federator and against any T colluding users, and robust against any P dropouts, as long as $N \geq B + T + P + 1$.

2 Related Work

2.1 Byzantine-Resilient Secure Aggregation

Many Byzantine-resilient secure aggregation schemes [SGA20,JNMAC23,HKJ20,VXK21, HLX+21,MMM+22,XOWZiA23,GMS+23,FMM+21,DWL+23,XLZ+22,HLW+23,BIMP+24, ZTX+24,ZL24,LLTW23,AJB,DCL+21,MYL+24] are proposed. They either are based on clustering and compromise privacy or require two federators during execution or consider computational privacy.

2.1.1 BREA and ByzSecAgg

The most relevant works are BREA [SGA20] and ByzSecAgg [JNMAC23]. BREA [SGA20] represents the first Byzantine-resilient secure aggregation scheme that only requires a single federator. It is built on Krum, which requires the federator to learn the pairwise distances between users' local updates to removes outliers. Each user employs a verifiable Shamir secret sharing scheme [Sha79], [Fel87] to secret shares its local model update with other users, such that they can compute the pairwise distances between the secret shares locally and send the computation results to the federator. Note that the Verifiable Secret Sharing (VSS) scheme used here is computational private, which relies on the hardness of computing discrete logarithms. Upon receiving a sufficient number of computation results, the federator can decode the pairwise distances between the local model updates and apply Krum for user selection. The federator then broadcasts the selected users to all participants and awaits users computing the aggregation of selected secret shares locally. Following the decoding of the aggregated results, the server is ready to update the global model for the next iteration.

BREA is secure against B Byzantine users, T colluding users and P dropout users. By treating the local model updates from Byzantine users as errors and dropout users as erasures, the federator can always decode the correct pairwise distances and aggregation result. This is achieved through leveraging the error correction property of Reed-Solomon code, ensuring reliable decoding as long as the number of users $N \geq 2B + 2T + P + 1$. However, BREA suffers from two privacy leaks. The first one is obvious: the server obtains all pairwise distances between the local model updates. This clearly leaks some relationship between users and can be extremely harmful even if the federator accidentally gets the local model update of a single user in plaintext. The second leakage arises from users' local computation of pairwise distances between secret shares. Specifically, in BREA, shares of the pairwise distance is computed by multiplication on secret shares. However, Shamir secret sharing is not multiplicative homomorphic [Ben86]. The reason for this lies in the fact that multiplying two secrets a and b, which are encoded by $u_a(x)$ and $u_b(x)$ respectively, gives us a secret encoded by the polynomial $u(x) = u_a(x) \cdot u_b(x)$. While the free coefficient of $u(x)$ is indeed the desired multiplication result, i.e. $u(0) = a{\cdot}b$, the other coefficients are no longer completely random and cannot perfectly hide the secrets a and b [BOGW88]. Thus, by interpolating the polynomial $u(x)$ during decoding, the federator obtains additional information about a and b, while we would only allow revealing the product ab as output.

ByzSecAgg [JNMAC23] improves the communication efficiency by partitioning local model updates into M smaller sub-vectors and letting each user use Mcliece-Sarwate secret sharing [MS81] rather than Shamir secret sharing scheme to share their local model updates. This reduces the dimension of problem from d to $\frac{d}{M}$. However, ByzSecAgg also considers computational privacy and leaks all pairwise distances between model updates. With regard to the randomness issue that arises when computing pairwise distances, ByzSecAgg alleviates it by incorporating additional randomness.

2.1.2 Other Algorithms

Some solutions require two federators during execution and focus on computational privacy. [XLZ+22] proposes PPFDL, which requires two federators during the training algorithm. PPFDL uses additive homomorphic encryption schemes and the garbled circuit to perform addition, multiplication and division for computing the aggregation while reducing the negative impact of irregular users. [HKJ20] also proposes a solution with two federators, where each user's model update is split into two additive secret shares and each federator gets one of the shares. The federators then collaboratively perform a distance-based secure aggregation rule based on the additive secret shares they hold. The pairwise distances are leaked to the federator. [DCL+21] uses additive homomorphic encryption and two federators perform secure two-party computation for performing the hamming distance-based aggregation rule to defend against Byzantine attacks. Byzantine-resilience of SecureFL [HLX+21] is also built on FLTrust, but two federators are required during training. Linear operations are computed by the additive homomorphism of additive secret sharing and computation of reciprocal square root and the ReLU function in FLTrust are realized by utilizing boolean sharing. [MMM+22] uses cosine similarity between local

model update and the aggregated update to identify Byzantine users. Two federators are required and computations are done by a two-trapdoor homomorphic encryption scheme. RFed [MYL+24] requires two federators and leverages an efficient privacy preserving mechanism Cheon-Kim-Kim-Song (CKKS), homomorphic encryption and secure multi-party computation to achieve both computational privacy and security. WVFL [ZTX+24] also proposes a solution that requires two federators and utilizes additive secret sharing and linear homomorphic hash functions.
Some schemes only need a single server, but also focus on computational privacy. [FMM+21] proposes SAFELearn, where each user receives the encrypted global model from the federator, decrypts it and trains a model update. Users send the encrypted update to the federator, who utilizes fully homomorphic encryption, multi-party computation or secure two-party computation for the aggregation. In NSPFL [ZL24], an aggregation server, a third party and a cloud service provider are necessary for the execution of the training algorithm and guaranteeing computational privacy. RVPFL [LLTW23] also requires a federator and another platform for aggregation in a private manner through homomorphic encryption. [HLW+23] uses cosine similarity as a measure to identify malicious updates. It leverages compressive sensing to approximate it and achieves computational privacy protection. [DWL+23] and [BIMP+24] achieve computational privacy by leveraging multi-party computation to perform a Byzantine-robust aggregation method. [AJB] obtains computational privacy by modifying the training data and the model training process. SAFEFL [GMS+23] uses secret sharing-based multi-party computation, i.e. the MP-SPDZ [Kel20] framework, for enabling the privacy-preserving computation of the Byzantine-resilience aggregation techniques.
Some works cluster the users and compromise the privacy to achieve security. Users are randomly clustered in SHARE [VXK21]. Then the Byzantine-resilient aggregation is performed in each cluster to filter malicious updates. The cluster aggregated update is revealed to the federator. Through several reclustering rounds, Byzantine robustness is enhanced. FedGT [XOWZiA23] clusters the users into overlapping groups and apply group testing to identify the Byzantine users. It also reveals the aggregate of each test group and therefore shows the trade-off between users' data privacy and security.

2.2 Secret Sharing based Multi-Party Computation

The BGW [BOGW88] protocol is one of the first and most celebrated information-theoretically private secret sharing based multi-party computation (MPC) protocols, which is secure even in the present of a malicious adversary. It heavily relies on Shamir secret sharing [Sha79] and use its property to perform addition, multiplication and scaling on the shares. Share addition and scaling are quite trivial and involves local computation only: decoding the computation on the secret shares directly gives us the desired computed secret. Unfortunately, as mentioned before in Section 2.1.1, share multiplication is non-trivial for Shamir secret sharing, not only because multiplication increases the degree of the encoding polynomial, but the coefficients of the computed encoding polynomial are also not random any more. Since the non-randomness in the coefficients may lead to some leakage of the secret, computing multiplication by merely multiplying the shares, as is done in addition, does not guarantee information-theoretic security. BGW solves these problems by performing two additional steps before reconstructing the computation, namely a degree reduction step and a re-randomization step, which require further sharing, verification, addition and secret reconstruction. [GRR98] presented the simplification of the additional steps. For security against malicious adversaries, Verifiable Secret Sharing (VSS) is used to ensure the validity of the shares received by users and error correction of Reed-Solomon code is used in decoding to mitigate the impact of malicious users manipulating the shares and the computation. By performing BGW, for each multiplication, the number of users needed are $n \geq 3t$, with t being the number of malicious users. However, the BGW protocol requires a lot of communication and interaction between users and between the server and the user. For every multiplication gate, the communication cost for the simplified BGW protocol is $O(n^4)$ over point-to-point private channels if no malicious user misbehaves, and is $O(n^6)$ over point-to-point private channels with $O(n^6)$ over broadcast channel if there is any malicious behavior [AL17]. Note that, if we have $n \geq 4t$ users, then the degree reduction step is not needed, as there are a sufficient number of users to provide redundancy for error correction in polynomials with degree $2t$, then we get a much simpler case.

[Bea92] presented Beaver triple, which will be introduced in detail in Section 3.2.3, to deal with the large amount of communication overhead and the increase of the degree in the BGW protocol when performing multiplication on shares. It splits the problem into a pre-processing phase and an online phase. Upon successfully generating enough number of Beaver triples in the pre-processing phase, the online phase is extremely efficient by consuming one Beaver triple for each multiplication and it doesn't increase the degree of the encoding polynomial.

BDOZ [BDOZ11] make use of Beaver triples and proposed a secret-sharing based multiparty computation protocol that is computationally private in the pre-processing phase and information-theoretically private in the online phase, and is secure against malicious adversary. Additive secret sharing is used in BDOZ, where the secret is the addition of all the shares. In the pre-processing phase, BDOZ uses semi-homomorphic encryption scheme to generate Beaver triples, then the online phase, by consuming Beaver triples for multiplication and using information-theoretic one-time Message Authentication Code (MAC) for integrity check, is able to perform addition and multiplication with optimal efficiency. After the introduction of BDOZ, more secret sharing based multi-party computation schemes emerged, including SPDZ [DPSZ12], SPDZ2 [DKL$^+$13], MASCOT [KOS16], Overdrive [KPR18] and MP-SPDZ [Kel20]. These schemes, all based on additive secret sharing, offer many options depending on the context when considering secret sharing based MPC. Nonetheless, they share a common strategy of dividing the process into the pre-processing and online phase, and utilizing Beaver triples to simplify the multiplication on shares. As a result, these Beaver triple based schemes guarantee information-theoretical security against any $t < n$ malicious users in the online phase, instead of honest majority or even a relatively less t due to the need of error correction.

3 System Model and Preliminaries

We use $[n]$ to denote the set of positive integers $\{1, \cdots, n\}$ and use $\lfloor x \rfloor$ for the largest integer less than or equal to x. We denote $x[i]$ as the secret share of the variable x held by user i and use $\{x\}$ to denote the secret shares of the variable x accompanied by the corresponding Message Authentication Code, i.e. $\{x\} = \{x[i], \mathrm{MAC}(x[i])\}$. $\langle x, y \rangle$ denotes the dot product of x and y. Vectors are denoted in bold type and scalars in normal type. For a ordered sequence $x_1, \ldots, x_n$ and a set $\mathcal{S} \subseteq [n]$ of indices let $x_{\mathcal{S}} = \{x_i : i \in S\}$. The mutual information between two discrete random variables X and Y is denoted as $I(X;Y)$ while $H(X)$ denotes the entropy of X.

3.1 System Model

We consider an FL setting with a single federator, N users and a Trusted Third Party (TTP) in LoByITFL, which is used only before the start of the training process. As is illustrated in 3.1.

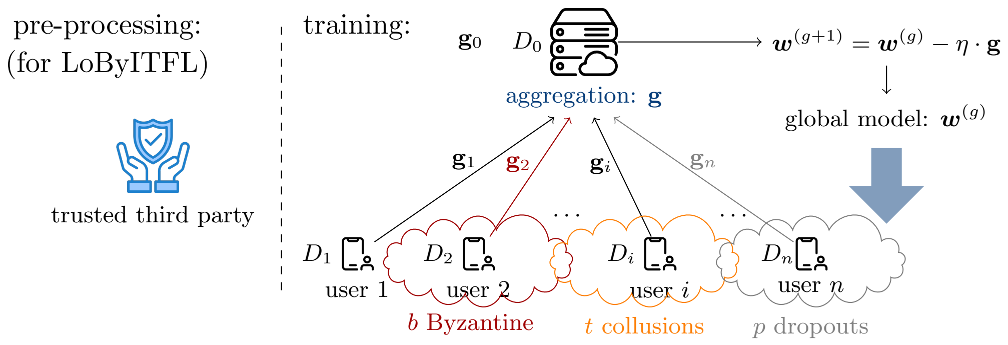

Figure 3.1: Federated learning system.

While the federator is honest-but-curious, B users are Byzantine, T users are colluding and P users may drop out during the execution. Each user i, for $i \in [n]$, holds a private dataset D_i. The federator has a small representative root dataset D_0 and coordinates the training process through interaction with the users. It possesses a d-dimensional global model $\boldsymbol{w} \in \mathbb{R}^d$ and aims to train the global model using private data held by the users to find the optimal global model $\boldsymbol{w}^*$:

$$\boldsymbol{w}^* = \operatorname{argmin}_{\boldsymbol{w}} F(\boldsymbol{w}).$$

$F(\boldsymbol{w})$ is the loss function and $\nabla f(D, \boldsymbol{w})$ is an unbiased estimator of the true gradient $\nabla F(\boldsymbol{w})$, i.e. $\nabla F(\boldsymbol{w}) = \mathbb{E}_{D \sim \mathcal{D}}[\nabla f(D, \boldsymbol{w})]$. Specifically, in each global iteration g, the federator sends the current global model $\boldsymbol{w}^{(g)}$ to the users. Each user i initializes its local model to the current global model, i.e., $\boldsymbol{w}_i^{(0)} = \boldsymbol{w}^{(g)}$, and updates it for one or more local iterations by training on D_i according to

$$\boldsymbol{w}_i^{(c+1)} = \boldsymbol{w}_i^{(c)} - \eta_u \cdot \nabla f(D_i; \boldsymbol{w}_i^{(c)}),$$

where η_u is the local learning rate and c is the local iteration. Each user, holding $\boldsymbol{w}_i$ after its local training iterations, sends the local model update $\mathbf{g}_i = \boldsymbol{w}_i - \boldsymbol{w}^{(g)}$ to the federator. Meanwhile, the federator trains on D_0 and obtains $\mathbf{g}_0$, which is assumed to be public. Upon receiving the local updates, the federator aggregates them according to some aggregation rule AGG, i.e., $\mathbf{g} = \mathrm{AGG}(\mathbf{g}_0, \mathbf{g}_1, \cdots, \mathbf{g}_n)$, and computes the global model for the next iteration [1]

$$\boldsymbol{w}^{(g+1)} = \boldsymbol{w}^{(g)} - \eta \cdot \mathbf{g},$$

where η is the global learning rate.

3.2 Preliminaries

3.2.1 FLTrust

FLTrust [CFLG20] assumes that the federator holds a small clean training dataset D_0 (referred to as root dataset) and performs local training to compute the federator model update $\mathbf{g}_0$ in each iteration. Upon receiving local model updates $\mathbf{g}_i$ from users, the federator first normalizes all local model updates as

$$\tilde{\mathbf{g}}_i = \frac{\mathbf{g}_i}{\|\mathbf{g}_i\|}, \forall i \in 0, 1, ..., N$$

[1] Although we present our scheme in the simple gradient descent setting, it does not depend on the exact update rule and is applicable to, e.g., momentum and higher order methods and adaptive learning rate schedules. Similarly, it is compatible with additional privacy mechanisms based on adding noise to updates, like differential privacy [?].

to mitigate the influence of potential extremely large/small local updates that may be submitted by Byzantine users. The federator then assigns a trust score TS_i to each user i, which is computed by the discriminator of the cosine similarity between the federator model update $\mathbf{g}_0$ and the local model update $\mathbf{g}_i$. FLTrust chooses the rectified linear unit (ReLU) function $\mathrm{ReLU}(x) = \max(0, x)$ as the discriminator function, shown in equation (3.1). This enables the federator to determine whether to trust the user and the degree of the trust.

$$TS_i = ReLU(cos(\theta_i)) = ReLU(\frac{\langle \mathbf{g}_0, \mathbf{g}_i \rangle}{\|\mathbf{g}_0\| \cdot \|\mathbf{g}_i\|}) = ReLU(\langle \tilde{\mathbf{g}}_0, \tilde{\mathbf{g}}_i \rangle), \tag{3.1}$$

Ultimately, the federator aggregates the local model updates by averaging the normalized updates weighted by their trust scores, as in equation (3.2), then uses the aggregation result $\mathbf{g}$ to update the global model for the next iteration.

$$\mathbf{g} = \frac{\|\mathbf{g}_0\|}{\sum_{i \in [n]} TS_i} \cdot \sum_{i \in [n]} (TS_i \cdot \tilde{\mathbf{g}}_i) \tag{3.2}$$

3.2.2 Linear Secret Sharing Scheme

An (N, T)-linear secret sharing (SS) scheme utilizes a randomized degree-T encoding polynomial $\boldsymbol{u}(x)$ over the finite field $\mathbb{F}_p$ that encodes a secret $\boldsymbol{s} \in \mathbb{F}_p^d$ into n secret shares, i.e. $\boldsymbol{s}[1], \cdots, \boldsymbol{s}[n]$, each secret share is an evaluation of the encoding polynomial at certain point. Observing T of these secret shares do not reveal any information about the secret and observing $T + 1$ of the secret shares can fully reconstruct the secret $\boldsymbol{s}$. We call T the threshold of the SS scheme. The reconstruction is through Lagrange polynomial interpolation: a degree-T polynomial can be interpolated by observing $T + 1$ evaluations of it. Instantiations of a linear SS scheme with threshold T can be Shamir SS [Sha79], McEliece-Sarwate SS [MS81], Lagrange coded computing [YLR+19] or additive SS.

These SS schemes are linear, i.e. additive homomorphic, thus, the addition of the secret shares is equivalent to the secret share of the addition and observing $T + 1$ such shares can reconstruct the addition. This holds for any linear combination of the secrets, but does not hold for multiplication. We detail this property of linear secret sharing scheme through Shamir secret sharing scheme [Sha79] and decsribe Lagrange Coded Computing (LCC) [YLR+19] in detail since we need it in ByITFL.

Shamir secret sharing [Sha79]: The encoding polynomial of Shamir secret sharing is $\boldsymbol{u}(x) = \boldsymbol{r}_0 + \boldsymbol{r}_1 x + \boldsymbol{r}_2 x^2 + \cdots + \boldsymbol{r}_T x^T$, by setting $\boldsymbol{r}_0$ be the secret, i.e. $\boldsymbol{r}_0 = \boldsymbol{s}$, and choosing T random vectors $\boldsymbol{r}_1, \boldsymbol{r}_2, \cdots, \boldsymbol{r}_T$ from the finite field randomly. It then distributes the secret $\boldsymbol{s}$ among a group of N users by sending each user i a share $\boldsymbol{s}[i] = \boldsymbol{u}(\alpha_i)$, where

α_i's are distinct values in $\mathbb{F}_p$. Each secret share is therefore an evaluation of the encoding polynomial $\boldsymbol{u}(x)$ at the point α_i. To reconstruct the secret, a group of more than T users collaboratively solve a Lagrange interpolation problem

$$\boldsymbol{s} = \boldsymbol{u}(0) = \sum_{i=1}^{T+1} \boldsymbol{u}(\alpha_i) \prod_{j=1, j\neq i}^{T+1} \frac{-\alpha_j}{\alpha_i - \alpha_j} = \sum_{i=1}^{T+1} \lambda_i \cdot \boldsymbol{u}(\alpha_i).$$

Since Shamir secret sharing scheme is linear, any linear combination of the secrets can be easily computed locally based on local secret shares. However, for multiplication, collaboration among the users becomes essential to preserve information-theoretically privacy.

More precisely, suppose two scalar secrets, denoted as s_1 and s_2, are shared using the degree-T polynomials $u_1(x)$ and $u_2(x)$, respectively. When aiming to compute the addition (or any linear combination) of the secrets, users can locally compute the addition on their shares, resulting in an evaluation point on the polynomial $u(x) = u_1(x) + u_2(x)$, with $u(0) = s_1 + s_2$. Since $u(x)$ is a random polynomial with degree T, the desired computation result can be obtained through polynomial interpolation and evaluating $u(x)$ at the point $x = 0$. However, when it comes to computation involving multiplication of the secrets, a significant difference emerges. The multiplication of two secrets s_1 and s_2 results in an evaluation point over the polynomial $u(x) = u_1(x) \cdot u_2(x)$, which is a polynomial of degree at most $2T$. While the constant term of $u(x)$ is indeed the desired product, i.e. $u(0) = s_1 \cdot s_2$, two issues arise. Firstly, the increase in the polynomial degree requires a higher number of users needed for decoding the computation result: in the above example, at least $2T + 1$ evaluation points are needed for polynomial interpolation. Secondly, the resulting polynomial $u(x) = u_1(x) \cdot u_2(x)$ is not a completely random polynomial, but with a specific structure, which may leak additional information to the adversary and thus cannot perfectly hide s_1 and s_2. As a consequence, we need additional communication and collaboration when computing functions involving multiplications on the secrets, i.e. an additional re-randomization step [BOGW88, GRR98, AL17] to make the encoding polynomial random again.

Lagrange Coded Computing: [YLR+19] proposed Lagrange Coded Computing (LCC). A secret $\boldsymbol{s}$ is partitioned into M smaller sub-vectors. LCC leverages the Lagrange polynomial to create redundancy during encoding, which allows the users to compute the target polynomial $f(x)$ over the secret as if no coding is taking place. LCC provides resiliency against P straggles, security against B Byzantine workers and information-theoretic privacy against T colluding workers.
Specifically, let the secret $\boldsymbol{s}$ be partitioned into M batches $\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}, \cdots, \boldsymbol{s}^{(M)}$, the com-

puting system has N workers and is in the finite field $\mathbb{F}_p$. By selecting $M+T$ distinct elements $\beta_1, \beta_2, \cdots, \beta_{M+T}$ from $\mathbb{F}_p$ and choosing T uniformly random $\boldsymbol{r}^{(i)}$ with the same size as $\boldsymbol{s}^{(i)}$'s, it is able to construct a Lagrange interpolation polynomial

$$\begin{aligned}\boldsymbol{u}(z) = &\sum_{j\in[M]} \boldsymbol{s}^{(j)} \cdot \prod_{l\in[M+T]\setminus\{j\}} \frac{z-\beta_l}{\beta_j-\beta_l} \\ &+ \sum_{j\in[T]} \boldsymbol{r}^{(j)} \cdot \prod_{l\in[M+T]\setminus\{M+j\}} \frac{z-\beta_l}{\beta_{M+j}-\beta_l}, \forall i \in [N],\end{aligned} \tag{3.3}$$

where $\boldsymbol{u}$ is a polynomial of degree at most $M+T-1$. It satisfies that $\boldsymbol{u}(\beta_i) = \boldsymbol{s}^{(i)}$ for $i \in \{1, 2, \cdots, M\}$ and $\boldsymbol{u}(\beta_i) = \boldsymbol{r}^{(i)}$ for $i \in \{M+1, \cdots, M+T\}$.
The coded dataset is then computed by selecting N distinct elements $\{\alpha_i\}_{i\in[N]}$ from $\mathbb{F}$ such that $\{\alpha_i\}_{i\in[N]} \cap \{\beta_j\}_{j\in[M]} = \varnothing$ and user i receives the secret share $\boldsymbol{s}[i] = \boldsymbol{u}(\alpha_i)$ for $i \in [N]$. LCC allows users to perform polynomial computations on the secret shares directly. Assume the desired polynomial is f, which is a degree $deg(f)$ polynomial, each worker i then computes $f(\boldsymbol{s}[i]) = f(\boldsymbol{u}(\alpha_i))$ locally. Each computation result is an evaluation of the resulting polynomial $f(\boldsymbol{u}(z))$, whose degree is at most $(M+T-1)\cdot deg(f)$, at the point α_i. Thus, upon having more than $(M+T-1)\cdot deg(f)$ computation results (evaluations) from the workers, the resulting polynomial $f(\boldsymbol{u}(z))$ can be interpolated. Thus, the desired computation results $f(\boldsymbol{s}^{(1)}), \cdots, f(\boldsymbol{s}^{(M)})$ can be obtained by evaluating the polynomial $f(\boldsymbol{u}(z))$ at points $\{\beta_j\}_{j\in[M]}$. Since P straggles, B Byzantine workers and T colluding workers are considered, the number of workers N should satisfy $N > (M+T-1)\cdot deg(f) + 2B + P + 1$ for Reed-Solomon error correction decoding.
However, since LCC is essentially a linear secret sharing scheme, before reconstructing the computed secret $f(\boldsymbol{s})$, additional operation, i.e. re-randomization of the computed polynomial, is also essential to perfectly hide the secret $\boldsymbol{s}$.

3.2.3 Beaver Triple

The Beaver triple [Bea92] is a useful tool to deal with the large communication cost which is required by the re-randomization step in the BGW protocol [BOGW88], when computing multiplication on shared secrets. Beaver triple is a triple of values a, b, c, where a, b are two independent and uniformly random values in the underlying finite field and c is the product of a and b, i.e. $c = ab$. The triple is secret-shared among all n users, who want to compute the product of two secret-shared messages x and y collaboratively. Note that, the chosen SS scheme has to be identical and a linear SS scheme with threshold T, and none of the users can obtain the values of the triple and the messages as long as the number of colluding users is less than the threshold of the chosen SS scheme. Also, one

Beaver triple can only be used once to provide privacy.
Each user i holds one secret share of the Beaver triple $a[i]$, $b[i]$, $c[i]$ and the messages $x[i]$ and $y[i]$ and wants to compute a secret share of the product xy. To this end, each user: 1) computes $x[i] - a[i]$ and $y[i] - b[i]$, 2) sends the results to one dedicated participant, who reconstructs and publicly announces the values of $x - a$ and $y - b$, and 3) computes its secret share of the product as

$$(xy)[i] = (x-a)(y-b) + (x-a)b[i] + (y-b)a[i] + c[i],$$

which only involves addition and scaling of secret shares. Reconstructing from enough shares $(xy)[i]$ gives $xy = (x-a)(y-b) + (x-a)b + (y-b)a + c$.
Therefore, we can convert the multiplication on the secret shares to linear computation, which can be trivially solved because of the additive homomorphism of the chosen linear SS scheme, by consuming one Beaver triple.

3.2.4 Message Authentication Code

In LoByITFL, we need the information-theoretically private one-time Message Authentication Code (MAC) for integrity check, such that Byzantine users cannot cheat during the computation without being detected.
Suppose the underlying finite field, where all operations are carried out, is $\mathbb{F}_p$. We use the information-theoretic one-time MAC with key $(\alpha, \beta) \in \mathbb{F}_p^2$ as in the BDOZ protocol [BDOZ11]. 'One-time' implies that the message authentication code remains secure only when the key is used at most once. Formally, the authentication code of a message x is defined to be $\text{MAC}_{\alpha,\beta}(x) = \alpha \cdot x + \beta$.
To allow computation to be performed on MAC, it is important to keep α constant over different MAC keys, such that they are additive homomorphic. It means that, when having two messages x, x' and the corresponding $\text{MAC}_{\alpha,\beta}(x) = \alpha \cdot x + \beta$ and $\text{MAC}_{\alpha,\beta'}(x') = \alpha \cdot x' + \beta'$ with the same α, we can do the following computations on the MACs to get the MACs of the computations:

1. Addition: $\text{MAC}_{\alpha,\beta}(x) + \text{MAC}_{\alpha,\beta'}(x') = \text{MAC}_{\alpha,\beta+\beta'}(x + x')$
2. Scaling: $k \cdot \text{MAC}_{\alpha,\beta}(x) = \text{MAC}_{\alpha,k\cdot\beta}(k\dot{x})$
3. Addition by constant: $\text{MAC}_{\alpha,\beta}(x) = \text{MAC}_{\alpha,\beta-\alpha k}(x + k)$

This allows the integrity check of the computation: an adversary who sees $\text{MAC}_{\alpha,\beta}(x)$ for a certain message x cannot forge another valid $\text{MAC}_{\alpha,\beta}(x')$ for $x \neq x'$ with a forgery probability $\frac{1}{\mathbb{F}_p}$. More precisely, the integrity check runs as follows. Let one party P_i, holding the message x and its corresponding $\mathrm{MAC}_{\alpha,\beta}(x)$, be the party who performs the computation, and another party P_j holding the MAC key (α, β) be the verifier. P_i performs linear computation on the message x, say $f(x)$, in the meanwhile, it performs the same linear function on the corresponding MAC. When P_i is supposed to open the computation result to P_j, it sends the computation result $f(x)$ and the computed $\mathrm{MAC}_{\alpha,\beta}(f(x))$ to P_j, such that the MAC key holder P_j can use the MAC key to compute $\alpha \cdot f(x) + \beta$ and check the consistency with the $\mathrm{MAC}_{\alpha,\beta}(f(x))$ it receives from P_i.

3.3 Threat Model and Defense Goals

We consider Byzantine attackers that arbitrarily deviate from the protocol and have access to all users' datasets. ByITFL and LoByITFL should be resilient and converge even when up to B such users collaboratively misbehave.

ByITFL and LoByITFL guarantee the privacy of honest users' local model updates in each iteration against eavesdroppers with unlimited computing resources, requiring perfect IT privacy. Considering up to T colluding users and the honest-but-curious federator, the privacy constraint is as follows:

Privacy against colluding users: Once knowing the current global model, which is required for the learning task, and the datasets of the colluding users, which may give information about what the datasets of the honest users look like, no set $\mathcal{T}$ of up to $|\mathcal{T}| = T$ colluding users can learn any additional information about the local model updates of the other honest users:

$$I(\mathbf{g}_{[N]\setminus\mathcal{T}}; \mathbf{g}_{\mathcal{T}}, M_{\mathcal{T}} \mid D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) = 0. \tag{3.4}$$

Privacy against the federator: Knowing the current global model and the root dataset, the federator should not gain any information about the local model updates of the honest users beyond the aggregation:

$$I(\mathbf{g}_{[N]\setminus\mathcal{T}}; \mathbf{g}_0, M_f \mid D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) = 0, \tag{3.5}$$

where $M_{\mathcal{T}}$ denotes the messages received by the colluding parties and M_f denotes the intermediate messages received by the federator.

We consider the possibility of a subset of up to P users experiencing delays or dropping out during protocol. The protocol should be IT private against the curious-but-honest federator and against any collusion of up to T users, robust against B Byzantine users and at the same time be able to tolerate up to P users staying silent during the execution.

4 Main Result

ByITFL and LoByITFL both are built on FLTrust [CFLG20] for Byzantine-resilience. The federator possesses a small representative root dataset to obtain a federator model update. This enables the computation of a trust score (TS) for each user through a discriminator function on the cosine similarity between the user's and federator's model update. To enable IT privacy, we use a polynomial as the discriminator function, which makes the computation of the aggregation result a polynomial. The two schemes we propose apply different methods to make this polynomial computation IT private. ByITFL utilizes the property of LCC [YLR+19] to provide IT privacy, corrects the aggregation result by assuming enough number of users and perform error correct decoding. While LoByITFL is suitable for any linear secret sharing scheme, by assuming a trusted third party before the training algorithm, the federator can always ensure the correctness of the aggregation by utilizing an integrity check on the computation results and remove the computations failed in the check. We describe the two schemes one by one.

4.1 ByITFL

We present ByITFL, which leverages the ability of FLTrust [CFLG20] to provide resilience against Byzantine attacks. Assuming the federator holds a small training dataset and performs local training to obtain a federator model update, ByITFL approximates the ReLU function used to compute the trust scores in FLTrust by a polynomial and uses LCC to provide IT privacy against eavesdroppers. ByITFL consists of the following five main steps:

A. Users **normalize and quantize** their local model updates. The federator model update is treated accordingly.

B. The **normalized updates** are partitioned into smaller sub-vectors and **secret shared** using LCC and ITVSS.

C. Users **validate the normalization** based on the secret shares from other users.

D. Users **compute a secret representation** of the aggregation result by evaluating the target polynomials.

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

We describe the details of each step in the sequel.
A. **Normalization and Quantization** To defend against Byzantine attacks performed on the magnitude, the federator and all users first normalize their model update $\mathbf{g}_i$ to a unit vector $\tilde{\mathbf{g}}_i = \frac{\mathbf{g}_i}{\|\mathbf{g}_i\|}, \forall i \in \{0, 1, \cdots, N\}$, so that the impact of extremely large/small local updates, more likely originating from Byzantine users, can be eliminated.
Since the training process is performed in the real domain and LCC (like every IT private secret sharing) works over finite fields, it is essential to transfer the normalized model updates $\tilde{\mathbf{g}}_i \in \mathbb{R}^d$ to vectors in a prime field $\bar{\mathbf{g}}_i \in \mathbb{F}_p^d$, where p is a large prime. Therefore, users apply an element-wise stochastic quantizer $Q_q(x)$ with $2q+1$ quantization intervals as in [SGA20], [JNMAC23]. The relation between p and q is explained later. Note that the stochastic rounding is unbiased, i.e., $\mathbb{E}_Q[Q_q(x)] = x$. Let $\phi(x) = x + p \mod p$ map integers to values in $\mathbb{F}_p$, the quantization is defined to be

$$\bar{\mathbf{g}}_i := \phi(q \cdot Q_q(\tilde{\mathbf{g}}_i)). \tag{4.1}$$

B. **Sharing of the Normalized Model Updates** The federator and the users first partition their normalized model updates $\bar{\mathbf{g}}_i$ into M smaller subvectors

$$\bar{\mathbf{g}}_i = [\bar{\mathbf{g}}_i^{(1)}, \bar{\mathbf{g}}_i^{(2)}, \cdots, \bar{\mathbf{g}}_i^{(M)}]^T, \forall i \in \{0, 1, \cdots, N\},$$

where each sub-vector is of size $\frac{d}{M}$ and $M \in [\frac{N-P+1}{2} - B - T]$.
We assume the federator model update is public, the federator broadcasts the sub-vectors of $\bar{\mathbf{g}}_0$ to the users. Each user i uses LCC [YLR+19] to secret share $\bar{\mathbf{g}}_i$ with all users by the degree-$(T+m-1)$ encoding polynomial

$$\begin{aligned} \boldsymbol{u}_i(z) = &\sum_{j\in[M]} \bar{\mathbf{g}}_i^{(j)} \cdot \prod_{l\in[M+t]\setminus\{j\}} \frac{z-\beta_l}{\beta_j-\beta_l} \\ &+ \sum_{j\in[T]} \boldsymbol{r}_i^{(j)} \cdot \prod_{l\in[M+t]\setminus\{M+j\}} \frac{z-\beta_l}{\beta_{M+j}-\beta_l}, \forall i \in [N], \end{aligned} \tag{4.2}$$

where $\beta_1, \cdots, \beta_{M+t}$ are $M+t$ distinct elements from $\mathbb{F}_p$ and $\boldsymbol{r}^{(j)}$'s are chosen independently and uniformly at random from $\mathbb{F}_p$. Note that $\boldsymbol{u}_i(\beta_1) = \bar{\mathbf{g}}_i^{(1)}, \cdots, \boldsymbol{u}_i(\beta_M) = \bar{\mathbf{g}}_i^{(M)}$ and the finite field size should be large enough to avoid any wrap around, which we describe in subsection 4.1. Secret shares are computed by evaluating $\boldsymbol{u}_i(z)$ at n distinct values $\{\alpha_l\}_{l\in[n]}$, which are selected from $\mathbb{F}_p$ such that $\{\alpha_l\}_{l\in[n]} \cap \{\beta_l\}_{l\in[M]} = \varnothing$. Hence, each user j receives a secret share of $\bar{\mathbf{g}}_i$ from other user i, i.e. $\bar{\mathbf{g}}_i[j] = \boldsymbol{u}_i(\alpha_j)$, which is a vector of size $\frac{d}{M}$, for $i, j \in [N]$.

Note that, we leverage the ITVSS protocol from [BOGW88] to prevent Byzantine users from misbehaving in the secret sharing phase.

C. **Validation of Normalization** Malicious users may misbehave during normalization. Thus, upon receiving a secret share, each user i verifies correct normalization by locally computing the squared $l2$-norm $\|\bar{\mathbf{g}}_j[i]\|_2^2$ of the secret shares for each $j \in [N]$ and sending the computed shares to the federator. This is possible due to LCC and a re-randomization step before sending computations to the federator, which will be detailed in the next subsection. Upon receiving the computation results, the federator utilizes error correction decoding of the underlying Reed-Solomon code, cf. [MS81], to reconstruct $\|\bar{\mathbf{g}}_i\|_2^2$ for each $\bar{\mathbf{g}}_i$ and checks if it is within a certain interval, i.e.,

$$\left| \|\bar{\mathbf{g}}_i\|_2^2 - \phi(q \cdot Q_q(1))^2 \right| < \varepsilon \cdot q^2, \tag{4.3}$$

where ε is a predefined threshold and can be set empirically. Note that the error correction requires the total number of users $N \geq 2B+2(M+T-1)+P+1$. 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.

D. **Users Secure Computation** In FLTrust [CFLG20], the federator assigns to each user a trust score $\mathrm{TS}_i = \mathrm{ReLU}(\cos(\theta_i)), \forall i \in [N]$, where θ_i is the angle between the federator's and the user's model update. The federator aggregates the local model updates by averaging the normalized updates weighted by their trust scores.

Making FLTrust IT private is not straightforward, which is why we choose a degree-k polynomial function $h(x) = h_0 + h_1 x + \cdots + h_k x^k$ as the discriminator function. We detail the choice of this discriminator function in Section 4.5. Therefore, the trust score for each user becomes

$$\mathrm{TS}_i \approx h(\cos(\theta_i)) = h(\langle \bar{\mathbf{g}}_0, \bar{\mathbf{g}}_i \rangle), \forall i \in [N],$$

and the aggregation result is

$$\mathbf{g} = \frac{\|\mathbf{g}_0\|}{\sum_{i \in [N]} \mathrm{TS}_i} \cdot \sum_{i \in [N]} (\mathrm{TS}_i \cdot \bar{\mathbf{g}}_i) = \frac{\|\mathbf{g}_0\|}{\Sigma_1} \cdot \mathbf{\Sigma}_2, \tag{4.4}$$

$$\text{where } \Sigma_1 = \sum_{i \in [N]} h(\langle \bar{\mathbf{g}}_0, \bar{\mathbf{g}}_i \rangle) \text{ and } \mathbf{\Sigma}_2 = \sum_{i \in [N]} (h(\langle \bar{\mathbf{g}}_0, \bar{\mathbf{g}}_i \rangle) \cdot \bar{\mathbf{g}}_i).$$

Since the federator possesses $\mathbf{g}_0$, for computing the aggregation $\mathbf{g}$ in a privacy-preserving manner, the federator only needs to compute the value of $\frac{\mathbf{\Sigma}_2}{\Sigma_1}$ in a private manner without learning individual users' private information beyond this quotient. The colluding users should learn nothing about other honest users during the computation.

Privacy Against Colluding Users: Both Σ_1 and $\mathbf{\Sigma}_2$ are polynomial functions of the model updates $\bar{\mathbf{g}}_0$ and $\bar{\mathbf{g}}_i$ for $i \in [n]$, where $\bar{\mathbf{g}}_0$ is public and $\bar{\mathbf{g}}_i$'s are secret shared among the users using LCC. Note that LCC allows the computation of an arbitrary polynomial f with degree $\deg(f)$ over its secret. Suppose user i holds a secret $\boldsymbol{s}_i$, the user partitions it and shares it among users via a degree-$(M+T-1)$ encoding polynomial $\boldsymbol{u}_i(z)$. Each user j, holding its secret share $\boldsymbol{s}_i[j] = \boldsymbol{u}_i(\alpha_j)$, is able to compute $f(\boldsymbol{s}_i[j]) = f(\boldsymbol{u}_i(\alpha_j))$ locally, which is an evaluation of the resulting polynomial $f(\boldsymbol{u}_i(z))$ at the point α_j. Upon having more than $(m+t-1)\deg(f)+1$ evaluations from the users, the resulting polynomial $f(\boldsymbol{u}_i(z))$ can be interpolated. The desired computation is obtained by evaluating the polynomial $f(\boldsymbol{u}_i(z))$ at points $\{\beta_l\}_{l\in[M]}$, i.e., $f(\boldsymbol{s}_i) = [f(\boldsymbol{s}_i^{(1)}), \cdots, f(\boldsymbol{s}_i^{(M)})]^T = [f(\boldsymbol{u}_i(\beta_1)), \cdots, f(\boldsymbol{u}_i(\beta_M))]^T$.

Hence, it is possible to perform the polynomial computations of Σ_1 and $\mathbf{\Sigma}_2$ on the secret shares, such that each user obtains an evaluation point of Σ_1 and $\mathbf{\Sigma}_2$. This guarantees that any set of up to T users are not able to learn anything from the shares.

Privacy Against the Federator: Privacy against the federator has not yet been guaranteed: the reconstructions of the computation results, i.e. $\|\bar{\mathbf{g}}_i\|_2^2$, Σ_1 and $\mathbf{\Sigma}_2$, cannot perfectly hide the secret values $\mathbf{g}_1, \cdots, \mathbf{g}_N$ against the federator. LCC is a linear secret sharing scheme that is additively, but not multiplicatively, homomorphic. Given two secrets a and b shared among n users with encoding polynomial $u_a(z)$ and $u_b(z)$, each user i, having the secret shares $a[i] = u_a(\alpha_i)$ and $b[i] = u_b(\alpha_i)$, is able to locally compute the sum of the shares $a[i] + b[i] = u_a(\alpha_i) + u_b(\alpha_i) = u_{a+b}(\alpha_i)$, which perfectly hides the secrets. This property does not hold for multiplication on shares [BOGW88, AL17]. The product of $a[i]$ and $b[i]$ results in a secret share of $u_a(z) \cdot u_b(z)$, whose evaluation at β_1 is indeed $a \cdot b$, but is not a completely random polynomial perfectly hiding the secret, i.e. $u_a(z) \cdot u_b(z) \neq u_{a\cdot b}(z)$. The federator can learn additional information about a and b beyond $a \cdot b$. We follow the re-randomization step from [GRR98, AL17], which involves sub-sharing the users' secret shares using ITVSS [BOGW88], and linearly combining to construct re-randomized secret shares.

The users own the re-randomized shares of Σ_1 and $\mathbf{\Sigma}_2$. It remains to ensure that the federator obtains the quotient $\mathbf{\Sigma}_2/\Sigma_1$ without gaining any additional information about Σ_1 and $\mathbf{\Sigma}_2$. Therefore, each user i 1) chooses an independent value λ_i uniformly at random from $\mathbb{F}_p$ and secret shares it by LCC and ITVSS among all users, 2) adds the shares of λ_j's from all other user j and obtains the share of[1] $\lambda = \sum_{j=[N]} \lambda_j$. Each user

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

multiplies the re-randomized shares of Σ_1 and $\mathbf{\Sigma}_2$ by their share of λ, performs another re-randomization and sends the resulting shares of $\lambda\Sigma_1$ and $\lambda\mathbf{\Sigma}_2$ to the federator.

E. **Secure Aggregation** The federator receives the secret shares of $\lambda\Sigma_1$ and $\lambda\mathbf{\Sigma}_2$, where the degree of the encoding polynomial for $\lambda\Sigma_1$ is $k \cdot (M+T-1)$ and for $\lambda\mathbf{\Sigma}_2$ is $(k+1) \cdot (M+T-1)$. With sufficient number of users sending evaluations to the federator, the federator is able to leverage the error correction property of Reed-Solomon codes [MS81] to decode the values of $\lambda\Sigma_1$ and $\lambda\mathbf{\Sigma}_2$. Therefore, we require the total number of users in the system to be $N \geq 2B + (k+1) \cdot (M+T-1) + P + 1$.

Upon decoding the correct values, the federator computes $\mathbf{\Sigma}_2/\Sigma_1 = \lambda\mathbf{\Sigma}_2/\lambda\Sigma_1$, converts the results from the finite field back to the real domain by de-quantizing by $Q_q(x)^{-1}$ and demapping by ϕ^{-1} and computes 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. Scaling the gradients by this value and summing N such entries results in the range $-dNq^3$ to dNq^3. Accounting for the 0 value, we thus require $p \geq 2dNq^3 + 1$.

4.2 Theoretical Analysis of ByITFL

The following Theorem 1 proves the robustness of ByITFL against Byzantine behavior during the protocol execution and the strong IT privacy guarantee achieved by ByITFL.

Theorem 1. *ByITFL with $N \geq 2B + (k+1) \cdot (M+T-1) + P + 1$ guarantees*

1) IT privacy against any T users according to (3.4) *and against the federator according to* (3.5)*, and*

2) Byzantine-resilience, i.e. it computes the correct aggregation result according to (4.4) *when up to B users are Byzantine.*

Proof of Theorem 1. (Privacy) We first prove the privacy against any T colluding users according to (3.4):

$$\begin{aligned}
&I(\mathbf{g}_{[n]\setminus\mathcal{T}}; \mathbf{g}_{\mathcal{T}}, M_{\mathcal{T}} \mid D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) \\
&= H(\mathbf{g}_{[n]\setminus\mathcal{T}} \mid D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) - H(\mathbf{g}_{[n]\setminus\mathcal{T}} \mid \mathbf{g}_{\mathcal{T}}, M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) \\
&= H(\mathbf{g}_{[n]\setminus\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) - H(D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) \\
&\quad - H(\mathbf{g}_{[n]\setminus\mathcal{T}}, \mathbf{g}_{\mathcal{T}}, M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) + H(\mathbf{g}_{\mathcal{T}}, M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) \\
&= H(\mathbf{g}_{[n]\setminus\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) - H(D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) \\
&\quad - H(\mathbf{g}_{[n]\setminus\mathcal{T}}, M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) + H(M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}),
\end{aligned} \tag{4.5}$$

where the last equation follows because $\mathbf{g}_{\mathcal{T}}$ is a deterministic function of $D_{\mathcal{T}}$ and $\boldsymbol{w}^{(g)}$. We then consider the exchanged messages $M_{\mathcal{T}}$, which include the shares of the normalized local model updates $\bar{\mathbf{g}}_{\mathcal{T}}[i]$ and sub-shares from the re-randomization step in Step C and Step D of the scheme, i.e. when computing the squared $l2$-norm $\|\bar{\mathbf{g}}_i\|_2^2$ to check the correctness of the normalization and the two sums Σ_1 and $\boldsymbol{\Sigma}_2$ in the aggregation. We can leverage the privacy guarantees of LCC [YLR+19], the re-randomization step [GRR98], [AL17] and ITVSS [BOGW88]. When the number of colluding users T satisfies $N \geq 2B+(k+1)\cdot(M+T-1)+P+1$, the exchanged messages $M_{\mathcal{T}}$ observed by the colluding users are completely random and independent of $\mathbf{g}_{[n]\setminus\mathcal{T}}$, $D_{\mathcal{T}}$ and $\boldsymbol{w}^{(g)}$, i.e.,

$$
\begin{aligned}
H(\mathbf{g}_{[n]\setminus\mathcal{T}}, M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) &= H(\mathbf{g}_{[n]\setminus\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) + H(M_{\mathcal{T}}),\\
H(M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) &= H(D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) + H(M_{\mathcal{T}}).
\end{aligned}
$$

By substituting the above into (4.5) we prove (3.4), showing that ByITFL is IT private against T users. For privacy against the honest-but-curious federator, we need to prove (3.5). We have

$$
\begin{aligned}
& I(\mathbf{g}_{[n]\setminus\mathcal{T}}; \mathbf{g}_0, M_f \mid D_0, \boldsymbol{w}^{(g)}, \mathbf{g})\\
&= H(\mathbf{g}_0, M_f \mid D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) - H(\mathbf{g}_0, M_f \mid \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g})\\
&= H(\mathbf{g}_0 \mid D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) + H(M_f \mid \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g})\\
&\quad - H(\mathbf{g}_0 \mid \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) - H(M_f \mid \mathbf{g}_0, \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g})\\
&= H(M_f \mid \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) - H(M_f \mid \mathbf{g}_0, \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}),
\end{aligned}
$$

where the last equation holds because $\mathbf{g}_0$ is a deterministic function of D_0 and $\boldsymbol{w}^{(g)}$. With regard to the exchanged messages M_f, we need to consider: 1) the computed shares of $\|\bar{\mathbf{g}}_i\|_2^2$, $\lambda\Sigma_1$ and $\lambda\boldsymbol{\Sigma}_2$ sent from the users, which are completely random, i.e. uniformly distributed and independent of $\mathbf{g}_0, \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}$, by leveraging the privacy guarantee of the re-randomization step [GRR98], [AL17], and 2) the reconstructed values of $\|\bar{\mathbf{g}}_i\|_2^2$ and $\boldsymbol{\Sigma}_2/\Sigma_1$. Since $\|\bar{\mathbf{g}}_i\|_2^2$ lies within a certain range for all possible model updates (ideally equivalent to one), the value of $\|\bar{\mathbf{g}}_i\|_2^2$ is also independent of $\mathbf{g}_0, \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}$. Regarding $\boldsymbol{\Sigma}_2/\Sigma_1$, we have $H(\boldsymbol{\Sigma}_2/\Sigma_1 \mid \mathbf{g}, \mathbf{g}_0) = 0$ since $\mathbf{g} = \|\mathbf{g}_0\| \cdot \boldsymbol{\Sigma}_2/\Sigma_1$. We denote the computed shares as $\boldsymbol{c}$, and have

$$
\begin{aligned}
& H(M_f \mid \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) = H(\boldsymbol{c}, \|\bar{\mathbf{g}}_i\|_2^2, \boldsymbol{\Sigma}_2/\Sigma_1 \mid \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g})\\
&= H(\boldsymbol{c}, \|\bar{\mathbf{g}}_i\|_2^2 \mid \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) + H(\boldsymbol{\Sigma}_2/\Sigma_1 \mid \boldsymbol{c}, \|\bar{\mathbf{g}}_i\|_2^2, \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g})\\
&= H(\boldsymbol{c}, \|\bar{\mathbf{g}}_i\|_2^2 \mid \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) = H(\boldsymbol{c}, \|\bar{\mathbf{g}}_i\|_2^2),
\end{aligned}
$$

and $H(M_f \mid \mathbf{g}_0, \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) = H(\boldsymbol{c}, \|\bar{\mathbf{g}}_i\|_2^2)$ for the same reason. This concludes the proof of IT privacy against the federator and the proof of Theorem 1.

(Byzantine-Resilience) Byzantine users can present arbitrary incorrect results in any step during the computation. Particularly, they can (1) share invalid secret shares in the secret sharing step, (2) incorrectly normalize local model updates $\tilde{\mathbf{g}}_i$ in the normalization step, (3) present an incorrect computation result of $(\|\mathbf{g}_j\|_2^2)[i]$ during validation of the normalization step, (4) misbehave when computing the discriminator function, $(\lambda\Sigma_1)[i]$ and $(\lambda\boldsymbol{\Sigma}_2)[i]$, and send incorrect computation results to the federator in step D.

To prevent Byzantine users from sending invalid secret shares among users, such as sending secret shares encoded by different encoding polynomials or encoded by some polynomial with higher degree, we leverage the information-theoretic verifiable secret sharing scheme described in [BOGW88] for sharing the model updates.

For Byzantine attacks in the second scenario, since the model updates presented and normalized by honest users lie within a certain range, the misbehaving malicious users presenting malicious normalized vectors and trying to manipulate the learning process through attacks on the magnitude can be detected and identified through the normalization validation. Thus, even if the malicious users present the model updates with the largest squared $l2$-norm that will be accepted by the normalization validation, it only has limited impact since the ϵ is empirically set to be very small.

For Byzantine attacks in the remaining scenarios, i.e. the attempts to send incorrect computation results back to the federator, the malicious behavior will be corrected by error correct decoding during reconstruction of the computations. In the presence of B Byzantine users, by utilizing error correction decoding of Reed-Solomon code [MS81], LCC allows to correct B errors as long as $N \geq 2B+(T+M-1)+P+1$, where $(T+M-1)$ is the degree of the encoding polynomial and P is the number of non-responsive users. Since multiplication on secret shares increases the degree of the encoding polynomial, i.e., the encoding polynomial for $\|\mathbf{g}_j\|_2^2$ is with degree $2(T+M-1)$, for $\lambda\Sigma_1$ is $(k+1)(T+M-1)$ and for $\lambda\boldsymbol{\Sigma}_2$ is $(k+2)(T+M-1)$, the B errors presented by Byzantine users can be corrected as long as $N \geq 2B + (k+2)(T+M-1) + P + 1$. Hence, correctness of the computation is guaranteed. □

4.3 LoByITFL

We propose LoByITFL, a Byzantine-resilient scheme with information-theoretic privacy and low communication cost. The resilience against Byzantine attacks of LoByITFL is based on the robustness of FLTrust [CFLG20] and the integrity check of Message Authentication Codes (MACs). IT privacy is obtained by embedding each user's model update into a finite field through stochastic quantization and secret sharing with all users by linear Secret Sharing (SS) schemes such as Shamir SS [Sha79], McEliece-Sarwate SS [MS81], Lagrange coded computing [YLR$^+$19] and additive SS (see e.g. [BDOZ11]). These linear SS schemes are inherently additively homomorphic; and multiplicative homomorphism can be ensured through Beaver triples. We split the scheme into two phases: the pre-processing phase and the training phase. In the pre-processing phase, we assume the existence of a Trusted Third Party for distributing Beaver triples to users. While in the training phase, per iteration, users share their local model updates with other users by linear secret sharing. With Beaver triples converting multiplication of the secret shares to addition and scaling, the discriminator function, which is a polynomial function, can be computed on the secret shares to obtain the shares of the aggregation result. Note that, during the computation, to prevent Byzantine users from providing corrupt computation results, we utilize additive homomorphic MACs for an integrity check. To this end, the TTP assigns a MAC for each secret share of the local model updates in the pre-processing phase, such that the users, when performing computations on the secret shares, have to perform the same computations on the corresponding MACs and send the computation result and the resulting MACs to the federator to prove the correctness of its computation. Therefore, the federator is able to identify misbehaving Byzantine users and reconstruct the aggregation result correctly. We describe LoByITFL in more details as follows.

LoByITFL is processed in two phases: the pre-processing phase and the training phase.

Pre-processing phase: We require a Trusted Third Party (TTP) in this phase. It generates a sufficient number of Beaver triples a, b, c and sends the corresponding secret shares to the users to enable multiplications on shares in the training phase. In addition, it also samples sufficiently many random vectors $\mathbf{r} \in \mathbb{F}_p^d$ and one random value $\lambda \in \mathbb{F}$ uniformly at random. Then, the TTP sends the random vectors $\mathbf{r}_{i,g}$ and the secret shares $\lambda_g[i]$ to user i, one for each iteration g, and sends the secret shares $\mathbf{r}_{i,g}[j]$ to user j. The random numbers are for secret sharing the local model updates in the training phase efficiently and preserve the privacy of intermediate computation results. We omit g in the following for brevity.

In order to prevent Byzantine users from providing corrupt computation results during the protocol, the TTP assigns an information-theoretic one-time MAC [BDOZ11] for each secret share of the Beaver triples and $\mathbf{r}$'s: $\text{MAC}_{\alpha,\beta}(x) = \alpha \cdot x + \beta$, where x is the secret share, (α, β) is the MAC key and both α and β are sampled from the underlying finite field $\mathbb{F}_p$ uniformly at random. Note that 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 can be performed on the MAC. This allows the integrity check for any linear computation f on a secret share x, by checking the consistency of the resulting MAC $f(\mathrm{MAC}_{\alpha,\beta}(x))$ and the MAC of the computation result $\mathrm{MAC}_{\alpha,\beta}(f(x))$ (we omit α and β later in the text for brevity). Therefore, the TTP also sends the MAC keys (α, β)'s used to generate the MACs for Beaver triples and random numbers to the federator. The federator can then perform the integrity check at the end of each share multiplication, mark users who do not pass the integrity as Byzantine and exclude them from future computation. In other words, at the end of the pre-processing phase, each user i receives $\mathbf{r}_i$, $\{\mathbf{r}_j[i]\}$, $\{\lambda[i]\}$ and a sufficient number of $\{a[i]\}$, $\{b[i]\}$, $\{c[i]\}$, while the federator receives all MAC keys (α, β) used for generating MACs of these variables. The number of random numbers and Beaver triples required for the training phase will be given later.

Training phase: The training phase in each iteration is conducted in five main steps.

A. Users and the federator **normalize and quantize** their model updates. Byzantine attacks are considered in both magnitude and direction of the model update vector. For defending attacks on the magnitude, the model updates $\mathbf{g}_i$, $\forall i \in \{0, 1, \cdots, N\}$, are normalized, which means the federator and all users compute the unit vector $\tilde{\mathbf{g}}_i = \frac{\mathbf{g}_i}{\|\mathbf{g}_i\|}$, $i \in \{0, \ldots, N\}$. Note that the SS schemes works over finite fields, but the training process is conducted in the real domain. We use the unbiased element-wise stochastic quantizer $Q_q(x)$ and the mapper $\phi(x)$ as in previous works [SGA20, JNMAC23, XHEB24], with q being the quantization parameter and $2q + 1$ being the quantization intervals, to map $\tilde{\mathbf{g}}_i$ to vectors in a prime field $\bar{\mathbf{g}}_i \in \mathbb{F}_p^d$, where p is a large enough prime to avoid any wrap around. The relation between p and q will be detailed later.

B. Users **secret share** their local model updates. The federator simply broadcasts $\mathbf{g}_0$ to all users. Each user i secret shares its local model update $\mathbf{g}_i$ to all other users with a linear SS scheme, thus keeping it IT private while making it available for computations on the shares. The secret sharing of user i's model update is supported by the uniformly random vector $\mathbf{r}_i$, that was distributed in the pre-processing phase. Specifically, each user $j \in [N]$ has a share $\mathbf{r}_i[j]$ of $\mathbf{r}_i$ and user i knows the plain value of $\mathbf{r}_i$. User i secret shares $\mathbf{g}_i$ by broadcasting $\mathbf{g}_i - \mathbf{r}_i$ to all other users. User j, for $j \in [n]$ can compute

$$\{\mathbf{g}_i[j]\} = (\mathbf{g}_i - \mathbf{r}_i) + \{\mathbf{r}_i[j]\}. \tag{4.6}$$

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

C. For preventing Byzantine users from incorrectly performing the normalization, each user $i \in [n]$, **validates the normalization** by computing secret shares of the squared $l2$-norm as

$$\{(\|\mathbf{g}_j\|_2^2)[i]\} = \{\langle \mathbf{g}_j[i], \mathbf{g}_j[i] \rangle\}, \tag{4.7}$$

for the local model update $\mathbf{g}_j$ of each user $j \in [n]$. Since $\{(\|\mathbf{g}_j\|_2^2)[i]\}$ is a d-dimensional vector, every $\{(\|\mathbf{g}_j\|_2^2)[i]\}$ computation is done by consuming d Beaver triples to perform a multiplication via linear computations, which enables computation on the linear secret shares and the corresponding MACs. Users send the computation results to the federator, who performs an integrity check on each $\{(\|\mathbf{g}_j\|_2^2)[i]\}$ using the MACs and reconstructs the value of $\|\mathbf{g}_j\|_2^2$. Similar to [XHEB24], the federator checks if $\|\mathbf{g}_j\|_2^2$ lies within a certain interval, i.e.,

$$\left| \|\bar{\mathbf{g}}_j\|_2^2 - \phi(q \cdot Q_q(1))^2 \right| < \varepsilon \cdot q^2,$$

where ε is a predefined small threshold, to compensate for quantization error. Users who fail the normalization check are excluded from future computations.

D. Users compute the discriminator function on the secret shares of the model updates, which gives the **secret shares of the aggregation result**.

This step is based on the non-private scheme FLTrust [CFLG20], which assigns to each user i a trust score determined by the cosine similarity between the federator and the local model update, i.e., $\mathrm{TS}_i = \mathrm{ReLU}(cos(\theta_i))$. We call the function used to compute trust scores discriminator function, i.e., FLTrust uses ReLU as the discriminator function. In [CFLG20], the federator then scales the local model updates by their trust scores and averages them for the aggregation result.

As discussed by the authors of [XHEB24], making FLTrust IT private is not straightforward and requires the transformation of the discriminator function from ReLU to a polynomial. The choice of the discriminator function will be discussed later. In general, the discriminator function is chosen to be a polynomial $h(x)$ with degree k, i.e. $h(x) = h_0 + h_1 x + \cdots + h_k x^k$, which gives the trust score for each user

$$\mathrm{TS}_i = h(\cos(\theta_i)) = h(\langle \bar{\mathbf{g}}_0, \bar{\mathbf{g}}_i \rangle), \forall i \in [N].$$

The aggregation result is therefore

$$\mathbf{g} = \frac{\|\mathbf{g}_0\|}{\sum_{i\in[N]} \mathrm{TS}_i} \cdot \sum_{i\in[N]} (\mathrm{TS}_i \cdot \bar{\mathbf{g}}_i) = \frac{\|\mathbf{g}_0\|}{\Sigma_1} \cdot \mathbf{\Sigma}_2, \\ \text{where } \Sigma_1 = \sum_{i\in[N]} h(\langle \bar{\mathbf{g}}_0, \bar{\mathbf{g}}_i \rangle) \text{ and } \mathbf{\Sigma}_2 = \sum_{i\in[N]} (h(\langle \bar{\mathbf{g}}_0, \bar{\mathbf{g}}_i \rangle) \cdot \bar{\mathbf{g}}_i). \tag{4.8}$$

Notice that, with the discriminator function being a polynomial, the computation of the aggregation only involves the computation of Σ_1 and $\mathbf{\Sigma}_2$, which are polynomial functions of $\mathbf{g}_0$ and $\mathbf{g}_i$, for $i \in [n]$. Since the federator holds $\mathbf{g}_0$, it suffices to compute the quotient $\mathbf{\Sigma}_2/\Sigma_1$ in a privacy-preserving manner. Hence, user i, having $\mathbf{g}_0$, $\{\mathbf{g}_j[i]\}$ for $j \in [n]$, and $\lambda[i]$, computes the secret shares of $\lambda\Sigma_1$ and $\lambda\mathbf{\Sigma}_2$, i.e. $\{(\lambda\Sigma_1)[i]\}$ and $\{(\lambda\mathbf{\Sigma}_2)[i]\}$, by performing polynomial computations on $\mathbf{g}_0$ and $\{\mathbf{g}_j[i]\}$ step by step. For each multiplication the users consume one Beaver triple. Therefore, the computations of $\{(\lambda\Sigma_1)[i]\}$ and $\{(\lambda\mathbf{\Sigma}_2)[i]\}$ can be solved by utilizing the additive homomorphism of both the linear SS scheme and the MAC. Finally, the users obtain one secret share each of $\lambda\Sigma_1$ and $\lambda\mathbf{\Sigma}_2$ and send them to the federator.

E. The federator **reconstructs the aggregation result** upon receiving the secret shares from users. By receiving the $\{(\lambda\Sigma_1)[i]\}$ and $\{(\lambda\mathbf{\Sigma}_2)[i]\}$ from user $i \in [n]$, the federator first checks the integrity of the computation results to ensure the correctness of each secret share, then reconstructs the value of $\lambda\Sigma_1$ and $\lambda\mathbf{\Sigma}_2$ by Lagrange polynomial interpolation, as long as it receives at least $t + 1$ correct computations. Note that, we do not need error correction decoding [MS81] like previous works, which is expensive in computation and requires a higher total number of users for decoding, since the federator ensures the correctness of all computation through the integrity check. Thus, the quotient $\mathbf{\Sigma}_2/\Sigma_1$ is obtained without additional leakage since λ is a uniformly random value unknown to all participants. The federator converts $\mathbf{\Sigma}_2/\Sigma_1$ from the finite field back to the real domain by performing the inverse of the quantizer $Q_q(x)$ and the mapper $\phi(x)$ and computes the global model for the next iteration.

The number of random numbers and Beaver triples required for the training phase (and distributed during the initialization phase) is given in Table 4.1.

To guarantee the correctness of the reconstructed results, none of the intermediate computations may cause a wrap around in the underlying finite field. Since each coordinate of $\bar{\mathbf{g}}_i$ is in the range $-q$ to q and each summation and multiplication during the polynomial computation can increase the range accordingly, we require $p \geq 2Nd^kq^{2k+1} + 1$, where N is the total number of users, d is the dimension of the model updates, q is the quantization parameter and k is the degree of the discriminator function.

Table 4.1: The number of random numbers and Beaver triples required per training iteration. $\{\boldsymbol{l}, \boldsymbol{m}, n\}$ are used for computing the dot product of two vectors and $\{x, \boldsymbol{y}, \boldsymbol{z}\}$ are used for computing the element-wise multiplication of a scalar and a vector, where $\boldsymbol{l}, \boldsymbol{m}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_p^d$, n is the dot product of $\boldsymbol{l}$ and $\boldsymbol{m}$, $\boldsymbol{z}$ is the component-wise multiplication of x and $\boldsymbol{y}$. Those operations generalize Beaver triples from scalars to vectors, each containing d scalar Beaver triples a, b, c.

	Notation	Amount	Purpose (for $i \in [n]$)
Random number	$\mathbf{r}$	n	secret share $\bar{\mathbf{g}}_i$
	λ	1	hide Σ_1 and $\boldsymbol{\Sigma}_2$
Beaver triple	$\mathbf{l}, \mathbf{m}, n$	n^2	compute $\Vert\bar{\mathbf{g}}_i\Vert_2^2$
	a, b, c	$(k-1)n^2$	compute $\langle \bar{\mathbf{g}}_0, \bar{\mathbf{g}}_i \rangle^l$, $l = 2, 3, ..., k$
	$x, \boldsymbol{y}, \boldsymbol{z}$	n^2	compute $h(\langle \bar{\mathbf{g}}_0, \bar{\mathbf{g}}_i \rangle) \cdot \bar{\mathbf{g}}_i$

4.4 Theoretical Analysis of LoByITFL

The following Theorem 2 proves the robustness of LoByITFL against Byzantine behavior during the protocol execution and the strong IT privacy guarantee achieved by LoByITFL.

Theorem 2. *LoByITFL with $N \geq B + T + P + 1$ guarantees*

1) IT privacy against any T users according to (3.4) *and against the federator according to* (3.5)*, and*

2) Byzantine-resilience, i.e. it computes the correct aggregation result according to (4.8) *when up to B users are Byzantine.*

Proof of Theorem 2. (Privacy) We first prove the privacy against any T colluding users according to (3.4):

$$\begin{aligned}
& I(\mathbf{g}_{[n]\setminus\mathcal{T}}; \mathbf{g}_{\mathcal{T}}, M_{\mathcal{T}} \mid D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) \\
&= H(\mathbf{g}_{[n]\setminus\mathcal{T}} \mid D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) - H(\mathbf{g}_{[n]\setminus\mathcal{T}} \mid \mathbf{g}_{\mathcal{T}}, M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) \\
&= H(\mathbf{g}_{[n]\setminus\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) - H(D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) \\
&\quad - H(\mathbf{g}_{[n]\setminus\mathcal{T}}, \mathbf{g}_{\mathcal{T}}, M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) + H(\mathbf{g}_{\mathcal{T}}, M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) \\
&= H(\mathbf{g}_{[n]\setminus\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) - H(D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) - H(\mathbf{g}_{[n]\setminus\mathcal{T}}, M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}) + H(M_{\mathcal{T}}, D_{\mathcal{T}}, \boldsymbol{w}^{(g)}),
\end{aligned} \tag{4.9}$$

where the last equation holds because $\mathbf{g}_{\mathcal{T}}$ is a deterministic function of $D_{\mathcal{T}}$ and $\boldsymbol{w}^{(g)}$. We then consider the exchanged messages. The exchanged messages $M_{\mathcal{T}}$ observed by the colluding users are 1) $\{\mathbf{r}_j[\mathcal{T}]\}$, $\{\lambda[\mathcal{T}]\}$ and $\{a[\mathcal{T}]\}$, $\{b[\mathcal{T}]\}$, $\{c[\mathcal{T}]\}$ sent from the TTP, 2) the values of $\mathbf{g}_i - \mathbf{r}_i$ sent from user i and the secret share $\{\mathbf{g}_i[\mathcal{T}]\}$, 3) $\{(\Vert\mathbf{g}_i\Vert_2^2)[\mathcal{T}]\}$ computed during normalization validation, 4) all intermediate results when consuming

Beaver triples for share multiplication computations, i.e., when computing multiplication of x and y, malicious users get the values of $\{x[\mathcal{T}] - a[\mathcal{T}]\}$, $\{y[\mathcal{T}] - b[\mathcal{T}]\}$, $x - a$, $y - b$ and $\{(xy)[\mathcal{T}]\}$, 5) $\{(\lambda\Sigma_1)[\mathcal{T}]\}$, $\{(\lambda\mathbf{\Sigma}_2)[\mathcal{T}]\}$. We can leverage the privacy guarantees of the secret sharing scheme, the Beaver triple [Bea92] and the MAC scheme. When the number of colluding users is smaller than the threshold of the secret sharing scheme, i.e. T, we have 1) the secret shares observed by the colluding users are completely random and independent of $\mathbf{g}_{[n]\setminus\mathcal{T}}$, $D_\mathcal{T}$ and $\boldsymbol{w}^{(g)}$, 2) the corresponding MACs observed by Byzantine users are also random because of the randomness of β and do not leak any information about the secret, 3) the plaintext values $\mathbf{g}_i - \mathbf{r}_i$, $x - a$ and $y - b$ observed by the Byzantine users are completely random and independent of $\mathbf{g}_{[n]\setminus\mathcal{T}}$, $D_\mathcal{T}$ and $\boldsymbol{w}^{(g)}$ because of the randomness $\mathbf{r}$, a and b, i.e.,

$$\begin{aligned} H(\mathbf{g}_{[n]\setminus\mathcal{T}}, M_\mathcal{T}, D_\mathcal{T}, \boldsymbol{w}^{(g)}) &= H(\mathbf{g}_{[n]\setminus\mathcal{T}}, D_\mathcal{T}, \boldsymbol{w}^{(g)}) + H(M_\mathcal{T}),\\ H(M_\mathcal{T}, D_\mathcal{T}, \boldsymbol{w}^{(g)}) &= H(D_\mathcal{T}, \boldsymbol{w}^{(g)}) + H(M_\mathcal{T}). \end{aligned}$$

By substituting the above into (4.9) we prove (3.4), showing that ByITFL is IT private against T users. For privacy against the honest-but-curious federator, we need to prove (3.5). We have

$$\begin{aligned} &I(\mathbf{g}_{[n]\setminus\mathcal{T}}; \mathbf{g}_0, M_f | D_0, \boldsymbol{w}^{(g)}, \mathbf{g})\\ &= H(\mathbf{g}_0, M_f | D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) - H(\mathbf{g}_0, M_f | \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g})\\ &= H(\mathbf{g}_0 \mid D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) + H(M_f \mid \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g})\\ &\quad - H(\mathbf{g}_0 | \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) - H(M_f | \mathbf{g}_0, \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g})\\ &= H(M_f | \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) - H(M_f | \mathbf{g}_0, \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}), \end{aligned}$$

where the last equation holds because $\mathbf{g}_0$ is a deterministic function of D_0 and $\boldsymbol{w}^{(g)}$. We then consider the exchanged messages M_f observed by the federator. M_f includes 1) the MAC keys sent by the TTP at the pre-processing phase, 2). all intermediate results when using Beaver triples for share multiplication computations, i.e., when computing multiplication of two variables x and y, the federator gets the values of $\{x[i] - a[i]\}$, $\{y[i] - b[i]\}$ for all $i \in [n]$ and the reconstructed values $x - a$ and $y - b$, 3) the computed secret shares $\{(\|\mathbf{g}_j\|_2^2)[i]\}$ for $j \in [n]$, $\{(\lambda\Sigma_1)[i]\}$ and $\{(\lambda\mathbf{\Sigma}_2)[i]\}$ sent from the users $i \in [n]$, 4) the reconstructed values of $\|\mathbf{g}_j\|_2^2$, $\lambda\Sigma_1$, $\lambda\mathbf{\Sigma}_2$ and the quotient $\mathbf{\Sigma}_2/\Sigma_1$.

The MAC keys sent from the TTP are random values that are independent of
$\mathbf{g}_0, \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}$. With regards to the intermediate results when computing share multiplications, the private secrets x and y is protected by the random values a and b, therefore independent. Since $\|\bar{\mathbf{g}}_j\|_2^2$ lies within a certain range for all possible model up-

dates (ideally equivalent to one), the value of $\|\bar{\mathbf{g}}_j\|_2^2$ is also independent of $\mathbf{g}_0, \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}$. The values of Σ_1 and $\mathbf{\Sigma}_2$ are protected by the random value λ. Regarding $\mathbf{\Sigma}_2/\Sigma_1$, we have $H(\mathbf{\Sigma}_2/\Sigma_1|\mathbf{g}, \mathbf{g}_0) = 0$ since $\mathbf{g} = \|\mathbf{g}_0\| \cdot \mathbf{\Sigma}_2/\Sigma_1$. We consider the MAC keys, all intermediate results during multiplication, the computed shares, $\lambda\Sigma_1$ and $\lambda\mathbf{\Sigma}_2$ together and denote them as $\boldsymbol{m}$, we have

$$\begin{aligned}
&H(M_f|\mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) \\
&= H(\boldsymbol{m}, \|\bar{\mathbf{g}}_i\|_2^2, \mathbf{\Sigma}_2/\Sigma_1|\mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) \\
&= H(\boldsymbol{m}, \|\bar{\mathbf{g}}_i\|_2^2 \mid \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) + H(\mathbf{\Sigma}_2/\Sigma_1 \mid \boldsymbol{c}, \|\bar{\mathbf{g}}_i\|_2^2, \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) \\
&= H(\boldsymbol{m}, \|\bar{\mathbf{g}}_i\|_2^2 \mid \mathbf{g}_0, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) = H(\boldsymbol{m}, \|\bar{\mathbf{g}}_i\|_2^2),
\end{aligned}$$

and $H(M_f \mid \mathbf{g}_0, \mathbf{g}_{[n]\setminus\mathcal{T}}, D_0, \boldsymbol{w}^{(g)}, \mathbf{g}) = H(\boldsymbol{m}, \|\bar{\mathbf{g}}_i\|_2^2)$ for the same reason. This concludes the proof of IT privacy against the federator. Hence, we conclude the proof of privacy.
(Byzantine-resilience) Byzantine users can present arbitrary incorrect results in any step during the computation. Particularly, they can (1) incorrectly normalize local model updates $\tilde{\mathbf{g}}_i$ in the normalization step, (2) compute the incorrect secret shares $\{\mathbf{g}_j[i]\}$ for other users' model updates in the secret sharing step, (3) present an incorrect computation result of $\{(\|\mathbf{g}_j\|_2^2)[i]\}$ during validation of the normalization step, (4) misbehave when computing the discriminator function, $\{(\lambda\Sigma_1)[i]\}$ and $\{(\lambda\mathbf{\Sigma}_2)[i]\}$, and send incorrect computation results to the federator in step D.
For Byzantine attacks in the first scenario, the misbehaving malicious users 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 the malicious users present the model updates with the largest squared $l2$-norm that will be accepted by the normalization validation, it only has limited impact since the ϵ is empirically set to be very small.
For Byzantine attacks in the remaining scenarios, the malicious behavior will be detected by requiring the users to perform the same computation on the corresponding MACs and through integrity check before reconstructing the computation result. Since users receive the MACs of each $\mathbf{r}_j[i]$, $\lambda[i]$ and Beaver triples $a[i]$, $b[i]$, $c[i]$ from the TTP and the federator receives all the corresponding MAC keys (α, β)s' from the TTP, the users and the federator can trust the correctness of the MAC value. According to equations (4.6), (4.7) and (4.8), after consuming Beaver triples on each multiplication during the computation, the computations of $\{\mathbf{g}_j[i]\}$, $\{(\|\mathbf{g}_j\|_2^2)[i]\}$, $\{(\lambda\Sigma_1)[i]\}$ and $\{(\lambda\mathbf{\Sigma}_2)[i]\}$ in the last three attack scenarios only involve linear operations on the secret shares and the corresponding MACs given by the TTP at the pre-processing phase, i.e. $\{\mathbf{r}_j[i]\}$, $\{\lambda[i]\}$ and $\{a[i]\}$, $\{b[i]\}$, $\{c[i]\}$. Therefore, the correctness of the computations entirely relies on

the integrity check of the information-theoretic one-time MAC [BDOZ11], 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 βs 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. $p \geq 2Nd^kq^{2k+1}+1$, 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. □

4.5 Choice of the Discriminator Function

In FLTrust [CFLG20], the ReLU function is chosen as the discriminator function, whose use in ByITFL and LoByITFL poses challenges. Most importantly, the ReLU function requires a comparison, which is difficult to compute under IT privacy, i.e., it inherently leaks information about the relative direction of local model updates with the federator model update. We first thought about approximating the ReLU function used in FLTrust by a polynomial. But it is only satisfactory when the degree k is greater than 6. We thus design our discriminator function based on the following core observation: As long as the function takes non-zero values in the interval between -1 and 0, the left tail is not required to be zero. If the Byzantine attackers present model updates resulting in cosine similarity around -1, we use the inverted vector (scaled by a negative trust score). Hence, such corrupt updates cannot harm the learning process beyond what corrupt model updates with a positive cosine similarity are capable of. Based on these observations, we find that it is not required to approximate the ReLU function by large-degree polynomials with high precision; 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 [LLTW23], 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 $k = 3$. The chosen degree-3 polynomial is given by $h(x) = 0.46897526x^3 + 0.56578977x^2 + 0.1860353x + 0.01363545$, and plotted in Figure 4.1 alongside with the degree-6 approximation polynomial and the ReLU function.

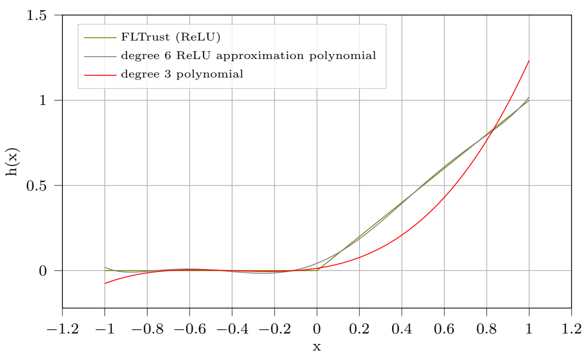

Figure 4.1: Comparison of different discriminator functions $h(x)$.

4.6 Complexity Analysis

Proposition 1. *ByITFL requires a user to communicate* $O(\frac{d}{M}n^3+n^4)$ *and the federator* $O(\frac{d}{m}n+n^2)$ *scalars. The computation cost is* $O((\frac{d}{M}n^3+n^4)\log^2 n\log\log n)$ *and* $O((\frac{d}{m}n+n^2)\log^2 n\log\log n)$*, respectively.*

Sketch Proof of Proposition 1. For each user sharing a single scalar, the computation complexity of encoding in LCC is $O(n\log^2 n\log\log n)$ [YLR+19] and that of ITVSS [BOGW88] is $O(n^2\log^2 n)$. Since the re-randomization step [GRR98] involves sub-sharing each secret share using VSS 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 counting arguments, here omitted for brevity.

□

Proposition 2. *LoByITFL requires a user to communicate* $O((d+k)n)$ *and the federator* $O((d+k)n^2)$ *scalars. The computation cost for each user and the federator is* $O((d+k)n)$ *and* $O((d+k)n^2\log^2 n\log\log n)$*, respectively.*

Remark 1. *When choosing a linear secret sharing scheme such as McEliece-Sarwate secret sharing [MS81] or Lagrange coded computing [YLR+19], that partition the model updates into* M *smaller sub-vectors, like ByzSecAgg and ByITFL did, the communication and computation complexity can be effectively optimized by reducing every* d *to* d/M*.*

Proof of Proposition 2. Our scheme is split into the pre-processing phase and the training phase. In the pre-processing phase, the TTP communicates $O((d+k)n)$ with each user and $O((d+k)n^2)$ with the federator for one iteration. The training phase is more complicated. The computation for each user sharing a scalar is, accoding to (4.6), the 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, i.e. the computation of consuming one Beaver triple, which only involves addition and scaling of the secret shares. For the federator, the computation complexity is when reconstructing the intermediate results during share multiplications and the multiplication 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 the counting arguments, which is omitted here for the sake of brevity. $\square$

In Table 4.2 and 4.3, we compare the communication and computation complexity of ByITFL and LoByITFL with respect to N, d and the partitioning parameter M to the previous solutions BREA and ByzSecAgg.

Table 4.2: Communication Complexity.

	Per-User	Federator
BREA [SGA20]	$O(dn+n^2)$	$O(dn+n^3)$
ByzSecAgg [JNMAC23]	$O(\frac{d}{M}n+n^2)$	$O(\frac{d}{M}n+n^3)$
ByITFL	$O(\frac{d}{m}n^3+n^4)$	$O(\frac{d}{m}n+n^2)$
LoByITFL	$O((d+k)n)$	$O((d+k)n^2)$

Table 4.3: Computation Complexity

	Per-User	Federator
BREA [SGA20]	$O(dn\log^2 n+dn^2)$	$O((dn+n^3)\log^2 n\log\log n)$
ByzSecAgg [JNMAC23]	$O(\frac{d}{M}n\log^2 n+\frac{d}{M}n^2)$	$O((\frac{d}{M}n+n^3)\log^2 n\log\log n)$
ByITFL	$O((\frac{d}{M}n^3+n^4)\log^2 n\log\log n)$	$O((\frac{d}{m}n+n^2)\log^2 n\log\log n)$
LoByITFL	$O((d+k)n)$	$O((d+k)n^2\log^2 n\log\log n)$

5 Experiments

We numerically demonstrate the convergence of the proposed schemes, i.e. ByITFL and LoByITFL, and compare to FedAvg and FLTrust. Our experiments are based on the implementation provided by [GMS+23] for our experiments. We consider MNIST [Den12] and CIFAR-10 [KH+09] equally distributed across $N = 40$ users. On 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 is chosen to be 0.1, and 0.01 for CIFAR-10.

We assume 25% of the users are Byzantine ($B = 10$) and perform either the Krum attack, trim attack, label flipping attack scaling attack or the adaptive attack. Krum attack and trim attack are untargeted local model poisoning attacks presented in [FCJG20], optimized for the aggregation rule Krum [BEMGS17] and Trimmed mean [YCKB18]. The label-flipping attack follows the same setting as in [FCJG20]. The scaling attack is equivalently known as the backdoor attack [BVH+20]. The adaptive attack is the attack presented in [FCJG20] and optimized for the proposed schemes. We design our adaptive attack by following the design of adaptive attack in FLTrust [CFLG20], but replace the discriminator function in FLTrust with the chosen degree-3 polynomial. As in [CFLG20], 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 $q = 1024$. As in [CFLG20], the size of the root dataset D_0 is 100. We set $\varepsilon = 0.02$ for the normalization validation. For i.i.d. MNIST and all attacks other than the adaptive attack, we plot the average results and standard deviation across 10 runs. Due to the large computation time, the results for the adaptive attack are solely based on 2 runs. In Table 5.1, it can be found that when plotted against FLTrust, the proposed schemes achieve comparably effective resilience against Krum attacks, trimmed attacks, label flipping attacks, and the adaptive tailored attacks on MNIST. Note that, while the adaptive attack on ByITFL and LoByITFL is optimized for the chosen degree 3 polynomial, the adaptive attack performed on FLTrust is optimized for the ReLU function. Hence, replacing the ReLU function with the appropriately tuned degree-3 polynomial does not incur additional vulnerabilities. Both schemes, however, are not able to counteract backdoor attacks, with our scheme resulting in an average attack success rate of 0.874 ± 0.276 and FLTrust of 0.617 ± 0.366 in i.i.d setting, as shown in Table 5.2. Similar results can be found for non-i.i.d MNIST and CIFAR-10, as indicated in Table 5.3 and Table 5.4, where each experiment is repeated ten times for non-i.i.d. MNIST and twice for CIFAR-10. We omit the results for the adaptive attack and the scaling attack on non-i.i.d. MNIST and CIFAR-10 as the adaptive attack is computationally too expensive and the scaling attack is indefensible.

Table 5.1: Testing accuracy comparison. We report the mean and standard deviation of 10 runs on i.i.d. MNIST. For reference, FedAvg under no attack achieves 0.962 ± 0.001.

	label-flipping	trim attack	Krum attack	adaptive attack
FLTrust	0.933 ± 0.008	0.891 ± 0.041	0.930 ± 0.005	0.913 ± 0.005
ours	0.940 ± 0.008	0.924 ± 0.046	0.925 ± 0.013	0.911 ± 0.008

Table 5.2: Testing accuracy and attack success rate comparison under scaling (backdoor) attack. We report the mean and standard deviation of 10 runs on i.i.d. MNIST.

	testing accuracy	attack success rate
FLTrust	0.918 ± 0.011	0.617 ± 0.366
ours	0.942 ± 0.010	0.874 ± 0.276

Table 5.3: Testing accuracy comparison. We report the mean and standard deviation of 10 runs on non-i.i.d. MNIST. For reference, FedAvg under no attack achieves 0.962 ± 0.001.

	label-flipping	trim attack	Krum attack
FLTrust	0.928 ± 0.005	0.916 ± 0.013	0.934 ± 0.004
ours	0.939 ± 0.006	0.911 ± 0.019	0.933 ± 0.008

Table 5.4: Testing accuracy comparison. We report the mean and standard deviation of 2 runs on CIFAR-10. For reference, FedAvg under no attack achieves 0.655 ± 0.005.

	label-flipping	trim attack	Krum attack
FLTrust	0.597 ± 0.002	0.633 ± 0.006	0.634 ± 0.005
ours	0.573 ± 0.006	0.634 ± 0.004	0.629 ± 0.005

Compute Resources All experiments in this section were conducted on an internal compute cluster of machines with CPUs of type Intel Xeon and AMD EPYC, GPUs of type Nvidia GTX 1080 Ti, RTX 4080 and RTX 4090, and between 64GB and 512GB of RAM. The overall computation time for all experiments is on the order of multiple days.

6 Conclusion

We introduced two schemes ByITFL and LoByITFL, two information-theoretically private and Byzantine-resilient federated learning schemes. ByITFL is the first Byzantine-resilient scheme for FL with full information-theoretic privacy and LoByITFL reduces the significant communication cost of ByITFL towards practicable applications by the use of a TTP during the pre-processing phase. We studied discriminator functions different from the originally proposed ReLU [CFLG20] that allow for IT private schemes without sacrificing the resiliency against Byzantine attacks. We prove the robustness and the privacy of our scheme, and experimentally compare the Byzantine-resilience aginst FLTrust [CFLG20]. While FLTrust, ByITFL and LoByITFL can effectively defend against label-flipping attacks, trim attacks, Krum attacks, and adaptive attacks individually tailored to the choice of the discriminator polynomial, we found that none of the schemes can resist a properly examined backdoor (scaling) attack. Improving our IT private scheme to also resist backdoor attacks while maintaining the strongest possible notion of privacy is left for future work.

Bibliography

[AHW$^+$17] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, “Privacy-preserving deep learning via additively homomorphic encryption,” *IEEE transactions on information forensics and security*, vol. 13, no. 5, pp. 1333–1345, 2017.

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

[AL17] G. Asharov and Y. Lindell, “A full proof of the bgw protocol for perfectly secure multiparty computation,” *Journal of Cryptology*, vol. 30, no. 1, pp. 58–151, 2017.

[BDOZ11] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, “Semi-homomorphic encryption and multiparty computation,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2011, pp. 169–188.

[Bea92] D. Beaver, “Efficient multiparty protocols using circuit randomization,” in *Advances in Cryptology—CRYPTO’91: Proceedings 11.* Springer, 1992, pp. 420–432.

[BEMGS17] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent,” *Advances in neural information processing systems*, vol. 30, 2017.

[Ben86] J. C. Benaloh, “Secret sharing homomorphisms: Keeping shares of a secret secret,” in *Conference on the theory and application of cryptographic techniques.* Springer, 1986, pp. 251–260.

[BIK$^+$17] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[BIMP$^+$24] Y. Ben-Itzhak, H. Möllering, B. Pinkas, T. Schneider, A. Suresh, O. Tkachenko, S. Vargaftik, C. Weinert, H. Yalame, and A. Yanai, “Scionfl: Efficient and robust secure quantized aggregation,” in *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2024, pp. 490–511.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness theorems for non-cryptographic fault-tolerant distributed computation,” in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC ’88. New York, NY, USA: Association for Computing Machinery, 1988, p. 1–10. [Online]. Available: https://doi.org/10.1145/62212.62213

[BVH$^+$20] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” in *International conference on artificial intelligence and statistics.* PMLR, 2020, pp. 2938–2948.

[CFLG20] X. Cao, M. Fang, J. Liu, and N. Z. Gong, “Fltrust: Byzantine-robust federated learning via trust bootstrapping,” *arXiv preprint arXiv:2012.13995*, 2020.

[DCL$^+$21] Y. Dong, X. Chen, K. Li, D. Wang, and S. Zeng, “FLOD: Oblivious defender for private byzantine-robust federated learning with dishonest-majority,” in *Computer Security – ESORICS 2021*, 2021, pp. 497–518.

[Den12] L. Deng, “The mnist database of handwritten digit images for machine learning research,” *IEEE Signal Processing Magazine*, 2012.

[DKL$^+$13] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, “Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits,” in *Computer Security–ESORICS 2013: 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings 18.* Springer, 2013, pp. 1–18.

[DPSZ12] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, “Multiparty computation from somewhat homomorphic encryption,” in *Annual Cryptology Conference.* Springer, 2012, pp. 643–662.

[DWL$^+$23] C. Dong, J. Weng, M. Li, J.-N. Liu, Z. Liu, Y. Cheng, and S. Yu, “Privacy-preserving and byzantine-robust federated learning,” *IEEE Transactions on Dependable and Secure Computing*, 2023.

[FCJG20] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to {Byzantine-Robust} federated learning," in *29th USENIX security symposium (USENIX Security 20)*, 2020, pp. 1605–1622.

[Fel87] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. IEEE, 1987, pp. 427–438.

[FMM+21] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllering, T. D. Nguyen, P. Rieger, A.-R. Sadeghi, T. Schneider, H. Yalame, and S. Zeitouni, "Safelearn: Secure aggregation for private federated learning," in *IEEE Security and Privacy Workshops (SPW)*, 2021, pp. 56–62.

[FYB18] C. Fung, C. J. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *arXiv preprint arXiv:1808.04866*, 2018.

[GBDM20] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?" *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 937–16 947, 2020.

[GMS+23] T. Gehlhar, F. Marx, T. Schneider, A. Suresh, T. Wehrle, and H. Yalame, "Safefl: Mpc-friendly framework for private and robust federated learning," 2023.

[GR+18] R. Guerraoui, S. Rouault *et al.*, "The hidden vulnerability of distributed learning in byzantium," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3521–3530.

[GRR98] R. Gennaro, M. O. Rabin, and T. Rabin, "Simplified vss and fast-track multiparty computations with applications to threshold cryptography," in *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, 1998, pp. 101–111.

[HKJ20] L. He, S. P. Karimireddy, and M. Jaggi, "Secure byzantine-robust machine learning," *arXiv preprint arXiv:2006.04747*, 2020.

[HLW+23] G. Hu, H. Li, T. Wu, W. Fan, and Y. Zhang, "Efficient byzantine-robust and privacy-preserving federated learning on compressive domain," *IEEE Internet of Things Journal*, 2023.

[HLX+21] M. Hao, H. Li, G. Xu, H. Chen, and T. Zhang, "Efficient, private and robust federated learning," in *Annual Computer Security Applications Conference*, 2021, pp. 45–60.

[JNMAC23] 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 e-prints*, pp. arXiv–2302, 2023.

[Kel20] M. Keller, “Mp-spdz: A versatile framework for multi-party computation,” in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 1575–1590.

[KH$^+$09] A. Krizhevsky, G. Hinton *et al.*, “Learning multiple layers of features from tiny images,” 2009.

[KMA$^+$21] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, “Advances and open problems in federated learning,” *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[KOS16] M. Keller, E. Orsini, and P. Scholl, “Mascot: faster malicious arithmetic secure computation with oblivious transfer,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 830–842.

[KPR18] M. Keller, V. Pastro, and D. Rotaru, “Overdrive: Making spdz great again,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2018, pp. 158–189.

[LLTW23] Z. Lu, S. Lu, X. Tang, and J. Wu, “Robust and verifiable privacy federated learning,” *IEEE Transactions on Artificial Intelligence*, 2023.

[MMM$^+$22] Z. Ma, J. Ma, Y. Miao, Y. Li, and R. H. Deng, “Shieldfl: Mitigating model poisoning attacks in privacy-preserving federated learning,” *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1639–1654, 2022.

[MMR$^+$17] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Artificial intelligence and statistics.* PMLR, 2017, pp. 1273–1282.

[MS81] R. J. McEliece and D. V. Sarwate, “On sharing secrets and reed-solomon codes,” *Communications of the ACM*, vol. 24, no. 9, pp. 583–584, 1981.

[MYL$^+$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 Transactions on Information Forensics and Security*, pp. 1–1, 2024.

[SGA20] J. So, B. Güler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2168–2181, 2020.

[Sha79] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[VXK21] R. K. Velicheti, D. Xia, and O. Koyejo, "Secure byzantine-robust distributed learning via clustering," *arXiv preprint arXiv:2110.02940*, 2021.

[XHEB24] Y. Xia, C. Hofmeister, M. Egger, and R. Bitar, "Byzantine-resilient secure aggregation for federated learning without privacy compromises," *arXiv preprint arXiv:2405.08698*, 2024.

[XKG19] C. Xie, S. Koyejo, and I. Gupta, "Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6893–6901.

[XLZ$^+$22] G. Xu, H. Li, Y. Zhang, S. Xu, J. Ning, and R. H. Deng, "Privacy-preserving federated deep learning with irregular users," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 1364–1381, 2022.

[XOWZiA23] M. Xhemrishi, J. Oestman, A. Wachter-Zeh, and A. G. i Amat, "Fedgt: Identification of malicious clients in federated learning with secure aggregation," *arXiv preprint arXiv:2305.05506*, 2023.

[YCKB18] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5650–5659.

[YLR$^+$19] 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 *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1215–1225.

[ZL24] Z. Zhang and Y. Li, "Nspfl: A novel secure and privacy-preserving federated learning with data integrity auditing," *IEEE Transactions on Information Forensics and Security*, 2024.

[ZLH19] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *Advances in neural information processing systems*, vol. 32, 2019.

[ZSWJ22] B. Zhao, P. Sun, T. Wang, and K. Jiang, "Fedinv: Byzantine-robust federated learning by inversing local model updates," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 9171–9179.

[ZTX+24] Y. Zhong, W. Tan, Z. Xu, S. Chen, J. Weng, and J. Weng, "Wvfl: Weighted verifiable secure aggregation in federated learning," *IEEE Internet of Things Journal*, 2024.