

ByzSecAgg: A Byzantine-Resistant Secure Aggregation Scheme for Federated Learning Based on Coded Computing and Vector Commitment

Tayyebeh Jahani-Nezhad, Mohammad Ali Maddah-Ali, *Fellow, IEEE*, and Giuseppe Caire, *Fellow, IEEE*

Abstract—In this paper, we propose **ByzSecAgg**, an efficient secure aggregation scheme for federated learning that is resistant to Byzantine attacks and privacy leakages. Processing individual updates to manage adversarial behavior, while preserving privacy of data against colluding nodes, requires some sort of secure secret sharing. However, the communication load for secret sharing of long vectors of updates can be very high. In federated settings, where users are often edge devices with potential bandwidth constraints, excessive communication overhead is undesirable. **ByzSecAgg** solves this problem by partitioning local updates into smaller sub-vectors and sharing them using ramp secret sharing. However, this sharing method does not admit bi-linear computations, such as pairwise distance calculations, which are needed for distance-based outlier-detection algorithms, and effective methods for mitigating Byzantine attacks. To overcome this issue, each user runs another round of ramp sharing, with a different embedding of data in the sharing polynomial. This technique, motivated by ideas from coded computing, enables secure computation of pairwise distance. In addition, to maintain the integrity and privacy of the local update, **ByzSecAgg** also uses a vector commitment method, in which the commitment size remains constant (i.e., does not increase with the length of the local update), while simultaneously allowing verification of the secret sharing process. In terms of communication load, **ByzSecAgg** significantly outperforms the related baseline scheme, known as **BREA**.

Index Terms—Federated learning, Secure aggregation, Coded computing, Secure coded computing, Byzantine-robustness, Vector commitment, Secure matrix multiplication.

I. INTRODUCTION

FEDERATED LEARNING (FL) is an emerging distributed learning framework that allows a group of distributed users (e.g., mobile devices) to collaboratively train a global model with their local private data, without sharing the data [1]–[3]. Specifically, in an FL system with a central server and several users, during each training iteration, the server sends the current state of the global model to the users. Receiving the global model, each user then calculates a local update with its local data, and sends the local update to the server. By aggregating the local updates, the server can

The work of T. Jahani-Nezhad and G. Caire was supported by the Gottfried Wilhelm Leibniz-Preis 2021 of the German Science Foundation (DFG). The work of M. A. Maddah-Ali has been partially supported by the National Science Foundation under Grant CCF-2348638.

T. Jahani-Nezhad and G. Caire are with the Department of Electrical Engineering and Computer Science, Technische Universität Berlin, 10587 Berlin, Germany (e-mail: t.jahani.nezhad, caire@tu-berlin.de).

M. A. Maddah-Ali is with the Department of Electrical and Computer Engineering, University of Minnesota Twin Cities, MN 55455 USA (e-mail: maddah@umn.edu)

update the global model for the next iteration. While the local datasets are not directly shared with the server, several studies have shown that a curious server can launch model inversion attacks to reveal information about the training data of the individual users from their local updates [4], [5]. Therefore, the key challenge to protect users' data privacy is to design *secure aggregation* protocols, which allow the aggregation of the local updates to be computed without revealing each individual data. Moreover, as some users may randomly drop out of the aggregation process (due to low batteries or unstable connections), the server should be able to robustly recover the aggregated local updates of the surviving users in a privacy-preserving manner.

As such motivated, a secure aggregation protocol **SecAgg** is proposed in [6]. In **SecAgg**, the local updates are masked before being sent to the server, using private and shared random vectors, such that the shared parts can be canceled out when aggregated. One of the major challenges for **SecAgg** is the communication load, which grows quadratically with the number of users. There has been a series of works aiming to improve the communication efficiency of **SecAgg** (see, e.g., [7]–[13]). For instance, **SwiftAgg+**, recently proposed in [14], significantly reduces the communication overheads without any compromise on worst-case information-theoretic security, and achieves optimal communication loads within diminishing gaps for secure aggregation.

In [7]–[14], the *honest-but-curious* model for secure aggregation is considered in which users correctly follow the protocol but some of them may try to gather information about other users' private information, potentially through collusion. However, some studies have also addressed the *malicious* model in which Byzantine adversaries have more capabilities and may run poisoning attacks, i.e., manipulate their inputs to change the outcome of the aggregation [15]–[17]. The problem of robustness against Byzantine adversaries in distributed and federated learning is well-researched in two categories: distance-based methods [18]–[21] and validation data-based methods [22], [23]. However, those defenses come with some potential privacy issues, as they require the server to process the local model updates.

As a Byzantine resistant secure aggregation scheme, in [24], **BREA** is proposed in which users share their model updates with others using Shamir's secret sharing and then calculate the pairwise distance between the received shares. The server then uses this information to identify and exclude Byzantine users from the aggregation process. **BREA** suffers

from two major shortcomings: (1) to verify secret sharing, in BREA every single element of data is committed separately, (2) the secret sharing is not designed efficiently. As a result, the loads of commitment and sharing grow linearly with the *aggregated size* of all *local updates*. An alternative approach is proposed in [25], which involves grouping users anonymously and randomly into subgroups with a hierarchical tree structure. The aggregation method similar to SecAgg is performed within each subgroup, and then the process repeats at the next level of subgroups toward the root. The anonymity and randomness of the grouping reduce the probability of successful Byzantine attacks, as attackers do not know which subgroup the compromised device belongs to. However, in that approach, smaller subgroups result in reduced privacy for participants, while larger subgroups make it easier for the Byzantine attacker to hide their malicious model among the honest ones. In [26], a similar idea is proposed, which involves repeating the random grouping multiple times to save more updates of benign users through robust aggregation. However, that approach increases the communication loads and also reveals some information about the local updates. In [27], ELSA is proposed, which utilizes a distributed trust setup where two servers in separate trust domains interact and exchange information with one another. Each iteration begins with two servers selecting a group of users to participate and sharing the current global model with them. These users then use the cryptography module to securely share their local updates with the two servers. The servers then use an interactive protocol to combine these updates and find the aggregation.

One notable approach that has gained attention is the utilization of concepts from coded computing to enhance the security and efficiency of distributed computing algorithms. Coded computing, originally developed in the context of distributed storage and computation, offers possibilities for optimizing communication and computation tasks in distributed systems [28]–[34]. Additionally, the concept of secure coded computation has been developed to address the challenge of preserving data privacy in large-scale computations, such as matrix multiplication, ensuring that sensitive information remains protected during computation [35]–[39]. Furthermore, by leveraging ideas from error-correcting codes and network coding, coded computing techniques enable the efficient processing of data and computation while mitigating the impact of Byzantine behaviors in computations [40]–[42].

In this paper, we propose ByzSecAgg, a novel single-server Byzantine-robust secure aggregation scheme in a federated setting. In addition to providing Byzantine-robustness and privacy preservation, ByzSecAgg addresses the issue of high communication load. This is particularly crucial in federated learning settings, where users are edge devices such as mobile phones. It is common for these devices to have bandwidth limitations or be unable to handle high communication loads. ByzSecAgg draws inspiration from the integration of techniques from diverse fields, including coded computing, cryptography and outlier detection algorithms. The proposed scheme is robust against user dropouts, collusion and Byzantine adversarial attacks, and it involves the following

steps:

- Each user partitions its local update vector into smaller sub-vectors and broadcasts constant-size commitments of them, regardless of the size of the local update. This ensures that the sub-vectors can be proven to be unchanged during the scheme, while still keeping them hidden.
- Users securely share these sub-vectors with others using ramp secret sharing, and these shares can be verified using the commitments.
- Inspired by coded computing techniques, each user creates another polynomial function to send additional shares of their sub-vectors to other users, allowing for the computation of pairwise distance of shares.
- The server then uses these pairwise distances of shares to decode the pairwise distances of the true local updates. Using these distances, the server employs a distance-based outlier detection algorithm to select a subset of users for aggregation.
- Finally, the server obtains the aggregation of local updates of the selected users by communicating with the users who locally aggregate the secret shares of the model updates that belong to the selected users.

ByzSecAgg ensures the privacy of individual local models by performing computations using secret shares, which prevent users from learning the true values of local updates. Additionally, the server is not able to obtain any information about the local models beyond the aggregation and the pairwise distances which are strictly required by the outlier detection method. Furthermore, the commitments in ByzSecAgg are binding and computationally hiding. This means that users cannot obtain any information from the commitments, but they are still able to verify the authenticity of the messages received from others and the validity of the shares.

Table I compares the baseline framework BREA in [24] and ByzSecAgg to achieve a certain precision, in terms of communication loads, presented as the number of symbols from the underlying finite field. For a fair comparison, we consider three metrics: the server communication load, the per-user communication load and the commitments size. Server communication indicates the total size of all messages which are sent or received by the server, per-user communication denotes the total size of all messages, that are sent by each user, and the size of the commitments represents the size of commitments made by users required for message verification. Compared with the baseline, as shown in Table I in the system consisting of N users, ByzSecAgg reduces server and per-user communication loads, as well as significantly decreasing commitment size. The proposed scheme allows for K , a design parameter, to be in the range $[1 : \frac{N-D+1}{2} - A - T]$, where D, T and A denote the maximum number of dropouts, colluding users, and Byzantine adversaries, respectively, for which the scheme is designed. Based on system parameters N and L , which is the size of each local update, the optimal value of K for minimizing communication loads can be chosen. For instance, for large L , in an extreme case, we can choose $K = \mathcal{O}(N)$, which significantly decreases the server and per-user communication loads. Note that, in the special case when

TABLE I

COMMUNICATION LOADS OF BYZANTINE-ROBUST SECURE AGGREGATION FRAMEWORKS IN FEDERATED LEARNING. HERE N IS THE TOTAL NUMBER OF USERS, L IS THE SIZE OF THE LOCAL UPDATES, T IS THE NUMBER OF COLLUDING USERS, D IS THE NUMBER OF DROPOUTS, A IS THE NUMBER OF BYZANTINE ADVERSARIES. IN **ByzSecAgg**, PARAMETER $K \in [1 : \frac{N-D+1}{2} - A - T]$ CAN BE CHOSEN BASED ON THE NETWORK. NOTE THAT IN THIS TABLE, WE HIGHLIGHTED THE PARAMETER L TO EMPHASIZE THE PERFORMANCE OF **ByzSecAgg** IN REDUCING COMMUNICATION LOAD, AS L IS TYPICALLY MUCH LARGER THAN N IN REAL-WORLD SCENARIOS.

Approach	Server communication	Per-user communication	Commitments size
BREA [24]	$(2A + T + 1)L + (T + A + \frac{1}{2})N(N - 1)$	$NL + \frac{N(N-1)}{2}$	TNL
ByzSecAgg	$(1 + \frac{2A+T}{K})L + (T + A + K - \frac{1}{2})N(N - 1)$	$\begin{cases} NL + \frac{3N(N-1)}{2}, & \text{if } K = 1 \\ \frac{2N}{K}L + \frac{3N(N-1)}{2}, & \text{if } K > 1 \end{cases}$	$\begin{cases} N(3T + 1), & \text{if } K = 1 \\ N(3K + 4T - 2), & \text{if } K > 1 \end{cases}$

$K = 1$, the server communication load in **ByzSecAgg** is equivalent to that in BREA, and the per-user communication load in **ByzSecAgg** requires only $N(N - 1)$ extra symbols compared to BREA to achieve a higher level of privacy. However, unlike BREA, the commitment size in **ByzSecAgg** remains constant, regardless of the size of the local updates.

Notation For $n \in \mathbb{N}$ the notation $[n]$ represents set $\{1, \dots, n\}$. In addition, for $n_1, n_2 \in \mathbb{Z}$ the notation $[n_1 : n_2]$ denotes the set $\{n_1, \dots, n_2\}$. Furthermore, the cardinality of set \mathcal{S} is denoted by $|\mathcal{S}|$. In addition, we denote the difference between two sets \mathcal{A} and \mathcal{B} as $\mathcal{A} \setminus \mathcal{B}$, which represents the set of elements belonging to \mathcal{A} but not to \mathcal{B} . In addition, $\mathbb{E}[X]$ and $H(X)$ refer to the expected value and the entropy of random value X respectively. $\Pr(A)$ is the probability of event A .

II. PROBLEM FORMULATION

We consider the Byzantine-robust secure aggregation problem, for a federated learning system, consisting of a server and N users. The objective of the server is to train a global model $\mathbf{w}_g \in \mathbb{R}^L$, with dimension $L \in \mathbb{N}$, using the data held at users, by minimizing a global cost function $\mathcal{L}(\mathbf{w}_g)$. In round t of the training phase, the server broadcasts the global model $\mathbf{w}_g^{(t)} \in \mathbb{R}^L$ to all users. Then each user n , $n \in [N]$ computes a private local update $\mathbf{w}_n \in \mathbb{R}^L$ based on its private local dataset. In this paper, we focus on perfect secure aggregation schemes, which rely on operations in a finite field to protect the privacy of the local updates [6]–[14]. Consider that each user employs an element-wise stochastic quantization method that involves a rounding function $Q : \mathbb{R} \rightarrow \mathbb{R}$ and a mapping function $\Lambda : \mathbb{R} \rightarrow \mathbb{F}$, which maps the integer numbers to elements of a finite field \mathbb{F} . The finite field is selected to be sufficiently large so that during the process of aggregation, there is no risk of encountering the boundary, thereby preventing potential issues.

User n also has a collection of local random variables \mathcal{Z}_n , whose elements are selected uniformly at random from \mathbb{F}^L , and independently of each other and of the local updates. It is assumed that each user can directly communicate with the server. Let $\mathbf{X}_n^{(L)} \in \mathbb{F}^* \cup \{\perp\}$ denote the message sent by user n to the server. In addition, let $\mathbf{M}_{n \rightarrow n'}^{(L)} \in \mathbb{F}^* \cup \{\perp\}$ denote the message that user n sends to user n' in the finite field \mathbb{F} in the algorithm. The null symbol \perp represents the case where no message is sent. Here $\mathbb{F}^* = \cup_{\ell \in \mathbb{N}} \mathbb{F}^\ell$. The message $\mathbf{M}_{n \rightarrow n'}^{(L)}$ is a function of \mathbf{w}_n , \mathcal{Z}_n . We denote the corresponding encoding function by $\phi_{n \rightarrow n'}^{(L)}$. Similarly, $\mathbf{X}_n^{(L)}$ is a function of \mathbf{w}_n , \mathcal{Z}_n ,

and the messages that user n has received from other users. We denote the corresponding encoding function by $\varphi_n^{(L)}$. Let \mathcal{U}_s refer to a subset of users selected by the server and their local updates are used for aggregation. Let $\mathcal{X}_s = \{\mathbf{X}_n^{(L)}\}_{n \in s}$ represent the set of messages the server receives from a subset of users \mathcal{S} , where $|\mathcal{S}| \leq N$. The received messages from the users are decoded by the server using the decoding function $\psi^{(L)}$ in order to use them for the aggregation process. In this setting, we assume that a subset of users $\mathcal{D} \subset [N]$ drops out, i.e., stay silent (or send \perp to other users and the server) during the protocol execution. We denote the maximum number of dropped-out users as $D \in \mathbb{N}$. We also assume that some of the users are curious and might collude to gain information about the local updates of the other users. Assume that the maximum number of colluding users is denoted by $T \in \mathbb{N}$. Note that the identities of dropouts and colluding users are not known beforehand.

A. Threat Model

We assume untargeted poisoning attacks, particularly model poisoning attacks which aim to reduce the effectiveness of the global model or prevent its convergence by directly modifying the local updates and selecting malicious parameters before sending them to other nodes (Type-1) [15]–[17]. We assume that the attacker can compromise at most $A \in \mathbb{N}$ benign users, and arbitrarily manipulate the local updates sent from these users. We also consider that the attack is a probabilistic polynomial time (PPT) algorithm with respect to a security parameter $\kappa \in \mathbb{N}$ [43]. It means that the attacker can run an algorithm within polynomial time and uses probabilistic methods to try to break the security of the system with a given security parameter κ . In the following, we refer to these compromised users as Byzantine adversaries. Another adversarial behavior of the Byzantine users is that they might send messages to other nodes, inconsistent with the protocol, requiring verification methods to address this issue (Type-2). Note that the sets of colluding and adversarial users are not necessarily disjoint, but the problem formulation is stated in the general form.

B. Security Model

To mitigate the threat posed by model poisoning attacks, a robust aggregation rules $\Omega : (\mathbb{F}^*)^{|\mathcal{S}|} \rightarrow \mathbb{F}^L$ as a defense strategy is employed to address Type-1 adversarial behavior, where \mathcal{S} represents the subset of users that send their messages

to the server. This function represents both the aggregation rule and the user selection method utilized by the server to identify outliers based on the received messages and then eliminate them from the aggregation process. The server updates the global model by applying function Ω to the decoded received messages, as

$$\mathbf{w}_g^{(t+1)} = \mathbf{w}_g^{(t)} - \delta_t \Lambda^{-1} \left(\Omega(\psi^{(L)}(\mathbf{X}_n^{(L)}, n \in \mathcal{S})) \right),$$

where δ_t is the learning rate at round t and $\Lambda^{-1} : \mathbb{F} \rightarrow \mathbb{R}$ is a demapping function.

In addition, to enable verification of the messages in this adversarial system to protect against Type-2 adversarial behavior, a vector commitment scheme is used. A vector commitment scheme is a cryptographic primitive that enables a user to commit to a vector with the following desirable features. It is computationally infeasible to determine the committed vector from the commitment value. It is also computationally infeasible to find a different vector that maps to the same commitment value. More precisely, a vector commitment scheme $\text{VC}=(\text{Setup}, \text{Commit}, \text{Witness}, \text{Verify})$ includes the following components:

- **Setup:** $1^\kappa \rightarrow (\text{pp}, \text{sp})$: this protocol is run by a trusted or distributed authority at the beginning to take security parameter κ and generates some public parameters (pp) and some local secret parameters (sp).
- **Commit:** $(\text{pp}, \nu) \rightarrow C$: this algorithm takes vector ν as input and outputs a commitment C .
- **Witness:** $(\text{pp}, \nu) \rightarrow \omega$: this algorithm takes vector ν as input and computes a witness ω .
- **Verify:** $(\text{pp}, C, \omega) \rightarrow b \in \{\text{True}, \text{False}\}$: this algorithm takes a witness ω and a commitment C as input and returns either **True** or **False** based on the validity of the witness.

C. The Goals

A *Byzantine-robust secure aggregation scheme* consists of the encoding functions $\phi_{n \rightarrow n'}^{(L)}, \varphi_n^{(L)}, n, n' \in [N]$, the decoding function $\psi^{(L)}$, the aggregation rule Ω , and the vector commitment scheme VC such that the following conditions are satisfied:

1. Correctness:

- **Correctness of the Commitments:** For an honest user n with quantized local update $\bar{\mathbf{w}}_n$ of data and a commitment $C_n = \text{Commit}(\text{pp}, \bar{\mathbf{w}}_n)$, the created witness $\omega_n = \text{Witness}(\text{pp}, \bar{\mathbf{w}})$ successfully satisfies $\text{Verify}(\text{pp}, C_n, \omega_n) = \text{True}$.
- **Correctness of the Final Result:** Despite the presence of at most D dropouts, A Byzantine adversaries, and T colluding users, the server maintains the capability to recover $\bar{\mathbf{w}} \triangleq \frac{1}{|\mathcal{U}_s|} \sum_{n \in \mathcal{U}_s} \bar{\mathbf{w}}_n$ which must be the outcome of $\Omega(\psi^{(L)}(\mathbf{X}_n^{(L)}, n \in \mathcal{S}))$.

2. Robustness against Byzantine Adversaries:

- **Commitment Binding:** The algorithm **Commit** should create a binding and deterministic commitment to the data.

Formally, for attacker \mathcal{A} who can simulate any user, it must hold

$$\Pr \left((\text{pp}, \text{sp}) \leftarrow \text{Setup}(1^\kappa), (\nu, \nu') \leftarrow \mathcal{A}(\text{pp}, \text{sp}) : \nu \neq \nu' \wedge \text{Commit}(\nu) = \text{Commit}(\nu') \right) \leq \epsilon(\kappa),$$

where function $\epsilon(\cdot)$ is a *negligible* function, which means for all $c > 0$ there exists a k_c such that for all $k > k_c$ we have $\epsilon(k) < \frac{1}{k^c}$. In short, this means that a Byzantine adversary committer can present two distinct values of ν with the same commitment C with vanishing probability.

- **Global Model Resiliency:** The aggregation rule Ω should prioritize global model resiliency, even in the presence of up to A Byzantine adversaries (Type-1), by ensuring that at the end of each iteration, the output of the aggregation rule remains close to the true gradient. Formally, at each iteration of the training process, for the true gradient vector $\mathbf{g} \triangleq \nabla \mathcal{L}(\mathbf{w}_g^{(t)})$, the output of the aggregation rule, i.e., $\mathbf{g}_{\text{GAR}} \triangleq \Omega(\psi^{(L)}(\mathbf{X}_n^{(L)}, n \in \mathcal{S}))$ must satisfy a well-defined closeness criterion $\xi(\mathbf{g}_{\text{GAR}}, \mathbf{g}, N, A)$.
- **Malicious Computation Results:** The general scheme should maintain robustness against any malicious computation results produced by Byzantine adversaries (Type-2) that may be sent to the server and other users during each step of the scheme.

3. Privacy Constraint:

- **Hiding Property:** If a proof (pp, C_n) and a witness ω_n for quantized local update $\bar{\mathbf{w}}_n, n \in [N]$, are given and the verification $\text{Verify}(\text{pp}, C_n, \omega_n)$ returns **True**, no user can figure out the value of $\bar{\mathbf{w}}_n$ with non-negligible probability.
- **Privacy of Individual Local Updates:** The privacy constraint ensures that no group of up to T colluding users can extract any information about the local models of other honest users. In addition, after receiving \mathcal{X}_s , the server should not gain any information about local updates of the honest users, beyond the aggregation of them, and beyond what is strictly required by the user selection (outlier detection) method—specifically, the pairwise distances of local updates used in this work.

For a Byzantine-robust secure aggregation scheme satisfying the above conditions, we define the average per-user communication load and the server communication load as follows:

Definition 1 (Average per-user communication load). The average per-user communication load, denoted by R_{user} , is defined as the aggregated size of all messages sent by users, i.e.,

$$R_{\text{user}} = \frac{1}{N} \sum_{\substack{n, n' \in [N], \\ n' \neq n}} (H(\mathbf{M}_{n \rightarrow n'}^{(L)}) + H(\mathbf{X}_n^{(L)})),$$

where the base of the $\log(\cdot)$ function in the definition of the entropy function $H(\cdot)$ is the size of the finite field.

Definition 2 (Server communication load). The server communication load, denoted by R_{server} , is defined as the aggre-

gated size of all messages received by the server, i.e.,

$$R_{\text{server}} = \sum_{n \in [N]} H(\mathbf{X}_n^{(L)}).$$

In this paper, we propose **ByzSecAgg**, a scheme for Byzantine-robust secure aggregation in a single server federated setting to meet the aforementioned conditions. In **ByzSecAgg**, each user partitions its local update vector of length L into K smaller sub-vectors of length $\frac{L}{K}$ and broadcasts constant-size commitments of them. Users then use ramp sharing [44] to share the sub-vectors with others, which can be verified using the commitments. This method allows for a major reduction in communication loads, however, it makes computing the pairwise distances between the local updates and removing the outliers challenging. To address this issue, we use techniques inspired by coded computing, where each user runs another ramp sharing, where data vectors are embedded in the coefficients differently. These two sets of shares admit the computation of the pairwise distances in a very efficient way. The server then receives the pairwise distances of the shares, which are used to recover the pairwise distances of the local updates. These distances are employed in the multi-Krum algorithm [18] as the outlier detection method, to select m users for aggregation. Finally, communication with the server is required so that the server can obtain the aggregation of local updates of the selected users. To be able to verify that users follow the sharing protocol correctly and also not change their data in the second round of sharing, we need to be able to verify it against some commitment. We suggest using some linear commitment scheme [45] that, unlike many existing solutions, the size of the commitment remains constant and does not grow with the size of data.

III. PRELIMINARIES

In order to develop a secure aggregation scheme that is resilient to Byzantine adversarial behavior, particularly of Type-1, it is essential to employ a robust aggregation method. Specifically, in this paper, we leverage the multi-Krum algorithm proposed in [18]. Note that any alternative aggregation methods that are robust against Type-1 adversarial attacks and are based on Euclidean pairwise distances of the local updates can also be employed in **ByzSecAgg**. However, depending on the structure of each method, the proposed scheme might require some modifications.

A. Multi-Krum Algorithm

In the distributed stochastic gradient descent (SGD) problem, and in the presence of Byzantine adversaries (Type-1), using the average of all users' gradients to update the model parameters is not robust since a single Byzantine user can cause an arbitrarily large error in the update. To handle this, the server must use a gradient aggregation rule (GAR) Ω that is resistant to malicious gradients that may be produced by up to A Byzantine adversaries. At iteration, t , assume that each honest user n calculates an estimate $\mathbf{w}_n^{(t)} = \mathbf{G}(\mathbf{w}_g^{(t)}, \zeta_n^{(t)})$ of the gradient of the cost function \mathcal{L} , where $\zeta_n^{(t)}$ is a random variable representing the sample or mini-batch of

samples drawn from user n 's dataset, and $\mathbf{w}_g^{(t)}$ is the global model parameter received from the server. On the other hand, Byzantine adversarial users may send some arbitrary vector to the server (Type-1 adversarial attack). One method to measure resilience against such Byzantine users is through the concept of (γ, A) -Byzantine resilience, as introduced and supported by convergence theories in [18].

Definition 3 ((γ, A)-Byzantine Resilience [18]). Consider N vectors $\mathbf{w}_1, \dots, \mathbf{w}_N \in \mathbb{R}^L$ which are gradient vectors received by the server from N users. If user i is non-Byzantine, then \mathbf{w}_i is independent identically distributed random vector, $\mathbf{w}_i \sim \mathbf{G}(\mathbf{w}_g^{(t)}, \zeta_i^{(t)})$, with $\mathbb{E}[\zeta_i^{(t)} \mathbf{G}] = \nabla \mathcal{L}(\mathbf{w}_g^{(t)})$. Vector \mathbf{w}_i for a Byzantine user can be any arbitrary vector. For any angular value $0 \leq \gamma < \pi/2$ and any integer $0 \leq A \leq N$ denoting the maximum number of Byzantine adversaries, a gradient aggregation rule (GAR) Ω is called (γ, A) -Byzantine resilient if vector $\mathbf{w}_{\text{GAR}} \triangleq \Omega(\mathbf{w}_1, \dots, \mathbf{w}_N)$ satisfies the following two conditions:

- 1) $\langle \mathbb{E}[\mathbf{w}_{\text{GAR}}], \mathbf{g} \rangle \geq (1 - \sin \gamma) \|\mathbf{g}\|^2 > 0$, where \mathbf{g} is the true gradient $\mathbf{g} \triangleq \nabla \mathcal{L}(\mathbf{w}_g^{(t)})$.
- 2) For $r \in \{2, 3, 4\}$, $\mathbb{E}[\|\mathbf{w}_{\text{GAR}}\|^r]$ is upper bounded by

$$\mathbb{E}[\|\mathbf{w}_{\text{GAR}}\|^r] \leq c \sum_{r_1 + \dots + r_{N-A} = r} \mathbb{E}[\|\mathbf{G}\|^{r_1}] \dots \mathbb{E}[\|\mathbf{G}\|^{r_{N-A}}],$$

where c is a generic constant and r_1, \dots, r_{N-A} are non-negative integers.

In this definition, Condition 1 ensures that the angle between the true gradient \mathbf{g} and the output \mathbf{w}_{GAR} is small enough and on average the output is in the same direction as the true gradient. Condition 2 ensures that the second, third, and fourth-order moments of the output of the aggregation rule are bounded by a linear combination of terms $\mathbb{E}[\|\mathbf{G}\|^{r_1}] \dots \mathbb{E}[\|\mathbf{G}\|^{r_{N-A}}]$. This is generally necessary to ensure the convergence of the SGD algorithm.

Krum algorithm [18] is a gradient aggregation rule that is resilient to the presence of A Byzantine adversaries in a distributed system consisting of N users as long as $N > 2A + 2$. The Krum aggregation algorithm returns the gradient computed by the user with the lowest score, which is determined by considering the $N - A - 2$ closest gradients to that user's gradient. More precisely, let $\mathbf{w}_1, \dots, \mathbf{w}_N$ be the gradients received by the server. Krum algorithm assigns to each vector \mathbf{w}_i score $s(i) \triangleq \sum_{j: i \rightarrow j} \|\mathbf{w}_i - \mathbf{w}_j\|^2$, where $i \rightarrow j$ denotes that \mathbf{w}_j belongs to the $N - A - 2$ closest vector (in terms of squared distance) to \mathbf{w}_i for any $i \neq j$. The output of Krum algorithm is $\text{KR}(\mathbf{w}_1, \dots, \mathbf{w}_N) = \mathbf{w}_{i^*}$, where i^* is the gradient with the lowest score, i.e., for all i we have $s(i^*) \leq s(i)$.

Lemma 1 ([18]). Consider i.i.d. random vectors $\mathbf{w}_1, \dots, \mathbf{w}_N \in \mathbb{R}^L$ such that $\mathbf{w}_i \sim \mathbf{G}(\mathbf{w}_g, \zeta_i)$, with $\mathbb{E}[\mathbf{G}(\mathbf{w}_g, \zeta_i)] = \nabla \mathcal{L}(\mathbf{w}_g)$. Define $\sigma^2(\mathbf{w}_g) \triangleq \frac{1}{L} \mathbb{E}[\|\mathbf{G}(\mathbf{w}_g, \zeta_i) - \nabla \mathcal{L}(\mathbf{w}_g)\|^2]$. Let $\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_A$ be any A random vectors that may depend on the \mathbf{w}_i 's. If $N > 2A + 2$ and $\eta(N, A) \sqrt{L} \sigma < \|\nabla \mathcal{L}(\mathbf{w}_g)\|$, where $\eta(N, A)$ is defined as

$$\eta(N, A) \triangleq \sqrt{2 \left(N - A + \frac{A(N - A - 2) + A^2(N - A - 1)}{N - 2A - 2} \right)},$$

then the Krum algorithm is (γ, A) -Byzantine resilient, where $0 \leq \gamma < \frac{\pi}{2}$ is defined by $\sin \gamma = \frac{\eta(N, A)\sqrt{L}\sigma}{\|\nabla \mathcal{L}(\mathbf{w}_g)\|}$.

The convergence analysis of the SGD using Krum algorithm is presented in [48]. The proof relies on certain conditions regarding the learning rates and the gradient estimator. In addition, in that paper, a stronger variant of Krum algorithm is proposed called *multi-Krum*. In multi-Krum, $m \in [N]$, $m < N - 2A - 2$, gradient vectors $\mathbf{w}_{i_1^*}, \dots, \mathbf{w}_{i_m^*}$ are selected that have the lowest scores and the output of the algorithm is the average of the selected vectors, i.e., $\frac{1}{m} \sum_{k=1}^m \mathbf{w}_{i_k^*}$.

Note that although multi-Krum offers theoretical robustness against Type-1 adversarial attacks, it, like other Byzantine-robust methods, remains vulnerable in certain scenarios. These include sophisticated poisoning strategies proposed in [46], [47], attacks tailored for non-IID data distributions as discussed in [48], and attacks based on techniques such as Projected Gradient Descent (PGD), as highlighted in [49]. However, these attacks fall beyond the scope and assumptions of this paper.

IV. THE PROPOSED SCHEME

In this section, we explain *ByzSecAgg* in detail. Consider a federated learning system with one server and N users. The scheme is designed to handle a maximum number of T colluding users, a maximum number of D dropout users, and a maximum number of A Byzantine users, as defined in Section II. The adversaries are assumed to be probabilistic polynomial time (PPT) algorithms with respect to the security parameter κ . The sets of colluding users, dropouts, and adversaries are not known beforehand.

User n has a local update $\mathbf{w}_n \in \mathbb{R}^L$. We focus on perfect secure aggregation schemes, which rely on operations in a finite field to protect the privacy of the local updates [6]–[14]. We choose a finite field, $GF(p)$ denoted by \mathbb{F}_p , for some prime number p which is large enough. User n samples vectors $\mathcal{Z}_n = \{\mathbf{z}_{n,j}, j \in [T]\}$ and $\bar{\mathcal{Z}}_n = \{\bar{\mathbf{z}}_{n,j}, j \in [T]\}$ uniformly at random from $\mathbb{F}_p^{\frac{L}{K}}$, for some parameter $K \in \mathbb{N}$. In addition, user n samples random scalars $\mathcal{R}_n = \{r_{n,i}^{(j)}, i \in [2(K+T)-1] \setminus \{K\}, j \in [N]\}$ uniformly at random from \mathbb{F}_p .

Each user takes the following steps:

1) **Quantization:** User n converts its local update vector $\mathbf{w}_n \in \mathbb{R}^L$ in real numbers to vector $\bar{\mathbf{w}}_n \in \mathbb{F}_p^L$ in finite field. This conversion allows for the use of finite field operations, which play a crucial role in protecting the privacy of the local updates. To achieve this, each user first applies the stochastic rounding function in [24], [50] element-wise as follows

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

where integer $q \geq 1$ is the number of quantization levels and $\lfloor x \rfloor$ is the largest integer that is less than or equal to x . Note that the rounding function is unbiased, i.e., $\mathbb{E}[Q_q(x)] = x$. In addition, to represent negative values in the finite field, a

mapping function is needed. The quantized version of the local update of user n denoted by $\bar{\mathbf{w}}_n$ is defined as follows

$$\bar{\mathbf{w}}_n \triangleq \Lambda\left(qQ_q(\mathbf{w}_n)\right), \quad (2)$$

where $\Lambda : \mathbb{R} \rightarrow \mathbb{F}_p$ is a mapping function that is applied element-wise and defined as

$$\Lambda(x) = \begin{cases} x & \text{if } x \geq 0, \\ x + p & \text{if } x < 0. \end{cases} \quad (3)$$

It should be noted that in this step, any rounding function that ensures the convergence of the model can be utilized. There are no restrictions on selecting the rounding function in the proposed method.

2) **Partitioning the local updates:** User n partitions its quantized local update $\bar{\mathbf{w}}_n$ into $K \in \mathbb{Z}$ sub-vectors, i.e.,

$$\bar{\mathbf{w}}_n = [\bar{\mathbf{w}}_{n,1}, \bar{\mathbf{w}}_{n,2}, \dots, \bar{\mathbf{w}}_{n,K}]^T,$$

where each part $\bar{\mathbf{w}}_{n,k}, k \in [K]$ is a vector of size $\frac{L}{K}$ and $K \in [1 : \frac{(N-D+1)}{2} - A - T]$. If the value of K does not divide L , we can zero-pad the quantized local models.

3) **Broadcasting the Commitments:** Since all steps are done in the presence of adversarial behavior of Type-2, some initial hiding commitments are needed. This enables each user to commit to certain values without revealing any information about those values. Using the commitments, each user can verify that the messages being received are effectively the ones for which the commitment was created and ensure that all users follow the protocol honestly. Inspired by the commitment scheme in [45], we consider a scheme where each user n produces the commitments as follows

$$C_i^{(n)} = \begin{cases} \prod_{j=1}^{\frac{L}{K}} (g^{\beta^{j-1}})^{[\bar{\mathbf{w}}_{n,i}]_j}, & \text{if } i \in [K], \\ \prod_{j=1}^{\frac{L}{K}} (g^{\beta^{j-1}})^{[\mathbf{z}_{n,i-K}]_j}, & \text{if } i \in [K+1 : K+T], \\ \prod_{j=1}^{\frac{L}{K}} (g^{\beta^{j-1}})^{[\bar{\mathbf{z}}_{n,i-K-T}]_j}, & \text{if } i \in [K+T+1 : K+2T], \\ \prod_{j=1}^N (g^{\beta^{j-1}})^{r_{n,i-K-2T}^{(j)}}, & \text{if } i \in [K+2T+1 : 3K+4T-1] \setminus \{i \neq 2K+2T\}, \end{cases} \quad (4)$$

where $[\bar{\mathbf{w}}_{n,i}]_j$ is the j -th entry of $\bar{\mathbf{w}}_{n,i}$, and g is a generator of cyclic group \mathbb{G} with product operation, of prime order $p \geq 2^{2\kappa}$. In addition, $\beta \in \mathbb{F}_p$ is a secret parameter generated by a trusted authority. The trusted authority then generates public parameters $(g^{\beta^0}, g^{\beta^1}, \dots, g^{\beta^{\max(\frac{L}{K}, N)-1}})$. Recall that the size of each commitment is equal to a single group element. These commitments are binding, and computationally hiding under the assumption that the discrete logarithm problem is hard in \mathbb{G} (see proofs in Subsection V).

4) **Secret Sharing (First Round):** User n forms the following polynomial function.

$$\mathbf{F}_n(x) = \sum_{k=1}^K \bar{\mathbf{w}}_{n,k} x^{k-1} + \sum_{t=1}^T \mathbf{z}_{n,t} x^{K+t-1}, \quad (5)$$

where the coefficient of K first terms are the partitions of $\bar{\mathbf{w}}_{n,k}$, for $k \in [K]$. Each user n uses its polynomial function $\mathbf{F}_n(\cdot)$ to securely share its local update with other users. Let $\{\alpha_i \in \mathbb{F}_p : i \in [N]\}$ be a set of N distinct non-zero values in \mathbb{F}_p . This set is revealed to all users, such that each user n sends one valuation of its created polynomial function at $\alpha_{\tilde{n}}$ to user \tilde{n} , for $\tilde{n} \in [N]$. In particular, user n sends a vector $\mathbf{s}_{n,\tilde{n}} \triangleq \mathbf{F}_n(\alpha_{\tilde{n}})$ to user \tilde{n} , and each vector has a size of $\frac{L}{K}$. In this step, if a user m drops out and stays silent, $\mathbf{F}_m(\cdot)$ are just presumed to be \perp . Since the proposed secret sharing is based on the ramp sharing scheme, the local update is kept private against T colluding users by adding T independent random vectors in (5) to the message [14], [44].

5) **Verification (First Round):** Having received the shares $\mathbf{s}_{\tilde{n},n}$ from users $\tilde{n} \in [N]$, user n can verify them. Since each evaluation of polynomial function $\mathbf{F}_{\tilde{n}}(x)$ is a linear combination (coded version) of its coefficients, due to the linear homomorphism, the verification of $\mathbf{s}_{\tilde{n},n}$ can be done using the encoding of the commitments $C_i^{(\tilde{n})}$ as follows

$$\prod_{j=1}^{\frac{L}{K}} (g^{\beta^{j-1}})^{[\mathbf{s}_{\tilde{n},n}]_j} \stackrel{?}{=} \prod_{j=1}^{K+T} (C_j^{(\tilde{n})})^{\alpha_{\tilde{n}}^{j-1}}. \quad (6)$$

Using this verification, user n ensures that it receives a valid evaluation of polynomial $\mathbf{F}_{\tilde{n}}(\cdot)$ in (5).

6) **Secret Sharing (Second Round):** In this step, if $K \geq 2$, user n forms the polynomial function

$$\tilde{\mathbf{F}}_n(x) = \sum_{k=1}^K \bar{\mathbf{w}}_{n,k} x^{K-k} + \sum_{t=1}^T \tilde{\mathbf{z}}_{n,t} x^{K+t-1}, \quad (7)$$

and sends $\tilde{\mathbf{s}}_{n,\tilde{n}} \triangleq \tilde{\mathbf{F}}_n(\alpha_{\tilde{n}})$ to user \tilde{n} for $n, \tilde{n} \in [N]$. In addition, user n creates the scalar polynomial function

$$N_n^{(j)}(x) = \sum_{i=1, i \neq K}^{2(K+T)-1} r_{n,i}^{(j)} x^{i-1}, \text{ for } j \in [N] \setminus \{n\}, \quad (8)$$

and sends $N_{n,\tilde{n}}^{(j)} \triangleq N_n^{(j)}(\alpha_{\tilde{n}})$ to user \tilde{n} for $j \in [N] \setminus \{n\}$. The coefficient of x^{K-1} in polynomial $N_n^{(j)}(x)$ is equal to zero.

In this step, each user communicates with other users and sends a vector of size $\frac{L}{K}$ and $N - 1$ scalar values. It is worth noting that the structure of the created polynomial function $\tilde{\mathbf{F}}_n(x)$ in this step is different from the polynomial in Step 4. Note that for $K = 1$, in the second round of secret sharing, sending only the shares from the scalar polynomial function in (8) is sufficient.

7) **Verification (Second Round):** To ensure that user n receives a valid evaluation of polynomials $\tilde{\mathbf{F}}_{\tilde{n}}(x)$ and $N_{\tilde{n}}^{(j)}(x)$ from user \tilde{n} , it can check

$$\prod_{j=1}^{\frac{L}{K}} (g^{\beta^{j-1}})^{[\tilde{\mathbf{s}}_{n,\tilde{n}}]_j} \stackrel{?}{=} \prod_{j=1}^K (C_j^{(\tilde{n})})^{\alpha_n^{K-j}} \prod_{i=K+T+1}^{K+2T} (C_i^{(\tilde{n})})^{\alpha_n^{i-T-1}}, \quad (9)$$

$$\prod_{j=1}^N (g^{\beta^{j-1}})^{N_{\tilde{n},n}^{(j)}} \stackrel{?}{=} \prod_{\substack{i=K+2T+1, \\ i \neq 2K+2T}}^{3K+4T-1} (C_i^{(\tilde{n})})^{\alpha_n^{i-K-2T-1}},$$

using the available commitments. In this verification, user n can not only confirm that it receives a valid evaluation of polynomial $\tilde{\mathbf{F}}_{\tilde{n}}(x)$ from user \tilde{n} , but it can also ensure that user \tilde{n} correctly creates polynomial $\tilde{\mathbf{F}}_{\tilde{n}}(x)$ without any malicious behavior. This is because the initial commitments made in Step 3 are utilized for the verification.

8) **Computing Noisy Inner Products of Shares:** In this step, user n calculates the following inner product and sends the result to the server.

$$\bar{d}_{i,j}^{(n)} = \langle \mathbf{F}_i(\alpha_n) - \mathbf{F}_j(\alpha_n), \tilde{\mathbf{F}}_i(\alpha_n) - \tilde{\mathbf{F}}_j(\alpha_n) \rangle + N_{i,n}^{(j)} + N_{j,n}^{(i)}, \quad (10)$$

where $i, j \in [N]$ and $i < j$. Here, one can see that the inner product of the shares, $\bar{d}_{i,j}^{(n)}$, is an evaluation of a polynomial function

$$\bar{d}_{i,j}(x) \triangleq \sum_{\ell=0}^{2(K+T-1)} a_{\ell;i,j} x^{\ell} + N_i^{(j)}(x) + N_j^{(i)}(x), \quad (11)$$

at point α_n , where $a_{\ell;i,j}$ is the coefficient of x^{ℓ} . In this expansion, it can be shown that the coefficient of x^{K-1} is

$$a_{K-1,i,j} = \sum_{k=1}^K \|\bar{\mathbf{w}}_{i,k} - \bar{\mathbf{w}}_{j,k}\|^2.$$

For $K = 1$, user n computes $\bar{d}_{i,j}^{(n)} = \|\mathbf{F}_i(\alpha_n) - \mathbf{F}_j(\alpha_n)\|^2 + N_{i,n}^{(j)} + N_{j,n}^{(i)}$ and sends the result to the server.

9) **Distance Recovery at the Server:** Since $\bar{d}_{i,j}(x)$ is a polynomial function of degree $2(K+T-1)$, the server can use Reed-Solomon decoding [51] to recover all the coefficients of this polynomial using $2(K+T) - 1$ evaluations of $\bar{d}_{i,j}(x)$. Since in the setting, there are at most A adversarial users (Type-2), $\bar{d}_{i,j}(x)$ can only be correctly recovered if the server receives at least $2(K+T+A) - 1$ outcomes from the users. In addition, $\bar{d}_{i,j}^{(n)}$, for that are sent to the server from the non-adversarial users $n \in [N] \setminus \mathcal{D}$ are indeed equal to $\bar{d}_{i,j}(\alpha_n)$, for $i < j \in [N]$. Therefore, the server is able to recover $\bar{d}_{i,j}(x)$. The coefficient of x^{K-1} in $\bar{d}_{i,j}(x)$ is equal to $a_{K-1;i,j} = \|\bar{\mathbf{w}}_i - \bar{\mathbf{w}}_j\|^2$ that the server looks for in order to find the outliers. According to (10), the remaining recovered coefficients of $\bar{d}_{i,j}(x)$ are distorted by noise and do not disclose any additional details about the local updates apart from the targeted pairwise distance sought by the server.

Then the server converts the calculated distances from the finite field to the real domain as follows.

$$d_{i,j} = \frac{\Lambda^{-1}(a_{K-1;i,j})}{q^2}, \quad (12)$$

where $\Lambda^{-1} : \mathbb{F}_p \rightarrow \mathbb{R}$ is a demapping function which is applied element-wise and is defined as

$$\Lambda^{-1}(\bar{x}) = \begin{cases} \bar{x} & \text{if } 0 \leq \bar{x} < \frac{p-1}{2}, \\ \bar{x} - p & \text{if } \frac{p-1}{2} \leq \bar{x} < p. \end{cases} \quad (13)$$

Assuming a sufficiently large field size p , we can guarantee the accurate recovery of pair-wise distances.

10) **Outlier Detection at the Server:** In this step, the multi-Krum algorithm (see [18] and Subsection III-A), a distance-based outlier detection method (or closeness criterion), is em-

ployed to ensure that the local updates selected by the server are consistent with each other. This will help to eliminate any local updates that are significantly different from the others (adversarial behavior of Type-1) and may not be suitable for inclusion in the update. Upon completion of the algorithm, the server selects a group of m users, whose local updates are close to each other and broadcasts a list of the chosen users, denoted by $\mathcal{U}_s \subset [N]$. As stated in [18], the multi-Krum algorithm ensures the resiliency of the global model, as defined in Section III and the convergence of the sequence of gradients (See Subsection V).

11) **Aggregation of the Shares:** Each user calculates the aggregation of shares of users belonging to set \mathcal{U}_s , i.e., $\mathbf{s}_n = \sum_{\tilde{n} \in \mathcal{U}_s} \mathbf{F}_{\tilde{n}}(\alpha_n)$, and sends it to the server. In this step, each user sends a vector of size $\frac{L}{K}$ to the server.

12) **Recovering the Aggregation:** Let us define

$$\begin{aligned} \mathbf{F}(x) &\triangleq \sum_{n \in \mathcal{U}_s} \mathbf{F}_n(x) & (14) \\ &= \sum_{k=1}^K x^{k-1} \sum_{n \in \mathcal{U}_s} \bar{\mathbf{w}}_{n,k} + \sum_{t=1}^T x^{K+t-1} \sum_{n \in \mathcal{U}_s} \mathbf{z}_{n,t}, \end{aligned}$$

as a polynomial of degree $K + T - 1$. Let $\mathcal{X}_S \triangleq \{\mathbf{s}_n\}_{n \in S}$ denote the set of messages received by the server, where $S = [N] \setminus \mathcal{D}$. We note that \mathbf{s}_n , for non-Byzantine users, is the evaluation of $\mathbf{F}(x)$ at α_n . Thus, if the server receives a minimum of $K + T + 2A$ results from the users, it can accurately recover the polynomial function $\mathbf{F}(x)$ using Reed-Solomon decoding. In this decoding process, the first K terms correspond to the partitions of $\bar{\mathbf{w}} = \sum_{n \in \mathcal{U}_s} \bar{\mathbf{w}}_n$. To ensure successful aggregation, the field size p must be sufficiently large to avoid encountering boundary issues during the process.

After recovering the aggregate of the users' local updates, the server updates the global model for the next iteration using the following procedure.

$$\mathbf{w}_g^{(t+1)} = \mathbf{w}_g^{(t)} - \frac{\delta_t}{q|\mathcal{U}_s|} \Lambda^{-1}(\bar{\mathbf{w}}), \quad (15)$$

where demapping function $\Lambda^{-1}(\cdot)$ is defined in (13), and δ_t is the learning rate at round t .

Algorithm I summarizes ByzSecAgg based on the aforementioned steps.

Remark 1: The proposed scheme uses some methods to mitigate adversarial behaviors of Type-1 and Type-2. To protect against adversarial behavior of Type-1, where attempts are made to manipulate the global model at the server by altering their local updates, ByzSecAgg employs a distance-based outlier detection mechanism (multi-Krum algorithm). This mechanism ensures the robustness of the global model against such adversarial modifications. Additionally, to protect the privacy of local updates in the outlier detection mechanism, ByzSecAgg proposes a privacy-preserving distance computation method, inspired by ramp secret sharing and coded computing. For protection against adversarial behavior of Type-2, first, a vector commitment scheme is utilized in ByzSecAgg to ensure that users follow the protocol correctly when creating secret shares of their local updates in two

Algorithm 1 The proposed Byzantine-Resistant Secure Aggregation Scheme: ByzSecAgg

```

1: for each iteration  $t = 0, \dots, T$  do
2:   for each user  $n \in [N]$  in parallel do
3:     Get the global model  $\mathbf{w}_g^{(t)}$  from the server
4:     Compute the local update  $\mathbf{w}_n$  based on the local dataset
5:     Create the quantized update  $\bar{\mathbf{w}}_n$  using (2)
6:     Partition the local update into  $K$  parts
7:     Generate commitments  $C_i^{(n)}$ ,  $i \in [3K + 4T - 1]$  using (4) and broadcast them
8:     Compute the first-round secret share  $\mathbf{s}_{n,\tilde{n}}$  using (5) and send to user  $\tilde{n} \in [N]$ 
9:     Verify the received shares  $\{\mathbf{s}_{\tilde{n},n}\}_{\tilde{n} \in [N]}$  using (6)
10:    Compute the second-round secret shares  $\tilde{\mathbf{s}}_{n,\tilde{n}}$  and noise shares  $\{N_{n,\tilde{n}}^{(j)}\}_{j \in [N]}$  using (7) and (8) respectively and send to user  $\tilde{n} \in [N]$ 
11:    Verify the received shares  $\tilde{\mathbf{s}}_{\tilde{n},n}$  and  $\{N_{\tilde{n},n}^{(j)}\}_{j \in [N]}$  by testing (9)
12:    Compute values  $\bar{d}_{i,j}^{(n)}$ , for  $i, j \in [N]$ , using (10) and send to the server
13:  end for
14:  Server recovers pairwise distances  $\|\bar{\mathbf{w}}_i - \bar{\mathbf{w}}_j\|^2$ , for  $i < j \in [N]$  after receiving the results using Reed-Solomon decoding
15:  Server converts the calculated distances from the finite field to the real domain and recovers  $\{d_{i,j}\}_{i < j \in [N]}$  using (12).
16:  Server selects set  $\mathcal{U}_s$  of users by applying multi-Krum algorithm on  $\{d_{i,j}\}_{i < j \in [N]}$ 
17:  Server broadcast set  $\mathcal{U}_s$  to all users
18:  for user  $n = 1, \dots, N$  do
19:    Calculate the aggregation of the first-round shares of users belonging to set  $\mathcal{U}_s$  and send the result to the server
20:  end for
21:  Server recovers  $\bar{\mathbf{w}} = \sum_{n \in \mathcal{U}_s} \bar{\mathbf{w}}_n$  after receiving sufficient results from users and employing Reed-Solomon decoding
22:  Server updates the global model using (15)
23: end for

```

different rounds. In addition, the ideas from error-correcting codes are used to ensure the correctness of the calculations of users.

Remark 2: To eliminate outliers while minimizing communication requirements, ByzSecAgg uses two different rounds of secret sharing to securely calculate the pairwise distance between the local updates. It is worth noting that the second round of secret sharing is designed to only allow the server to calculate the pairwise distance between the local updates, rather than the pairwise distance between the *partitions* of local updates.

V. ANALYSIS OF THE PROPOSED SCHEME

A. Main results

In this subsection, we present the main results, achieved by *ByzSecAgg* described in Section IV

Theorem 1. *We assume that (i) the cost function \mathfrak{L} is three times differentiable with continuous derivatives, and is non-negative, i.e., $\mathfrak{L}(x) \geq 0$; (ii) the learning rates satisfy $\sum_t \delta_t = \infty$ and $\sum_t \delta_t^2 < \infty$; (iii) the second, the third, and the fourth moments of the quantized gradient estimator satisfy $\mathbb{E}_{Q, \zeta} \|Q_q(\mathbf{G}(\mathbf{w}_g, \zeta))\|^r \leq A_r + B_r \|\mathbf{w}_g\|^r$ for some constant A_r and B_r , where $r \in 2, 3, 4$; (iv) there exist a constant $0 \leq \gamma < \pi/2$ such that for all $\mathbf{w}_g \in \mathbb{R}^L$, $\eta(N, A) \sqrt{L\sigma^2(\mathbf{w}_g) + \frac{L}{4q^2}} \leq \|\nabla \mathfrak{L}(\mathbf{w}_g)\| \sin \gamma$; (v) the gradient of the cost function \mathfrak{L} satisfies that for $\|\mathbf{w}_g\|^2 \geq R$, there exists constants $\varepsilon > 0$ and $0 \leq \theta < \pi/2 - \gamma$ such that*

$$\begin{aligned} \|\nabla \mathfrak{L}(\mathbf{w}_g)\| &\geq \varepsilon > 0, \\ \frac{\mathbf{w}_g^T \mathfrak{L}(\mathbf{w}_g)}{\|\mathbf{w}_g\| \|\nabla \mathfrak{L}(\mathbf{w}_g)\|} &\geq \cos \theta. \end{aligned}$$

Then, *ByzSecAgg* guarantees,

- Robustness against D dropouts, and A Byzantine adversarial users (Type-1 and Type-2) such that the trained model is (γ, A) -Byzantine resilient,
- The sequence of the gradients $\nabla \mathfrak{L}(\mathbf{w}_g^{(t)})$ converges almost surely to zero,
- Any group of up to T colluding users cannot extract any information about the local models of other honest users. The server cannot gain any information about local updates of the honest users, beyond the aggregation of them and pair-wise distances.

In addition, the communication loads in *ByzSecAgg* are as follows

$$\begin{aligned} R_{\text{server}} &= \left(1 + \frac{2A + T}{K}\right)L + (T + A + K - \frac{1}{2})N(N - 1), \\ R_{\text{user}} &\leq \min\left(\frac{2N}{K}, N\right)L + \frac{3N(N - 1)}{2}, \end{aligned}$$

symbols from \mathbb{F}_p , for some $K \in \mathbb{N}$ in $[1 : \frac{N-D+1}{2} - A - T]$ $N \geq 2A + D + \max(2K + 2T - 1, m + 3)$ and $m < N - 2A - D - 2$. Furthermore, the size of the commitment for each user in the achievable scheme remains constant, equal to $(3T + 3K - 2) + \mathbb{I}(K > 1)(T)$, regardless of the size of the individual local updates L . Additionally, $\mathbb{I}(\cdot)$ is an indicator function.

Remark 3: As explained in Section IV, K and m are designed parameters. K is the number of partitions of the local updates. By changing K , one can change R_{server} and R_{user} . One option is to choose K such that the aggregation of the per-user communication load and the server communication load is minimized. On the other hand, m is the number of users whose local updates are selected by the server for aggregation, as discussed in Subsection III-A and Section IV

Proof. In the following, we prove that *ByzSecAgg* satisfies the following conditions:

1) Correctness:

- **Correctness of Commitments:** The function $h(x) \triangleq g^x$, in (4), creates a bijection mapping between the finite field \mathbb{F}_p and cyclic group \mathbb{G} , where g is a generator of \mathbb{G} . This function also has the linear homomorphic property which states that $\forall a_1, \dots, a_n \in \mathbb{F}_p$ and $\forall x_1, \dots, x_n \in \mathbb{F}_p$ we have

$$h\left(\sum_{i=1}^n a_i x_i\right) = \prod_{i=1}^n h(x_i)^{a_i}.$$

If the user is honest, it will share the evaluations of the same polynomial $\mathbf{F}_n(x)$ ($\tilde{\mathbf{F}}_n(x)$ and $N_n^{(j)}(x)$) in the first (second) round of secret sharing that it had initially committed to in Step 3. Due to the linearly homomorphic property of the commitments in (4), the verification step in (6) (in (9)) passes for the shares of the honest user.

- **Correctness of the Final Result:** The received vectors by the server at the end of Step 12 are different evaluations of polynomial $\mathbf{F}(x)$ of degree $K + T - 1$ defined in (14). The correctness condition of the final result of *ByzSecAgg* is satisfied as the server can correctly recover $\sum_{n \in \mathcal{U}_s} \bar{\mathbf{w}}_n$ by utilizing the Reed-Solomon decoding after receiving $K + T + 2A$ outcomes from the users with at most A adversarial outcomes.

2) Robustness against Byzantine Adversaries:

- **Commitment Binding:** The Discrete Logarithm (DL) Assumption [52] states that given a prime p , a generator g of \mathbb{G} , and an element $a \in \mathbb{F}_p$, no adversary with a probabilistic polynomial-time algorithm can compute a given g and g^a , i.e., $\Pr[\mathcal{A}(g, g^a) = a] = \epsilon(\kappa)$ (negligible) for any such adversarial attack algorithm \mathcal{A} .

In addition, consider the polynomial function $q_{\mathbf{w}_{n,i}}(x)$ defined as

$$q_{\mathbf{w}_{n,i}}(x) \triangleq [\bar{\mathbf{w}}_{n,i}]_1 + x[\bar{\mathbf{w}}_{n,i}]_2 + \dots + x^{\frac{L}{K}-1}[\bar{\mathbf{w}}_{n,i}]_{\frac{L}{K}},$$

where $[\bar{\mathbf{w}}_{n,i}]_j$ is the j -th entry of the i -th partition of the quantized local update $\bar{\mathbf{w}}_n$. It can be verified that commitments $C_i^{(n)}$ in (4) for $i \in [K]$ are equal to $g^{q_{\mathbf{w}_{n,i}}(\beta)}$. By contradiction, assume that there exists an adversary \mathcal{A} that breaks the binding property of commitments. This means that the adversary creates two vectors $\bar{\mathbf{w}}_{n,i}$ and $\bar{\nu}_{n,i}$, where $\bar{\mathbf{w}}_{n,i} \neq \bar{\nu}_{n,i}$ but $C_i^{(n)} \triangleq g^{q_{\mathbf{w}_{n,i}}(\beta)} = g^{q_{\nu_{n,i}}(\beta)}$.

In this case, it can be shown that the adversary can break the DL assumption, which means it is able to recover secret β by having g^β . Define the polynomial function $\tilde{q}(x)$ as $q_{\mathbf{w}_{n,i}}(x) - q_{\nu_{n,i}}(x)$. The corresponding commitment is equal to $\tilde{C}_i^{(n)} \triangleq g^{\tilde{q}(\beta)} = g^{q_{\mathbf{w}_{n,i}}(\beta)} / g^{q_{\nu_{n,i}}(\beta)} = 1$ because the commitment scheme is homomorphic. Therefore, $\tilde{q}(\beta) = 0$, which means that β is a root of the polynomial $\tilde{q}(x)$. The adversary can easily solve the instance of the discrete logarithm problem and find the secret parameter β by factoring $\tilde{q}(x)$ [53]. This implies that breaking the binding property would enable solving a problem that is assumed to be computationally difficult, reinforcing the notion that the commitment scheme must possess the binding property.

For the evaluation binding, we can present a similar argument. Let's assume there exists an adversarial algorithm \mathcal{A} that breaks the evaluation binding property of commitment C and computes two different evaluations for $\mathbf{F}_n(x)$ in

⑤ at point α that satisfy Verify in ⑥. We can construct an algorithm \mathcal{B} that uses \mathcal{A} to break the DL assumption. \mathcal{B} presents a DL instance $(g, g^\beta, \dots, g^{\beta^{\frac{L}{K}}})$. Algorithm \mathcal{A} outputs commitments C and two distinct values for share, i.e., $\mathbf{F}_n(\alpha)$ and $\mathbf{F}'_n(\alpha)$, that satisfy ⑥. Specifically, we have

$$\prod_{j=1}^{\frac{L}{K}} (g^{\beta^{j-1}})^{[\mathbf{F}_n(\alpha)]_j} = \prod_{j=1}^{\frac{L}{K}} (g^{\beta^{j-1}})^{[\mathbf{F}'_n(\alpha)]_j} = \prod_{j=1}^{K+T} (C_j^{(n)})^{\alpha^{j-1}}.$$

Consequently, we observe that

$$\prod_{j=1}^{\frac{L}{K}} (g^{\beta^{j-1}})^{[\mathbf{F}_n(\alpha)]_j - [\mathbf{F}'_n(\alpha)]_j} = 1.$$

This implies that β is a root of polynomial $F''(x) = \sum_{j=1}^{\frac{L}{K}} x^{j-1} ([\mathbf{F}_n(\alpha)]_j - [\mathbf{F}'_n(\alpha)]_j)$. Therefore, algorithm \mathcal{B} can compute β by factoring $F''(x)$ once it receives the results from \mathcal{A} . Thus, the success probability of solving the DL instance is the same as the success probability of \mathcal{A} .

- Malicious Computation Result (Type-2 Adversarial Attack): Here we discuss the adversarial attack of Type-2 and how ByzSecAgg resolves it:

- The transmission of invalid secret shares in Step 4 and Step 6: this kind of attack is thwarted through the utilization of the commitments, in which the validity of secret shares at each round can be confirmed by verifying ⑥ and ⑨, provided that $N - D \geq 2A + 1$.
- The transmission of incorrect pairwise distance of shares to the server in Step 8 or incorrect aggregation of the shares of selected users in Step 11: Since the values sent at each of these two steps must represent different evaluation points of a certain polynomial, this kind of attack will be detected and corrected based on Reed-Solomon decoding algorithm with at most D erasures and at most A errors, provided that $N - D \geq 2(K + T + A) - 1$ in Step 8 and $N - D \geq K + T + 2A$ in Step 11.

- Global Model Resiliency (Type-1 Adversarial Attack): Byzantine users have the ability to modify their local updates to manipulate the global model and can send arbitrary vectors as their local updates to the server. Additionally, in line with classical assumptions in machine learning literature, each data sample used for computing the local update is uniformly and independently drawn by honest users. To detect outliers, ByzSecAgg uses the multi-Krum algorithm introduced in Subsection III-A. Instead of the estimator $\mathbf{G}(\mathbf{w}_g^{(t)}, \zeta_n^{(t)})$, in the proposed scheme, the local update of the honest user is created using the quantized estimator $Q_q(\mathbf{G}(\mathbf{w}_g^{(t)}, \zeta_n^{(t)}))$. In ⑨, it is demonstrated that even when the multi-krum algorithm is applied to the quantized vector $Q_q(\mathbf{w}_n)$ (as used in Step 1 of the proposed scheme), it remains (α, A) -Byzantine resilient and the sequence of the gradients converges almost surely to zero. This is due to the fact that the quantization is unbiased and has a bounded variance. According to Subsection III-A if $m < N - 2A - D - 2$, the multi-Krum algorithm can prevent this type of attack and ensure that the scheme is (γ, A) -Byzantine resilient and guarantees the convergence.

Note that the first five conditions in Theorem ①, i.e., (i)-(v), are essential in the convergence proof of multi-Krum algorithm (see ⑩, ⑪).

3) Privacy Constraint:

- Hiding Property: For the sake of contradiction, suppose there exists an adversarial attack algorithm \mathcal{A} that can break the hiding property of commitment C and correctly compute vector $\nu \in \mathbb{F}_p$. We will demonstrate how to utilize \mathcal{A} to construct another algorithm \mathcal{B} that can break the Discrete Logarithm (DL) assumption.

Let (g, g^a) be a DL instance that \mathcal{B} aims to solve. \mathcal{B} selects a random $\tilde{\beta} \in \mathbb{F}_p^L$ and computes $(g, g^{\tilde{\beta}}, \dots, g^{\tilde{\beta}^{L-1}})$. It then considers a vector $\nu = [a, \nu_1, \nu_2, \dots, \nu_L]$, where $\nu_1, \nu_2, \dots, \nu_{L-1} \in \mathbb{F}_p$ are chosen arbitrarily, and the first entry of the vector a is the answer for the DL instance. Since \mathcal{B} knows g^a , it can compute the vector commitment as $C = g^a \cdot g^{\sum_{i=1}^{L-1} \nu_i \tilde{\beta}^i}$ and sends it to \mathcal{A} . Upon receiving the commitment, \mathcal{A} computes and returns vector ν . At this point, \mathcal{B} can extract the solution a , as it corresponds to the DL instance. Thus, the success probability of solving the DL instance using \mathcal{B} is equivalent to the success probability of \mathcal{A} . The aforementioned construction illustrates that if an adversary \mathcal{A} can break the hiding property of commitment C , then we can utilize it to construct another algorithm \mathcal{B} capable of breaking the DL assumption.

- Privacy of Individual Local Updates: The hiding property of the commitments ensures that neither the server nor any user can compute the local model $\bar{\mathbf{w}}_n$ (as well as noise vectors \mathbf{z}_n and $\tilde{\mathbf{z}}_n$) from the commitments $C_i^{(n)}$ in ④. Thus, by considering the messages exchanged during the scheme, it is sufficient to establish the privacy of each local update against a group of up to T colluding users, denoted by set \mathcal{T} . Denote the colluding users by $\tilde{U}_1, \tilde{U}_2, \dots, \tilde{U}_T$. Let us define the set of messages received by \tilde{U}_i as $\mathcal{M}_{\tilde{U}_i}$, which includes two types of messages: $\mathbf{s}_{n,i}$, $\tilde{\mathbf{s}}_{n,i}$ and $N_{n,i}^{(j)}$ for $n \in [N] \setminus \{\mathcal{T} \cup \mathcal{D}\}$, $j \in [N] \setminus \{n\}$. These messages correspond to the shares from the first and second rounds of secret sharing in Step 4 and Step 6, respectively. According to ⑤ and ⑦, the fact that the random vectors are chosen uniformly and independently at random from $\mathbb{F}_p^{\frac{L}{K}}$, and directly from the privacy guarantee in ramp secret sharing ④, for $n \in [N] \setminus \mathcal{T}$, we have

$$\begin{aligned} & \Pr(\bar{\mathbf{w}}_n = \nu | \mathcal{M}_{\tilde{U}_i}, i \in [T]) \\ &= \Pr(\bar{\mathbf{w}}_n = \nu | \mathbf{s}_{n,i}, \tilde{\mathbf{s}}_{n,i}, N_{n,i}^{(j)}, j \in [N] \setminus n, i \in [T]) \\ &= \Pr(\bar{\mathbf{w}}_n = \nu), \end{aligned}$$

which means that the colluding users cannot gain any further information about the local updates of the other users.

Furthermore, the server is also provided with $\tilde{d}_{i,j}^{(n)}$ for $i < j \in [N]$ and $\mathbf{s}_n = \sum_{\tilde{n} \in \mathcal{U}_s} \mathbf{s}_{\tilde{n},n}$ from users $n \in [N] \setminus \mathcal{D}$, from which it can reconstruct $\tilde{d}_{i,j}^{(n)}(x)$ as defined in ⑪, as well as the polynomial function $\mathbf{F}(x)$ as defined in ⑭. Therefore, from $\mathbf{F}(x)$ the server can

recover the aggregation $\bar{\mathbf{w}} = \sum_{n \in \mathcal{U}_s} \bar{\mathbf{w}}_n$ and from the coefficient of x^{K-1} in $\bar{d}_{i,j}^{(n)}(x)$ it can recover the mutual distance $\|\bar{\mathbf{w}}_i - \bar{\mathbf{w}}_j\|^2$. It is important to note that due to the inclusion of random noises during the computation in

Step 8 by each user, the server cannot retrieve any other details about the local models. The only information that can be obtained is the mutual distances between the local updates, which play a vital role in outlier detection.

B. Communication Loads in ByzSecAgg

According to Step 12, the total number of vectors that need to be received by the server to recover the final aggregation is $T + K + 2A$, each of size $\frac{L}{K}$ symbols. Moreover, in order to recover the pairwise distance $d_{i,j}$ between $\bar{\mathbf{w}}_i$ and $\bar{\mathbf{w}}_j$, the server needs $2(K + T + A) - 1$ symbols of size a single field element from the users, for $i, j \in [N], i \neq j$. Thus, the server communication load in ByzSecAgg is $R_{\text{server}} \leq (1 + \frac{2A+T}{K})L + (T+A+K-\frac{1}{2})N(N-1)$, where the inequality is a result of the presence of dropped-out users in the setting.

In ByzSecAgg, each user participates in two rounds of secret sharing. In the first round, they send one vector of size $\frac{L}{K}$ symbols and in the second round, they send one vector of size $\frac{L}{K} + (N-1)$ symbols to each of the other users. Additionally, each user sends one vector of size $\frac{L}{K}$ symbols to the server. However, if $K = 1$, in second round of secret sharing, the shares from the scalar polynomial function in (8) are only sent. In addition, each user sends at most $\frac{N(N-1)}{2}$ pair-wise distances of size a single field element to the server. Thus, the per-user communication load in ByzSecAgg is upper-bounded as $R_{\text{user}} \leq \min(\frac{2N}{K}, N)L + \frac{3N(N-1)}{2}$. In addition, the commitment size for each user is only $3T + 1$ symbols when $K = 1$, and $3K + 4T - 2$ symbols when $K > 1$, where each symbol has the size of a single group element.

Therefore, the communication loads R_{server} and R_{user} in Theorem 1 are achieved. Additionally, according to discussions in Subsection V to meet all requirements, the inequality

$$N \geq 2A + D + \max(2K + 2T - 1, m + 3), \quad (16)$$

must be fulfilled. In addition, the following condition

$$1 \leq K \leq \frac{N - D + 1}{2} - A - T, \quad (17)$$

must be satisfied for the number of partitions of the local updates, where $K \in \mathbb{Z}$. \square

C. Computational Complexity of ByzSecAgg

Computational Complexity at the User: In ByzSecAgg, each user n performs the following three operations:

1) Secret Sharing: The user n generates secret shares $s_{n,\tilde{n}}$ and $\tilde{s}_{n,\tilde{n}}$ for $\tilde{n} \in [N]$. Generating secret shares is equivalent to the evaluation of the polynomials in (5) and (7) in N distinct values. If the field supports FFT, evaluation of a polynomial function of degree m at m points has a computational complexity of $\mathcal{O}(m \log^2 m)$ [54]. In ByzSecAgg, we need to compute polynomials of degree $K + T - 1$ at N points, where the coefficients are vectors of size $\frac{L}{K}$ and $K + T - 1 < N$. Therefore, generating the shares has a

computational cost of $\mathcal{O}(\frac{2NL}{K} \log^2 N)$. In addition, user n generates shares $N_{n,\tilde{n}}^{(j)}$ from scalar polynomial function in (8) for $\tilde{n} \in [N]$ and $j \in [N] \setminus \{n\}$ which has a computation cost of $\mathcal{O}(N^2 \log^2 N)$.

2) Inner Product Computation: The user computes the inner product of the shares, $\bar{d}_{i,j}^{(n)}$, as defined in (10) for $i, j \in [N], i < j$. This step has a computational complexity of $\mathcal{O}(N^2 \frac{L}{K})$.

3) Aggregation: The user aggregates the shares of other users belonging to the set \mathcal{U}_s . Assuming the cardinality of \mathcal{U}_s is $\mathcal{O}(N)$, the aggregation of secret shares for the selected users has a computational complexity of $\mathcal{O}(N \frac{L}{K})$. Therefore, the overall computational cost for each user is $\mathcal{O}(\frac{2NL}{K} \log^2 N + N^2 \frac{L}{K} + N^2 \log^2 N)$.

Computational Complexity at the Server: In ByzSecAgg, the server performs the following three operations:

1) Distance Recovery: The server should recover the pairwise distances between local updates. The computation performed by the server includes interpolation of a polynomial $\bar{d}_{i,j}(x), i, j \in [N]$ in (11) of degree $2(K + T - 1) < N$ using Reed-Solomon decoding. The complexity of interpolation of a polynomial of degree m is $\mathcal{O}(m \log^2 m)$, when the field supports FFT [54]. Thus, recovering $\mathcal{O}(N^2)$ pairwise distances has the computation cost of $\mathcal{O}(N^3 \log^2 N)$.

2) Outlier Detection: The server utilizes the Multi-Krum algorithm to eliminate outliers and select a set of users. Based on the recovered pairwise distances, the server calculates the score of each local update and selects m users with the lowest scores. This involves computing the sum of distances of each local update to its closest $N - A - 2$ neighbors. Selecting the smallest $N - A - 2$ distances out of $N - 1$ can be done using a sorting algorithm which has a computation cost of $\mathcal{O}(N \log N)$ per vector. For N vectors, the total complexity is $\mathcal{O}(N^2 \log N)$. Similarly, selecting $m < N - 2A - 2$ users with the smallest sum of distances has a cost of $\mathcal{O}(N \log N)$. In total, the complexity of this step is $\mathcal{O}(N^2 \log N)$.

3) Recovering the Aggregation: The server interpolates polynomial function $\mathbf{F}(x)$ in (14) to recover the final aggregation result of the selected users. The computational complexity of this step is $\mathcal{O}(\frac{NL}{K} \log^2 N)$.

Therefore, the computational cost of the server in ByzSecAgg is $\mathcal{O}((N^3 + \frac{NL}{K}) \log^2 N)$.

D. The Size of Finite Field in ByzSecAgg

In this subsection, we analyze the minimum field size required to achieve a certain accuracy for ByzSecAgg to work properly. Let us assume that each user n has local update \mathbf{w}_n , which is a vector of length L . We use the quantization function in (1) with a quantization level $q \geq 1$ and quantization error in $[0, \frac{1}{q}]$.

Assume that the value of the entries of the local models is bounded, i.e., $\forall n \in [N], i \in [L]$, we have $-\tau < [\mathbf{w}_n]_i < \tau$, where τ is a positive integer. Here, $[\mathbf{w}_n]_i$ represents the i -th entry of local model \mathbf{w}_n . Consequently, $\forall n \in [N], i \in [L], -\tau q \leq \lfloor q[\mathbf{w}_n]_i \rfloor \leq \tau q - 1$. In ByzSecAgg, there are two main computations: the pairwise distances and the aggregation

of the local updates. These computations are performed in a finite field, which must be large enough to avoid boundary issues during the process. We choose a finite field $GF(p)$, denoted by \mathbb{F}_p , for some prime number p . The choice of p must satisfy the following conditions:

Aggregation Recovery Condition: To ensure the correct recovery of the aggregation, we have

$$\begin{aligned} \sum_{n \in \mathcal{U}_s} \bar{\mathbf{w}}_n &= \frac{1}{q} \Lambda^{-1} \left(\sum_{n \in \mathcal{U}_s} \Lambda(qQ(\mathbf{w}_n)) \right) \\ &\stackrel{(eq\ 1)}{=} \frac{1}{q} \Lambda^{-1} \left(q \Lambda \left(\sum_{n \in \mathcal{U}_s} Q(\mathbf{w}_n) \right) \right) \\ &\stackrel{(eq\ 2)}{=} \sum_{n \in \mathcal{U}_s} Q(\mathbf{w}_n) = \sum_{n \in \mathcal{U}_s} \frac{\lfloor q\mathbf{w}_n \rfloor}{q}, \end{aligned}$$

where (eq 1) holds due to the linearity of the function $\Lambda(\cdot)$ and (eq 2) holds if the following condition is met $q \left| \sum_{n \in \mathcal{U}_s} Q(\mathbf{w}_n) \right| < \frac{p-1}{2}$. Thus, the first condition for p is $p > 2N\tau q + 1$.

Pairwise Distance Recovery Condition: To ensure the correct recovery of the pairwise distances, we have

$$\begin{aligned} \|\bar{\mathbf{w}}_i - \bar{\mathbf{w}}_j\| &= \frac{1}{q^2} \Lambda^{-1} \left(\|\Lambda(qQ(\mathbf{w}_i)) - \Lambda(qQ(\mathbf{w}_j))\|^2 \right) \\ &= \frac{1}{q^2} \Lambda^{-1} \left(q^2 \|Q(\mathbf{w}_i) - Q(\mathbf{w}_j)\|^2 \right) \\ &= \|Q(\mathbf{w}_i) - Q(\mathbf{w}_j)\|^2, \end{aligned}$$

which holds if $q^2 \|Q(\mathbf{w}_i) - Q(\mathbf{w}_j)\|^2 < \frac{p-1}{2}$. Thus, the second condition for p is $p > 2L(2\tau q - 1)^2 + 1$.

Combining these conditions, we select p such that

$$p > 2 \max\{L(2\tau q - 1)^2, N\tau q\} + 1. \quad (18)$$

With a larger quantization level q , the scheme achieves better accuracy because the variance introduced by the quantization function decreases as q increases. However, for a higher quantization level, a larger field size is required.

Remark 4: In *ByzSecAgg*, choosing the finite field \mathbb{F}_p to satisfy (18) is not a consequence of using the ramp secret sharing scheme. Since in the process of partitioning the local updates into K parts, we partition a vector, not a symbol. If $K \nmid L$, we can zero-pad the local model vector. Shamir's secret sharing would require the same field size. Therefore, *BREA* would also use the same field size for a certain level of precision in the quantization step.

E. Comparison with BREA [24]

The comparison results between *ByzSecAgg* and *BREA* [24] are summarized in Table I, which highlights the differences between the two schemes based on the defined parameters. In this subsection, to demonstrate the communication load reduction performance of *ByzSecAgg* compared to *BREA*, the theoretical results presented in Table I are evaluated for different parameter values. In addition, to have better intuition, the parameter values are selected based on real-world and well-known architectures.

Figure 1 compares the communication loads, measured in symbols, between *ByzSecAgg* and *BREA*. The parameter values selected for the evaluation involves $N = 1000$ users, with $T = 0.1N$ colluding users, $A = 0.1N$ Byzantine adversaries, and $D = 0.2N$ potential dropouts. The comparison focuses on three metrics: (a) per-user communication load, (b) server communication load, and (c) commitment size of each user. The number of parameters of commonly used neural networks, including GoogleNet [55], ResNet [56], and AlexNet [57], are considered for parameter value L . The results demonstrate that *ByzSecAgg* achieves an order-wise reduction in communication loads and commitment sizes compared to *BREA*. In this comparison, for *ByzSecAgg*, the number of partitions (K) is selected to minimize the aggregation of per-user communication load and server communication load. As is shown, the commitment size of the proposed scheme remains constant and is not influenced by the length of local updates. The slight variations observed in the graph are due to changes in the selected value for K .

Figure 2 illustrates the communication loads of the two schemes as the number of users (N) varies. For the evaluation, the values $L = 21.8M$, $T = 0.1N$ colluding users, $A = 0.1N$ Byzantine adversaries, and $D = 0.2N$ dropouts are considered. Similarly, in this comparison, for *ByzSecAgg* the number of partitions (K) is selected to minimize the aggregation of per-user communication load and server communication load. Note that compared with *BREA*, the proposed scheme achieves the same performance in terms of convergence and resilience properties.

Remark 5: Note that, in terms of privacy, *BREA* in [24] has more information leakage compared to our proposed scheme. The reason is that, in *BREA*, each user employs Shamir's secret sharing to generate shares, computes the pairwise distances between the received shares, and sends the results back to the server. It can be shown that multiplying the shares from two Shamir's secret sharing polynomials does not preserve the privacy of the product of the secrets. In *ByzSecAgg*, we address this issue by using $N_n^{(j)}(x)$ and their shares, which are employed to compute noisy inner products of shares in Step 8. Therefore, *ByzSecAgg* has a higher level of privacy.

According to Subsection V-C and [24], Table II provides a comparison of the computational complexity for the server and per-user operations in *ByzSecAgg* and *BREA*. The partitioning step proposed in *ByzSecAgg* reduces the computational complexity of the server and per user for large K . However, in the per-user complexity, there is an additional term $N^2 \log^2 N$, as *ByzSecAgg* aims to achieve a higher level of privacy. *ByzSecAgg* appears more scalable due to its reduced dependency on L in both server and per-user computations when partitioning is applied $K > 1$.

VI. CONCLUSION

In this paper, we propose *ByzSecAgg*, an efficient secure aggregation scheme for federated learning that mitigates Byzantine adversarial attacks while protecting the privacy of individual local updates. *ByzSecAgg* employs techniques

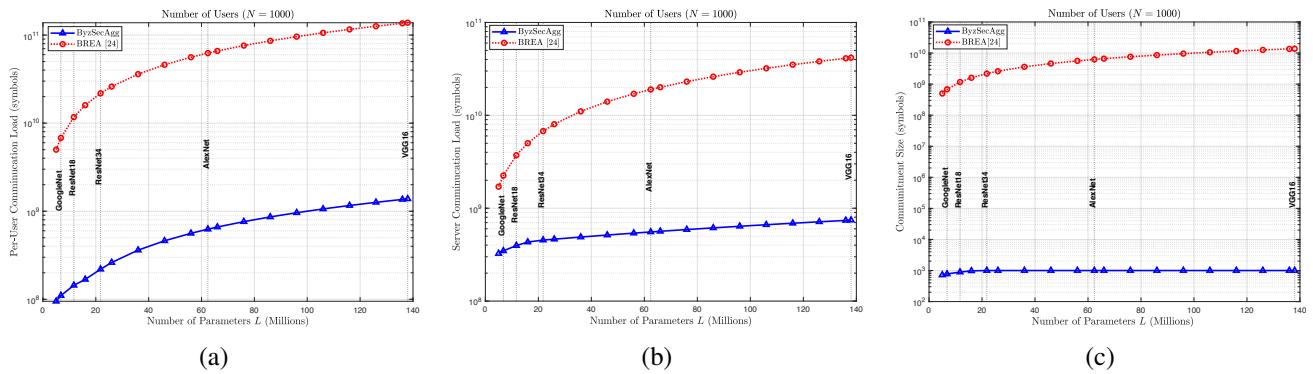


Fig. 1. The comparison of communication loads, measured in symbols, between ByzSecAgg and BREA [24] with a varying number of local update parameters (L). The figure illustrates the comparison using $N = 1000$ users, including $T = 0.1N$ colluding users, $A = 0.1N$ Byzantine adversaries, and $D = 0.2N$ potential dropouts. Three metrics are considered: (a) per-user communication load, (b) server communication load, and (c) commitment size of each user.

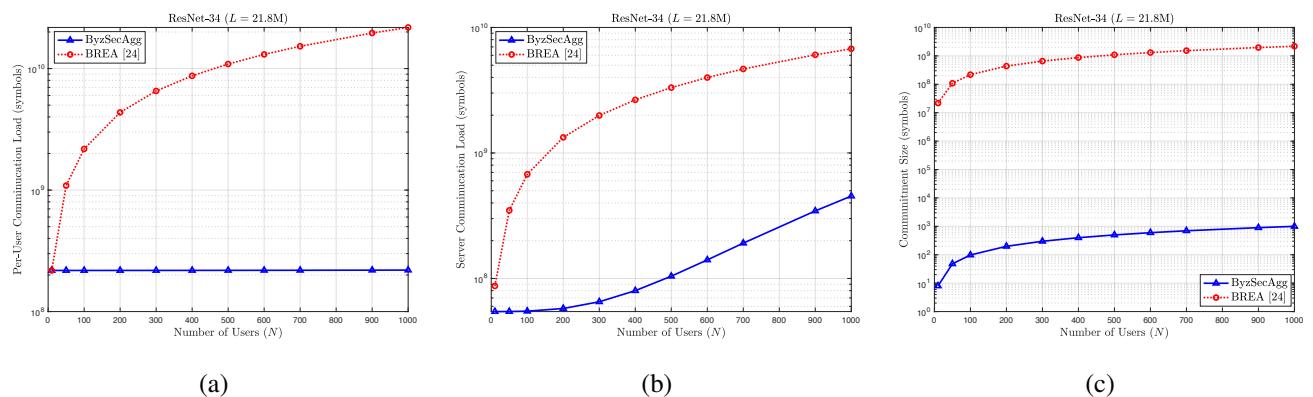


Fig. 2. The comparison of ByzSecAgg and BREA [24] in terms of communication loads while varying the number of users. The experiment considers ResNet34 with $L = 21.8M$ parameters. Out of the total users (N), $T = 0.1N$ are colluding users, $A = 0.1N$ are Byzantine adversaries, and $D = 0.2N$ may drop out. The comparison evaluates three metrics: (a) per-user communication load, (b) server communication load, and (c) commitment size of each user.

TABLE II
THE COMPARISON OF BYZSECAGG AND BREA [24] IN TERMS OF COMPUTATIONAL COMPLEXITY

Approach	Server computational complexity	Per-user computational complexity
BREA [24]	$\mathcal{O}((N^3 + NL) \log^2 N)$	$\mathcal{O}(NL \log^2 N + N^2 L)$
ByzSecAgg	$\mathcal{O}((N^3 + \frac{NL}{K}) \log^2 N)$	$\mathcal{O}(\frac{2NL}{K} \log^2 N + N^2 \frac{L}{K} + N^2 \log^2 N)$

such as ramp secret sharing and coded computing to reduce communication loads and enable secure computation of pairwise distances, which are used for distance-based outlier detection algorithms. Additionally, we use a linear commitment scheme with a constant commitment size to ensure message integrity during the protocol and protect against adversarial behaviors of Byzantine users. In terms of communication loads, ByzSecAgg outperforms the baseline scheme BREA and offers the advantage of a constant commitment size, regardless of local update sizes. Furthermore, it achieves the same level of performance in terms of convergence and resilience properties.

REFERENCES

- [1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, vol. 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282, PMLR, 2017.
- [2] 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.
- [3] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.
- [4] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Advances in Neural Information Processing Systems*, vol. 32, pp. 14747–14756, 2019.
- [5] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients - how easy is it to break privacy in federated learning?," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 16937–16947, 2020.
- [6] 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*, pp. 1175–1191, 2017.
- [7] J. So, B. Güler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 10, pp. 1860–1873, 2019.

- on Selected Areas in Information Theory, vol. 2, no. 1, pp. 479–489, 2021.
- [8] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, “Secure single-server aggregation with (poly) logarithmic overhead,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1253–1269, 2020.
- [9] B. Choi, J.-y. Sohn, D.-J. Han, and J. Moon, “Communication-computation efficient secure aggregation for federated learning,” *arXiv preprint arXiv:2012.05433*, 2020.
- [10] Y. Zhao and H. Sun, “Information theoretic secure aggregation with user dropouts,” in *Proceedings of the 2021 IEEE International Symposium on Information Theory (ISIT)*, pp. 1124–1129, 2021.
- [11] R. Schlegel, S. Kumar, E. Rosnes, and A. G. i. Amat, “CodedPaddedFL and CodedSecAgg: Straggler mitigation and secure aggregation in federated learning,” *IEEE Transactions on Communications*, vol. 71, no. 4, pp. 2013–2027, 2023.
- [12] J. So, C. He, C.-S. Yang, S. Li, Q. Yu, R. E. Ali, B. Guler, and S. Avestimehr, “LightSecAgg: a lightweight and versatile design for secure aggregation in federated learning,” in *Proceedings of Machine Learning and Systems*, vol. 4, pp. 694–720, 2022.
- [13] T. Jahani-Nezhad, M. A. Maddah-Ali, S. Li, and G. Caire, “SwiftAagg: Communication-efficient and dropout-resistant secure aggregation for federated learning with worst-case security guarantees,” in *Proceedings of the 2022 IEEE International Symposium on Information Theory (ISIT)*, pp. 103–108, 2022.
- [14] T. Jahani-Nezhad, M. A. Maddah-Ali, S. Li, and G. Caire, “SwiftAagg+: Achieving asymptotically optimal communication loads in secure aggregation for federated learning,” *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 4, pp. 977–989, 2023.
- [15] S. Shen, S. Tople, and P. Saxena, “Auror: Defending against poisoning attacks in collaborative deep learning systems,” in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pp. 508–519, 2016.
- [16] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, vol. 108 of *Proceedings of Machine Learning Research*, pp. 2938–2948, PMLR, 2020.
- [17] L. Lyu, H. Yu, X. Ma, C. Chen, L. Sun, J. Zhao, Q. Yang, and P. S. Yu, “Privacy and robustness in federated learning: Attacks and defenses,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2022.
- [18] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent,” in *Advances in Neural Information Processing Systems*, vol. 30, pp. 119–129, 2017.
- [19] E. M. El Mhamdi, R. Guerraoui, and S. Rouault, “The hidden vulnerability of distributed learning in Byzantium,” in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 3521–3530, PMLR, 2018.
- [20] C. Fung, C. J. Yoon, and I. Beschastnikh, “Mitigating sybils in federated learning poisoning,” *arXiv preprint arXiv:1808.04866*, 2018.
- [21] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, “Byzantine-robust distributed learning: Towards optimal statistical rates,” in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 5650–5659, PMLR, 2018.
- [22] C. Xie, S. Koyejo, and I. Gupta, “Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance,” in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97 of *Proceedings of Machine Learning Research*, pp. 6893–6901, PMLR, 2019.
- [23] C. Xie, S. Koyejo, and I. Gupta, “Zeno++: Robust fully asynchronous SGD,” in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119 of *Proceedings of Machine Learning Research*, pp. 10495–10503, PMLR, 2020.
- [24] 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.
- [25] Z. Zhang, J. Li, S. Yu, and C. Makaya, “SAFELearning: Enable backdoor detectability in federated learning with secure aggregation,” *arXiv preprint arXiv:2102.02402*, 2021.
- [26] R. K. Velicheti, D. Xia, and O. Koyejo, “Secure Byzantine-robust distributed learning via clustering,” *arXiv preprint arXiv:2110.02940*, 2021.
- [27] M. Rathee, C. Shen, S. Wagh, and R. A. Popa, “ELSA: Secure aggregation for federated learning with malicious actors,” in *Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP)*, pp. 1961–1979, IEEE, 2023.
- [28] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding up distributed machine learning using codes,” *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2017.
- [29] Q. Yu, M. A. Maddah-Ali, and S. Avestimehr, “Polynomial codes: an optimal design for high-dimensional coded matrix multiplication,” in *Advances in Neural Information Processing Systems*, pp. 4403–4413, 2017.
- [30] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, “Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding,” *IEEE Transactions on Information Theory*, vol. 66, no. 3, pp. 1920–1933, 2020.
- [31] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, “On the optimal recovery threshold of coded matrix multiplication,” *IEEE Transactions on Information Theory*, vol. 66, no. 1, pp. 278–301, 2019.
- [32] T. Jahani-Nezhad and M. A. Maddah-Ali, “CodedSketch: A coding scheme for distributed computation of approximated matrix multiplication,” *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 4185–4196, 2021.
- [33] S. Dutta, V. R. Cadambe, and P. Grover, “Short-dot: Computing large linear transforms distributedly using coded short dot products,” in *Advances in Neural Information Processing Systems*, vol. 29, pp. 2092–2100, 2016.
- [34] N. Ferdinand and S. C. Draper, “Hierarchical coded computation,” in *Proceedings of the 2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 1620–1624, IEEE, 2018.
- [35] W.-T. Chang and R. Tandon, “On the capacity of secure distributed matrix multiplication,” in *Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2018.
- [36] Z. Jia and S. A. Jafar, “On the capacity of secure distributed batch matrix multiplication,” *IEEE Transactions on Information Theory*, vol. 67, no. 11, pp. 7420–7437, 2021.
- [37] R. G. D’Oliveira, S. El Rouayheb, and D. Karpuk, “GASP codes for secure distributed matrix multiplication,” *IEEE Transactions on Information Theory*, vol. 66, no. 7, pp. 4038–4050, 2020.
- [38] T. Tang, R. E. Ali, H. Hashemi, T. Gangwani, S. Avestimehr, and M. Annavaram, “Adaptive verifiable coded computing: Towards fast, secure and private distributed machine learning,” in *Proceedings of the 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 628–638, IEEE, 2022.
- [39] H. Akbari-Nodehi and M. A. Maddah-Ali, “Secure coded multi-party computation for massive matrix operations,” *IEEE Transactions on Information Theory*, vol. 67, no. 4, pp. 2379–2398, 2021.
- [40] S. R. H. Najarkolaei, M. A. Maddah-Ali, and M. R. Aref, “Coded secure multi-party computation for massive matrices with adversarial nodes,” in *Proceedings of the 2020 Iran Workshop on Communication and Information Theory (IWCIT)*, pp. 1–6, IEEE, 2020.
- [41] 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 *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1215–1225, PMLR, 2019.
- [42] M. Soleymani, R. E. Ali, H. Mahdavifar, and A. S. Avestimehr, “List-decodable coded computing: Breaking the adversarial toleration barrier,” *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 3, pp. 867–878, 2021.
- [43] J. Katz and Y. Lindell, *Introduction to Modern Cryptography: Principles and Protocols*. Chapman and Hall/CRC, 2008.
- [44] G. R. Blakley and C. Meadows, “Security of ramp schemes,” in *Advances in Cryptology, Proceedings of CRYPTO ’84*, vol. 196 of *Lecture Notes in Computer Science*, pp. 242–268, Springer, 1984.
- [45] K. Nazirkhanova, J. Neu, and D. Tse, “Information dispersal with provable retrievability for rollups,” in *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, pp. 180–197, ACM, 2022.
- [46] M. Fang, X. Cao, J. Jia, and N. Gong, “Local model poisoning attacks to Byzantine-robust federated learning,” in *29th USENIX Security Symposium*, pp. 1605–1622, USENIX Association, 2020.
- [47] C. Xie, O. Koyejo, and I. Gupta, “Fall of empires: Breaking Byzantine-tolerant sgd by inner product manipulation,” in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, vol. 115 of *Proceedings of Machine Learning Research*, pp. 261–270, PMLR, 2020.
- [48] S. Huang, Y. Li, C. Chen, L. Shi, and Y. Gao, “Multi-metrics adaptively identifies backdoors in federated learning,” in *Proceedings of the*

- IEEE/CVF International Conference on Computer Vision*, pp. 4652–4662, 2023.
- [49] H. Wang, K. Sreenivasan, S. Rajput, H. Vishwakarma, S. Agarwal, J. Sohn, K. Lee, and D. S. Papailiopoulos, “Attack of the tails: Yes, you really can backdoor federated learning,” in *Advances in Neural Information Processing Systems*, vol. 33, pp. 16070–16084, 2020.
- [50] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37 of *Proceedings of Machine Learning Research*, pp. 1737–1746, PMLR, 2015.
- [51] S. Lin and D. J. Costello, *Error control coding*, vol. 2. Prentice hall New York, 2001.
- [52] A. Kate, G. M. Zaverucha, and I. Goldberg, “Constant-size commitments to polynomials and their applications,” in *Proceedings of the International conference on the theory and application of cryptology and information security*, pp. 177–194, Springer, 2010.
- [53] V. Shoup, *A computational introduction to number theory and algebra*. Cambridge university press, 2005.
- [54] K. S. Kedlaya and C. Umans, “Fast polynomial factorization and modular composition,” *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1767–1802, 2011.
- [55] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.
- [56] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [57] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

Giuseppe Caire (IEEE Fellow) was born in Torino in 1965. He received a B.Sc. in Electrical Engineering from Politecnico di Torino in 1990, an M.Sc. in Electrical Engineering from Princeton University in 1992, and a Ph.D. from Politecnico di Torino in 1994. He has been a post-doctoral research fellow with the European Space Agency (ESTEC, Noordwijk, The Netherlands) in 1994–1995, Assistant Professor in Telecommunications at the Politecnico di Torino, Associate Professor at the University of Parma, Italy, Professor with the Department of Mobile Communications at the Eurecom Institute, Sophia-Antipolis, France, a Professor of Electrical Engineering with the Viterbi School of Engineering, University of Southern California, Los Angeles, and he is currently an Alexander von Humboldt Professor with the Faculty of Electrical Engineering and Computer Science at the Technical University of Berlin, Germany.

He received the Jack Neubauer Best System Paper Award from the IEEE Vehicular Technology Society in 2003, the IEEE Communications Society and Information Theory Society Joint Paper Award in 2004 and in 2011, the Okawa Research Award in 2006, the Alexander von Humboldt Professorship in 2014, the Vodafone Innovation Prize in 2015, an ERC Advanced Grant in 2018, the Leonard G. Abraham Prize for best IEEE JSAC paper in 2019, the IEEE Communications Society Edwin Howard Armstrong Achievement Award in 2020, the 2021 Leibniz Prize of the German National Science Foundation (DFG), and the CTTC Technical Achievement Award of the IEEE Communications Society in 2023. Giuseppe Caire is a Fellow of IEEE since 2005. He has served in the Board of Governors of the IEEE Information Theory Society from 2004 to 2007, and as officer from 2008 to 2013. He was President of the IEEE Information Theory Society in 2011. His main research interests are in the field of communications theory, information theory, channel and source coding with particular focus on wireless communications.

Tayyebeh Jahani-Nezhad received her B.Sc. degree in Electrical Engineering and M.Sc. degree in Communication Systems from Isfahan University of Technology, Iran, in 2015 and 2017, respectively. She earned her Ph.D. in Communication Systems from Sharif University of Technology, Iran, in 2022. She is currently a Postdoctoral Researcher at the Communications and Information Theory Chair (CommIT), Technische Universität Berlin, Germany. Her research focuses on developing efficient and secure distributed learning frameworks, particularly in coded distributed computing and federated learning.

Mohammad Ali Maddah-Ali (IEEE Fellow, 2023) is an Associate Professor at the University of Minnesota Twin Cities. He received his B.Sc. degree in Electrical Engineering from Isfahan University of Technology, his M.A.Sc. degree from the University of Tehran, and his Ph.D. in Electrical and Computer Engineering from the University of Waterloo, Canada, in 2007.

From 2007 to 2008, he was with the Wireless Technology Laboratories at Nortel Networks, Ottawa, ON, Canada. He then held a Postdoctoral Fellowship at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, from 2008 to 2010. From September 2010 to September 2020, he served as a Communication Research Scientist at Nokia Bell Labs, NJ, USA.

Dr. Maddah-Ali is the recipient of several honors, including the NSERC Postdoctoral Fellowship (2007), the Best Paper Award at the IEEE International Conference on Communications (ICC) in 2014, the IEEE Communications Society and IEEE Information Theory Society Joint Paper Award in 2015, and the IEEE Information Theory Society Paper Award in 2016. He served as an Associate Editor for the IEEE Transactions on Information Theory (2019–2022) and as Lead Editor for the IEEE Journal on Selected Areas in Information Theory. He is currently a distinguished lecturer of the IEEE Information Theory Society.