Enhanced Security and Privacy via Fragmented Federated Learning

Najeeb Moharram Jebreel, Josep Domingo-Ferrer, Fellow, IEEE, Alberto Blanco-Justicia, and David Sánchez

Abstract—In federated learning (FL), a set of participants share updates computed on their local data with an aggregator server that combines updates into a global model. However, reconciling accuracy with privacy and security is a challenge to FL. On the one hand, good updates sent by honest participants may reveal their private local information, whereas poisoned updates sent by malicious participants may compromise the model’s availability and/or integrity. On the other hand, enhancing privacy via update distortion damages accuracy, whereas doing so via update aggregation damages security because it does not allow the server to filter out individual poisoned updates. To tackle the accuracy-privacy-security conflict, we propose fragmented FL (FFL), in which participants randomly exchange and mix fragments of their updates before sending them to the server. To achieve privacy, we design a lightweight protocol that allows participants to privately exchange and mix encrypted fragments of their updates so that the server can neither obtain individual updates nor link them to their originators. To achieve security, we design a reputation-based defense tailored for FFL that builds trust in participants and their mixed updates based on the quality of the fragments they exchange and the mixed updates they send. Since the exchanged fragments’ parameters keep their original coordinates and attackers can be neutralized, the server can correctly reconstruct a global model from the received mixed updates without accuracy loss. Experiments on real data sets show that FFL can prevent semi-honest servers from mounting privacy attacks, can effectively counter-poisoning attacks, and can keep the accuracy of the global model.

Index Terms—Accuracy, federated learning (FL), fragmentation, privacy, security.

I. INTRODUCTION

FEDERATED learning (FL, [1]) enables multiple participants to jointly train a machine learning (ML) model without sending their data to a central server. In FL, participants compute model updates based on their local data and a global model received from a coordination server, to whom they send the resulting updates. The server aggregates the received updates to obtain a new global model, which is redistributed among the participants. This collaboration entails mutual benefits for all parties: 1) the participants and the server obtain more accurate models due to learning from joint training data; 2) participants keep their private local data in their devices; and 3) the server distributes the computational load of training across the participants’ devices (e.g., smartphones) [2]. Since training data do not leave the participants’ devices, FL is a suitable option for scenarios dealing with personal data, such as facial recognition [3], voice assistants [4], healthcare [5], next-word prediction [6], and intrusion detection in IoT networks [7], or in case data collection and processing are restricted due to privacy protection laws such as the GDPR [8]. Despite these advantages, FL is vulnerable to privacy and security attacks [9], [10].

Regarding privacy, several works [11], [12], [13] have demonstrated that a semi-honest server can analyze individual updates to infer sensitive information on a participant’s local training data. Recent powerful privacy attacks show that it is possible to reconstruct the original training data by inverting the gradients of updates [14], [15], [16].

Regarding security, FL is vulnerable to poisoning attacks [17]. Since the server has no control over the behavior of participants, any of them may deviate from the prescribed training protocol to attack the model by conducting either untargeted poisoning (i.e., Byzantine) attacks [18], [19] or targeted poisoning attacks [20], [21], [22]. In the former type of attacks, the attacker aims to degrade the model’s overall performance, whereas, in the latter, he aims to cause the global model to misclassify some attacker-chosen inputs.

Some solutions have been proposed to prevent the server from analyzing individual updates or linking them to their originators. These involve well-known privacy-enabling methods: differential privacy (DP) [23], homomorphic encryption (HE) [24], and secure multiparty computation (SMC) [25]. DP-based methods protect the participants’ data by injecting random noise into the parameters of updates at the cost of sacrificing the accuracy of the global model [26], [27]. On the other hand, HE- and SMC-based methods securely aggregate the updates of participants before sending them to the server. Yet, HE and SMC have a high-computational cost and they prevent the server from inspecting individual updates, which makes approaches based on these techniques...
vulnerable to security attacks. It is noted that countermeasures against security attacks require direct access by the server to individual updates to detect and/or filter out those that are poisoned [18], [28], [29].

Simultaneously achieving privacy, security, and accuracy is a tough challenge for FL [9], [17], [30]. In fact, this is one of several challenges in ML due to contradicting requirements (see [31] about other conflicts).

Our goal is to address the following puzzle: “Can we prevent a semi-honest server from performing privacy attacks on individual updates while learning an accurate global model and ensuring protection against security attacks?”

To this end, we propose fragmented FL (FFL), a framework in which participants randomly exchange fragments of updates among them before sending them to the server. Our work brings the following contributions.

1) We propose a novel lightweight protocol that: i) allows participants to privately exchange and mix random fragments of their updates; ii) enables the server to correctly aggregate the global model from the mixed updates; and iii) prevents the server from recovering the complete original updates or linking them to their originators.

2) We propose a new reputation-based defense tailored to FFL against security attacks. Specifically, the server selects participants for training and adaptively aggregates their mixed updates according to their global reputations. Also, honest participants do not exchange fragments with participants with low local reputations. Reputations are computed based on the quality of the updates the participants send and the fragments they exchange.

3) We provide extensive theoretical and empirical analyses to assess the accuracy, privacy, and security offered by FFL, and we quantify the computation overhead and communication cost it incurs.

The remainder of this article is organized as follows. Section II discusses related works. Section III introduces preliminary notions. Section IV describes the attacks being considered. Section V presents the FFL framework. Sections VI and VII provide privacy and security analyses of FFL. Section VIII details the experimental setup and evaluates our approach with respect to the accuracy, robustness against attacks, and runtime. Section IX gathers conclusions and proposes several lines of future research. Convergence and complexity are analyzed in the supplementary materials.

II. RELATED WORK

Most works in the FL literature tackle either robustness versus attacks against privacy or robustness versus attacks against security (poisoning attacks), but they do not consider both types of robustness simultaneously.

A. Private FL

On the privacy side, several works have been proposed to prevent the server from seeing individual updates. Bonawitz et al. [32] and So et al. [33] used secret sharing to hide individual updates. Aono et al. [34], Hardy et al. [35], and Zhang et al. [36] encrypted the local updates and use HE to compute the global model from the encrypted updates. However, both approaches are vulnerable to poisoning attacks because they hinder the analysis of individual updates. Other works adopt DP [4], [11], [37], which adds noise to local updates before sending them to the server. DP is practical but it only offers strong privacy guarantees for small values of $\epsilon$ that, due to the noise they add, significantly hamper the accuracy of the global model [26], [27].

B. Secure FL

On the security side, several attack-resistant aggregation defenses have been proposed, such as multi-Krum [18], Bulyan [38], FoolsGold [21], and DRACO [39]. However, they require direct access to the original individual updates, which interferes with protecting the privacy of participants. Although the median and the trimmed mean [28] can be applied on mixed updates, they are not suitable for high-dimensional models because their estimation error scales up with the size of the model in a square-root manner [40].

C. Private and Secure FL

Recently, the literature has witnessed a growing interest in achieving FL, that is, both privacy-preserving and secure. Naseri et al. [41] used DP to address both privacy and robustness against backdoor attacks. However, it does not deal with the tradeoff between privacy and accuracy, and it does not consider poisoning attacks different from backdoor attacks. PEFL [42] tries to address both privacy and security, but assumes there are two noncolluding servers that collaborate to filter out malicious updates while preventing each other from seeing individual updates. Moreover, PEFL builds on linear HE and a packing technique, and it involves exchanging the encrypted updates in four interacting protocols between the two servers for filtering and aggregation, which causes high communication and computation overheads. ShieldFL [43] also assumes two noncolluding servers, and uses a two-trapdoor HE scheme based on the Paillier cryptosystem to achieve both secure and privacy-preserving FL. In ShieldFL, the two servers execute three interactive protocols to compute the cosine similarity of the local updates in the ciphertext. They then use the computed cosine similarities to filter out potential poisoned updates. Unfortunately, this also imposes significant computation and communication costs on the participating entities. BREA [44] proposes a single-server framework where each participant secret-shares her local update with all the other participants in the system. Also, each participant locally computes the participant-wise Euclidean distances to the shares of all participants, and then sends the computed distances to the server, which uses them to filter out potential poisoned updates. The server then selects the potential good updates and asks the participants to locally aggregate the selected good shares and upload the aggregates to the server. Finally, the server reconstructs the global model from the received aggregates and sends it to the participants in the next training round. Although this work has the advantage of being single server, it imposes high computation
and communication overheads on the participating parties. Ma et al. [45] integrated local DP on the participant’s side with an intermediate shuffler between the participants and the aggregator server to achieve privacy. Besides, they use a Byzantine-robust stochastic aggregation algorithm at the server side to achieve security. However, this work is subject to the inevitable tradeoff between privacy and accuracy due to the use of DP. In addition, the shuffler is a third party, and, hence, its honesty is not guaranteed. SAFELeaming [46] is based on the work of [32] to support backdoor detection and privacy-preserving aggregation simultaneously. To this end, it randomly divides participants into subgroups, securely aggregates a submodel for each subgroup and filters out malicious submodels instead of individual models. However, this approach faces a tradeoff of another kind: the smaller the number of participants in a subgroup, the less privacy for these participants; conversely, the larger the number of participants, the easier it is for the attacker to hide her malicious model amid honest ones. This article proposes to disclose part of the parameters of the aggregated submodels, leading to another privacy/security tradeoff. Domingo-Ferrer et al. [47] proposed a co-utile FL to solve the accuracy-privacy-security conflict. The proposed solution preserves the global model accuracy and allows defending against security attacks, but its privacy is limited to only breaking the link between local updates and their originators. Although this provides a level of privacy protection, a semi-honest server still has direct access to original unlabeled updates. Hence, it can use them to perform several privacy attacks, such as membership inference attacks (MIAs) and reconstruction attacks. Chen et al. [48] and Zhang et al. [49] employed trusted execution environments (TEEs) on the participants (for local training) and on the servers (for secure aggregation) to achieve accurate and privacy-preserving FL. However, the limited memory size of current TEEs makes them impractical for large DL models or large-scale FL systems. Besides, the authors of those papers do not consider data poisoning attacks such as label-flipping attacks, and assume trust in the manufacturers of the TEEs, which seems too strong an assumption.

III. PRELIMINARIES

A. Deep Neural Networks

A deep neural network (DNN) is a function \( F(x) \), obtained by composing \( L \) functions \( f_l, l \in [1, L] \), that maps an input \( x \in \mathbb{R}^n \) to an output \( y \in \mathbb{R}^2 \). Each \( f_l \) is a layer, that is, parameterized by a weight matrix \( w_l \), a bias vector \( b_l \), and an activation function \( \sigma_l \). In this article, we use predictive DNNs as \( z \)-class classifiers.

B. Federated Learning

In federated learning (FL), \( K \) participants and an aggregator server \( A \) collaboratively build a global model \( W \). In each training round \( t \in [1, T] \), the aggregator randomly selects a subset of participants \( S \) of size \( n = C \cdot K \geq 1 \) where \( K \) is the total number of participants in the system, and \( C \) is the fraction of participants that are selected in \( t \). After that, \( A \) distributes the current global model \( W^t \) to all participants in \( S \). Besides \( W^t \), \( A \) sends a set of hyperparameters to be used to train the local models, which includes the number of local epochs \( E \), the local batch size \( BS \), and the learning rate \( \eta \). After receiving \( W^t \), each participant \( k \) divides her local data into batches of size \( BS \) and performs \( E \) local training epochs on her data to compute her local update \( W_k^{t+1} \). Finally, participants upload their updates to \( A \), which aggregates them into a new global model \( W^{t+1} \). The federated averaging algorithm (FedAvg) [1] is usually employed to perform the aggregation, and it is defined as

\[
W^{t+1} = \frac{1}{K} \sum_{k=1}^{K} d_k W_k^{t+1}
\]

where \( d_k \) is the number of data points locally held by worker \( k \), and \( d \) is the total number of data points locally held by the \( K \) workers, that is, \( d = \sum_{k=1}^{K} d_k \). Note that FedAvg is the standard way to aggregate updates in FL and is not meant to counter security attacks.

IV. ATTACK MODELS

A. Privacy Attack Model

We focus on a semi-honest server \( A^* \), which follows the protocol honestly, but tries to infer information about the private local data of by participants for training. Even though privacy attacks may also be orchestrated by participants based on the successive global models, the performance of such attacks is quite limited and degrades significantly as the number of participants increases [13]. Server-side attacks are much stronger, especially, when the server sees local updates individually and can link them to their originators. In particular, MIAs aim to determine if a specific example is part of the training data set [12], [13]. Property inference attacks try to infer specific properties about the training data [13], [50]. Distribution estimation attacks aim to obtain examples from the same distribution of the participants’ training data [51]. Finally, reconstruction attacks are the most ambitious in that they attempt to extract the original training data from a participant’s local update [14], [15], [16]. To mount any of these attacks, the server needs to access individual local updates. Therefore, our goal is to disable privacy attacks by preventing the server from obtaining the participants’ original updates.

B. Security Attack Model

Since the server has no control over the participants’ behavior, a malicious participant may deviate from the prescribed training protocol and attack the global model. Depending on the attacker’s objective, security attacks can be divided into untargeted attacks [20], [52], [53] and targeted attacks [22], [54], [55]. The former are attacks against model availability (e.g., to prevent the model from converging), whereas the latter are against model integrity (e.g., to fool the global model into making incorrect predictions on some attacker-chosen inputs). Security attacks can be performed in two ways: 1) model poisoning 2) data poisoning. In model poisoning [18], [19], [22], the attacker manipulates the model parameters before sending her update to the server. In data poisoning [20], [29], the attacker injects bad or biased data into her training data.
set before training her local model. In our work, we consider a number of attackers \( K' \leq K/5 \), that is, no more than 20% of the \( K \) participants in the system. Although some works in the literature assume larger percentages of attackers, finding more than 20% of attackers in real-world FL scenarios is highly unlikely. For example, with the millions of users of Gboard [6], controlling even a small percentage of user devices requires the attacker(s) to compromise a large number of devices, which demands huge effort and resources and is, therefore, impractical. We assume the \( K' \) attackers carry out two types of attacks: 1) untargeted attacks based on Gaussian noise [19], [57] and 2) targeted attacks based on label-flipping [20], [29]. Furthermore, we assume that the attacker(s) have no control over the server or the honest participants.

V. FRAGMENTED FEDERATED LEARNING

In this section, we present the FFL framework. First, we give an overview of our framework and then present its design and protocols in detail. Table I summarizes the notation used in this article.

A. Overview

Fig. 1 shows an overview of the FFL framework. The key idea is to have the participants randomly fragment and mix their updates before sending them to the server. Specifically, two participants agree on some symmetric random indices in their update vectors and exchange the parameters of those indices after they are encrypted. The participants then send the encrypted mixed updates to the server instead of their original updates. Since exchanging parameters is done without changing their original coordinate positions, the server can calculate the average of the mixed updates after decrypting the encrypted mixed updates and obtain the same updated global model that would result from averaging the original updates. Sending mixed updates instead of the original ones breaks the link between the updates and their originators, and does not give the server direct access to individual updates. Thus, FFL prevents a semi-honest server from mounting powerful privacy attacks. Also, the global model’s accuracy is preserved because the parameters of the exchanged fragments are kept unaltered. Yet, exchanging fragments between participants should neither impose significant communication or computation overheads nor protect against server-side privacy attacks at the price of facilitating participant-side privacy attacks. Thus, we design a fragment exchanging protocol based on lightweight cryptographic tools to: 1) allow the participants to exchange and mix random fragments of their updates while incurring very minor communication and computation overheads; 2) prevent participants from seeing each other’s original fragments; and 3) allow the server to correctly compute the updated global model from the mixed updates without being able to recover the individual original updates or link them to their originators. However, averaging updates to compute the global model goes against countering security attacks. Also, the design of the exchange protocol gives \( n' \) attackers the chance of poisoning \( 2n' \) coordinates. We tackle both issues by designing a novel reputation-based defense tailored for FFL that builds trust in participants based on the quality of the mixed updates they send and the fragments they exchange. Specifically, the server holds a global reputation vector and uses it to select
participants for training and adaptively aggregate their mixed updates. A participant who repeatedly sends poisoned mixed updates will have a lower global reputation, and, thus, will not be selected in future training rounds. Also, the mixed updates she sends will have little to no influence on the global model aggregation. On the other hand, each participant holds a local reputation vector and uses it to decide whether to exchange fragments with the other participants. The local reputation increases or decreases based on the quality of the fragments exchanged by participants. An attacker who exchanges poisoned fragments with honest participants will eventually find no honest participant to exchange with. The fragments exchanged by participants. An attacker who

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure}
\caption{Fragmented Federated Learning}
\end{figure}

\textbf{Protocol 1} Fragmented Federated Learning

\begin{enumerate}
\item \textbf{Input:} $K, C, BS, E, \eta, T$
\item \textbf{Output:} $W^T$, the global model after $T$ training rounds
\item $A$ initializes $W^0, \gamma^0 = \{\gamma^0_k = 0\}_{k=1}^K$;
\item \textbf{for each participant} $k \in [1, K]$ do
\item \quad $k$ initializes $\zeta^0_k = \{\zeta^0_{k,j} = 0\}_{j=1}^K$;
\item \textbf{end}
\item \textbf{for each round} $t \in [0, T - 1]$ do
\item \quad $S \leftarrow \text{SELECT_PARTICIPANTS}(C, \gamma^t)$;
\item \quad $A$ sends $W^t$ to all participants in $S$
\item \quad \textbf{for each participant} $k \in S$ in parallel do
\item \quad \quad $k$ calls $(W_k^{t+1})_{mix}', \text{Enc}_{pk_A}(s_{r_j}) \leftarrow \text{PARTICIPANT_UPDATE}(k, W^t)$;
\item \quad \quad $k$ sends $(W_k^{t+1})_{mix}', \text{Enc}_{pk_A}(s_{r_j})$ to $A$;
\item \quad \quad $A$ decrypts $\text{Enc}_{pk_A}(s_{r_j})$ with its private key to obtain $s_{r_j}$;
\item \quad \quad $A$ generates $r_j \leftarrow \text{PRNG}(s_{r_j})$;
\item \quad \quad $A$ decrypts $(W_k^{t+1})_{mix}'$ with $r_j$ as $(W_k^{t+1})_{mix} \leftarrow (W_k^{t+1})_{mix}' \oplus r_j$;
\item \quad \textbf{end}
\item \quad \text{COMPUTE_SIMILARITY}();
\item \quad \text{UPDATE_REPUTATIONS}();
\item \quad Let $\nu^t$ be the computed trust vector from Expression (9);
\item \quad $A$ aggregates $W^t' \leftarrow \frac{1}{\sum_{k \in S} d_k} \sum_{k} \nu_k^t(W_k^{t+1})_{mix}$;
\item \textbf{end}
\item \textbf{Function} \text{PARTICIPANT_UPDATE}(k, W^t)
\item \quad $W_k \leftarrow W^t$;
\item \quad \textbf{for each local epoch} $e \in [1, E]$ do
\item \quad \quad \textbf{for each batch} $\beta$ of size $BS$ do
\item \quad \quad \quad $W_k \leftarrow W_k - \eta \nabla L(W_k, \beta)$;
\item \quad \quad \textbf{end}
\item \quad \textbf{end}
\item \quad $W_k \leftarrow d_k W_k$;
\item \quad exchanged=false;
\item \quad Let $Q_{\nu^t_k}$ be the first quartile in $\nu^t_k$;
\item \quad Let $S_{\nu^t_k}$ be the set of participants with local reputations greater than $Q_{\nu^t_k}$;
\item \quad \textbf{while} not exchanged do
\item \quad \quad $j \leftarrow$ select a random participant of $S_{\nu^t_k}$;
\item \quad \quad \textbf{if} $\zeta^t_{j,k} \leq Q_{\nu^t_j}$ then
\item \quad \quad \quad $(W_{mix})', \text{Enc}_{pk_A}(s_{r_j}) \leftarrow \text{EXCHANGE_FRAGMENTS}(k, j)$;
\item \quad \quad \quad exchanged=true;
\item \quad \quad \textbf{return} $(W_{mix})', \text{Enc}_{pk_A}(s_{r_j})$;
\item \quad \quad \textbf{end}
\item \quad \textbf{end}
\item \textbf{end}
\end{enumerate}

1) \textbf{Adaptive Selection of Participants:} At each round $t$, $A$ uses Procedure 1 to select a set $S$ of potential honest participants. First, the first quartile of the global reputation $Q_{1,\gamma}$ is computed. Then, participants with a global reputation greater than or equal to $Q_{1,\gamma}$ are assigned to the candidate set $S_c$. Finally, the server selects a random set $S$ of $n = \max(C \cdot \lfloor S_c \rfloor, 2)$ participants from $S_c$. As we explain later in this section, we expect the attackers (up to 20%) to have global reputations less than $Q_{1,\gamma}$. Note that, in the standard FL setting, $n = \max(C \cdot K, 1)$, but we replace 1 by 2 because FFL needs at least 2 participants to be applicable. After that, $A$ sends the global model $W^t$ to the participants in $S$ through a secure channel. The server also adds the pseudonyms of the selected participants to a public board seen by all participants.

2) \textbf{Local Training:} Each participant $k \in S$ trains the received model on her private local data to obtain her local update $W_k$. Then, $k$ scales her computed local update by the number of data points she holds, $d_k$. The next step is for $k$ to randomly select another participant $j$ from the public board to exchange a fragment of her encrypted update $W_k$ with her.

3) \textbf{Participant Selection for Exchange:} To avoid making herself a bridge for poisoning the global model, $k$ uses her
local reputation vector \( \zeta_j^t \) to decide whether to exchange fragments with any other participant \( j \). To do so, \( k \) computes the first quartile of her local reputation vector \( Q_1c_k \) and assigns the participants in the public board that have local reputations greater than \( Q_1c_k \) to the set \( Sc_k \). Then, \( k \) selects a random participant \( j \) from \( Sc_k \) and asks her to exchange fragments. When \( j \) receives the request from \( k \) for fragment exchange, she checks the local reputation of \( k \), \( \zeta_j^t \), and if \( \zeta_j^t < Q_1c_j^t \), \( j \) rejects \( k \)'s request for exchange. Otherwise, \( j \) accepts to exchange fragments with \( k \), and they call protocol EXCHANGE_FRAGMENTS. At the end of the protocol, both \( k \) and \( j \) obtain two encrypted mixed updates with their corresponding encrypted one-time pad (OTP) seeds: \( W_k \)'s mixed update and Enc\_{\text{pk}_s}(s_{r_j}) for participant \( k \), and \( W_j \)'s mixed update and Enc\_{\text{pk}_s}(s_{r_k}) for participant \( j \), and send them to \( A \). Note that the seeds to generate the mixed updates’ OTPs are encrypted under \( A \)'s public key and swapped between \( k \) and \( j \). Later in this section, we detail how local reputations are computed.

4) Fragments Exchanging and Mixing: The protocol for exchanging fragments is key in our approach. We design this protocol to privately and efficiently exchange fragments of updates between participants. Specifically, a participant \( k \) (the Initiator) seeks to exchange a random fragment of her update with another participant \( j \) (the Acceptor). Both participants leverage a key exchange protocol to jointly generate two complementing masks used to fragment their updates. The fragments are then encrypted using OTPs, that is, random sequences added to the fragments, that allow Initiator and Acceptor to combine their updates but prevent each other from accessing their counterpart’s update parameters. These OTPs can only be removed by the central server after they have been mixed using the secret information provided by both participants.

Let PRNG(\( \cdot \)) be a public pseudo-random number generator that takes an integer as a seed. Let \( \mathbb{G}_p \) be the multiplicative group of integers modulo a large prime \( p \) (2048-bit long), and \( g \) a generator of the group. All subsequent operations with vectors are performed coordinate-wise.

Protocol EXCHANGE_FRAGMENTS is as follows.

1) Participant \( k \) randomly generates integers \( a, s_{r_k}, \) and \( s_{pk}, \) and sends \( g^a \mod p \) and Enc\_{pk_s}(s_{r_k}) to \( j \).

2) Participant \( j \) generates integers \( b, s_{r_j}, \) and \( s_{pk} \) and computes \( g^b \mod p \), \( r_j = \text{PRNG}(s_{r_j}) \), and \( r_j = \text{PRNG}(s_{pk}) \), with the last two numbers having bitlength \( |W| \lambda \) (the bitlength of an update, that is, the number of parameters times the bitlength of a parameter). \( j \) generates a mask \( m = \text{PRNG}(g^a \mod p) \) of 0’s and 1’s of bitlength \( |W| \) (equal to the number of update parameters) and its 1-complement inverse mask \( \neg m \) such that \( m \oplus \neg m = 1_{|W|} \) (adding modulo 2 a mask to its 1-complement inverse yields the all ones mask). Then, \( j \) sends \( g^b \mod p \), Enc\_{pk_s}(s_{r_j}), W_j \oplus r_j \oplus \rho_j \), and \( (W_j \oplus \neg m) \oplus \rho_j \) to \( k \), where \( \oplus \) is an operator between an update and a mask that preserves the \( i \)th update parameter if the \( i \)th mask bit is 1, and clears to 0 the \( i \)th update parameter if the \( i \)th mask bit is 0.

The random uniform binary mask \( m \) will be used to fragment both \( j \)'s and \( k \)'s updates. Also, both \( r_j \) and \( \rho_j \) are OTPs known to \( j \) only and used by \( j \) to hide her original fragments from \( k \).

3) \( k \) computes \( r_k = \text{PRNG}(s_{r_k}) \) and \( \rho_k = \text{PRNG}(s_{pk}) \) both of length \( |W| \lambda \), generates \( m = \text{PRNG}(g^a \mod p) \) and its inverse \( \neg m \). Then, \( k \) computes her encrypted mixed update as follows:

\[
(W_k)'_{\text{mix}} = (W_k)_{\text{mix}} \oplus r_j \oplus m \\
= W_k \oplus (W_k \oplus m) \\
\oplus (W_j \oplus r_j \oplus \rho_j) \oplus ((W_j \oplus \neg m) \oplus \rho_j).
\]

(2)

In Expression (2), the result of subexpression \( W_k \oplus (W_k \oplus m) \) is to clear to 0 all parameters of \( W_k \) at coordinates where the mask \( m \) has a 1. On the other hand, subexpression \( W_j \oplus (W_j \oplus \neg m) \) clears to 0 all parameters of \( W_j \) where the mask \( m \) has a 0. Bitwise adding both subexpressions yields the mixed update \( (W_k)'_{\text{mix}} \). Note that the two appearances of \( \rho_j \) cancel each other and \( r_j \) encrypts the mixed update into \( (W_k)'_{\text{mix}} \). Since the mask \( m \) is random, the mixed update can be expected to contain the same number of 1s and 0s. Hence, \( (W_k)'_{\text{mix}} \) can be expected to contain half of the coordinates from \( W_j \) and the other half from \( W_k \). Another important remark is that the mixed update is encrypted using \( r_j \), whereas \( k \) has only Enc\_{pk_s}(s_{r_j}), so \( k \) cannot obtain the full cleartext mixed update.

After that, \( k \) sends \( (W_k \oplus r_k \oplus \rho_k) \), and \( (W_k \oplus \neg m) \oplus \rho_k \) to \( j \). She also sends \( (W_j)'_{\text{mix}}, \text{Enc}_{pk_s}(s_{r_j}) \) to the server. Receiving Enc\_{pk_s}(s_{r_k}) allows the server to decrypt the encrypted mixed update but does not allow it to extract any original fragment separately or link it to the fragment’s originator, as we show in Appendix VI. Moreover, the use of the Diffie-Hellman key exchange method [58] to generate seed \( g^{ab} \) ensures that no one but \( k \) and \( j \) knows the generated mask \( m \).

4) Similarly, \( j \) computes her encrypted mixed update as

\[
(W_j)'_{\text{mix}} = (W_j)_{\text{mix}} \oplus r_k \oplus m \\
= W_j \oplus (W_j \oplus m) \\
\oplus (W_k \oplus r_k \oplus \rho_k) \oplus ((W_k \oplus \neg m) \oplus \rho_k).
\]

(3)

Finally, \( j \) sends \( (W_j)'_{\text{mix}}, \text{Enc}_{pk_s}(s_{r_k}) \) to the server.
A naive and simpler way to exchange fragments would be to encrypt the updates coordinate by coordinate using the server public key before exchanging fragments. However, this would cause significant communication and computation overheads for both the server and the participants. Instead, our protocol uses OTPs as a means to encrypt the mixed updates. In other words, we rely on symmetric-key encryption, which is more efficient when dealing with models that may contain millions of parameters.

5) Decryption of Encrypted Mixed Updates: When at training round \( t \) the server receives a mixed encrypted update \((W^{t+1})_{mix}^k\) and its corresponding encrypted seed \(Enc_{pk_A}(s_{r_j})\), it uses its private key to obtain the clear seed \(s_{r_j} = Dec_{sk_A}(Enc_{pk_A}(s_{r_j}))\). Then \(A\) regenerates \(r_j\) using \(s_{r_j}\) as a seed to \(PRNG(\cdot)\) and bitwise adds \(r_j\) to \((W^{t+1})_{mix}^k\) to obtain \((W^{t+1})_{mix}^k\). \(A\) does the same for all \(k \in S\) to get the plain mixed updates \((W^{t+1})_{mix}\) \text{for} \(k \in S\).

6) Computation of Reputation and Trust Values: We explain how reputations are computed and used to neutralize potential attackers in the system. First, the similarity between the mixed updates and their centroid is computed to measure their quality. Second, the global and local reputations are updated based on the computed similarities. Third, the global reputations are used to compute the trust values for the senders of the mixed updates, and these trust values finally weight the mixed updates when aggregating them to obtain the new global mixed updates.

\[ sim_{t} = \cos \varphi = \frac{(\nabla_{mix}^t)_{k}}{|(\nabla_{mix}^t)_{k}| \cdot |\text{med}_{mix}^t|}. \] (6)

This yields a cosine similarity value \( cs_{t+1}^k \in [-1, 1] \) for each participant’s mixed update. Note that poisoned mixed updates, being a minority, can be expected to have lower cosine similarity with \( \text{med}_{mix}^t \) than good mixed updates.

To compute a combined similarity value that captures the behaviors of both untargeted and targeted attacks, \(A\) performs the following steps.

1) Normalize and invert the computed distances into the range \([0, 1]\) as \(d_{sf}^t = 1 - d_{sf}^t / \max_{j \in S}(d_{sf}^t)\).
2) Normalize the cosine similarity vector \( cs^t \) into the range \([0, 1]\) as \( cs_{t+1}^k = (cs_{t+1}^k + 1)/2 \).
3) Compute the aggregated similarity value of \(k\) as \( sim_{t+1}^k = \alpha cs_{t}^k + \beta (Q_1_{\text{sim}}^t - Q_{1\text{sim}}^t) \) and obtains the set \( \{(\nabla_{mix}^t)_{k} \mid k \in S\} \). Then, \(A\) computes \(\text{med}_{mix}^t\) as the coordinate-wise median of the previous set. Since honest participants share the same objective and are a majority, the median of the mixed last-layer gradients is expected to lie in the same direction as the honest participants’ last-layer gradients. Thus, \(A\) computes the cosine similarity between \(\text{med}_{mix}^t\) and each \((\nabla_{mix}^t)_{k}\) as

\[ cs_{t+1}^k = \cos \varphi = \frac{(\nabla_{mix}^t)_{k} \cdot \text{med}_{mix}^t}{|\nabla_{mix}^t_{k}| \cdot |\text{med}_{mix}^t|}. \] (6)

This ensures that, when \(k\) obtains a small similarity value because of \(j\)’s poisoned fragment, she reduces \(j\)’s local reputation. As a result, if \(j\) exchanges poisoned fragments with the other participants, she will get a bad local reputation among all honest participants and thus become an outcast, so that no honest participant will accept to exchange fragments with him in the future.

c) Adaptive model aggregation: The server adaptively aggregates the received mixed updates using the global reputations of their senders. First, \(A\) computes the trust

\[ ds_{sf}^t = |\text{med}_{mix}^t - |(\nabla_{mix}^t)_{k}||| \].

For each participant \(k\), \(ds_{sf}^t\) represents how far the magnitude of the participant’s mixed update is from \(\text{med}_{mix}^t\). In the case of untargeted attacks, poisoned mixed updates are expected to deviate more from \(\text{med}_{mix}^t\) and, thus, result in larger \(ds_{sf}^t\) values. It would also be possible to compute distances based on the updates themselves and the median update rather than their magnitudes, but this would cause a higher computational overhead on the server. To capture the behavior of targeted attacks, \(A\) extracts the mixed gradients of the last layer \(L\) and computes the corresponding mixed gradients of the set

\[ \text{computes the corresponding mixed gradients of the set} \]
vector $\nu^t$ as

$$
\nu^t = \max(\tanh(\gamma^t+1 - Q1_{\gamma^t+1}), 0).
$$

(9)

The hyperbolic tangent function (tanh) squashes its negative inputs into the range $[-1, 0]$, and its positive inputs into the range $[0, 1]$. Since the attackers are expected to have values lower than $Q1_{\gamma^t+1}$, tanh will make their trust scores in $\nu^t$ less than 0. The maximum in Expression (9) sets the attackers' trust values to 0. Note that, since the reputations of honest participants are likely to increase in every training round, their trust scores will converge to 1 as training evolves, due to the use of tanh. Finally, the server uses the values in the computed trust vector $\nu^t$ to reweight and aggregate the mixed updates (line 18 of Protocol 1). Since the Attackers’ trust values are 0, their updates are neutralized in the aggregation.

VI. PRIVACY ANALYSIS

In this section, we theoretically demonstrate the effectiveness of FFL against privacy attacks.

A. Privacy Between Participants

Proposition 1: Two participants $j$ and $k$ exchanging fragments in Protocol EXCHANGE_FRAGMENTS do not learn each other’s fragments.

Proof: Since the protocol is symmetric, we only need to prove that participant $k$ does not learn participant $j$’s fragments. Participant $k$ receives the following from participant $j$: $g^b \bmod p$, Enc$_{pk_A}(s_{r_j})$, $W_j \oplus r_j \oplus \rho_j$, and $(W_j \ominus m) \oplus \rho_j$. Then, participant $k$ can compute the common mask $m$, but she cannot decrypt $s_{r_j}$, which would allow her to recreate $r_j$. However, participant $k$ can add $W_j \oplus r_j \oplus \rho_j$ and $(W_j \ominus m) \oplus \rho_j$, which, combined to $k$’s knowledge of $m$, allows $k$ to learn the bits of $r_j$ that encrypt parameters for which bits in the mask are 0. However, in the mixed update $(W_{k})_{\text{mix}}$ computed by participant $k$ in Expression (2), all parameters from participant $j$ corresponding to mask positions equal to 0 are cleared to 0. Hence, the bits of $r_j$ discovered by $k$ do not allow her to retrieve any parameter of $j$.

Note that a third-party intruder observing the exchange of fragments cannot do better than any of the two participants at learning the other participant’s parameters. In fact, the intruder is likely to be in a worse position, because he does not know $m$ unless it is leaked by one of the participants.

B. Unlinking Participants From Their Updates

Proposition 2: Given a mixed update $(W_k)_{\text{mix}}$ obtained by a participant $k$ after exchanging fragments, the probability that a certain subset of $u$ parameters in $W_k$ is entirely present in participant $k$’s mixed update $(W_{k})_{\text{mix}}$ is $(1/2)^u$.

Proof: By construction of Protocol EXCHANGE_FRAGMENTS, in the mixed update $(W_k)_{\text{mix}}$ the original parameters of participant $k$ are found only where the mask $m$ has bits with value 0. Now, the probability a specific set of $u$ positions in $m$ being 0 is $(1/2)^u$ if the generator PRNG used to obtain $m$ is good.

A consequence of Proposition 2 is that mixing updates effectively unlinks them from their originators: indeed, the probability that a specific set of parameters in the original update survives in the mixed update decreases exponentially with the set size. The effectiveness of unlinking updates against privacy attacks is examined in the next sections.

C. Robustness Against Membership Inference Attacks

MIAs [12], [13] leverage a participant’s local update $W_k$ to infer if a specific data point $(x, y)$ was part of her training data. In [12], a semi-honest server exploits the distinguishable pattern that $(x, y)$ leaves on $W_k$. To carry out the attack, the server trains a binary classifier using some available data containing member and nonmember data points and the components of a model trained on the member data points. The binary classifier predicts a membership score for any target data point $(x, y)$. The membership score is the probability that $(x, y)$ belongs to a target participant’s training data. The attack components include: the calculated gradient vector on the data point, the activations of the intermediate layers of the participant’s model, the activation of the output layer, and a scalar representing the loss of the model on the data point. The authors demonstrate that the gradient vector on the target data point is the most important component for the success of the attack. According to the way a semi-honest server performs MIAs, we can state the following proposition.

Proposition 3: Given a mixed update $(W_k)_{\text{mix}}$ of a participant $k$, it is not possible to correctly predict the membership score of a target data point $(x, y)$ belonging to $k$ using $(W_{k})_{\text{mix}}$.

Proof: The binary classifier needs the learned pattern (that is, the correct attack components) to correctly predict the membership score of $(x, y)$. However, in FFL, the semi-honest server will compute an intermediate layer activation $f_{k,l}(x) = \sigma_l((w_{k,l})_{\text{mix}} \cdot x + (b_{k,l})_{\text{mix}})$ instead of $f_{k,l}(x) = \sigma_l(w_{k,l} \cdot x + b_{k,l})$. This will result in random activations and a random loss scalar as well. Based on that, calculating the gradient vector by backpropagating from a wrong loss scalar through the parameters of a mixed model will result in a completely random attack component, and, hence, in a random membership score prediction for the data point $(x, y)$. Therefore, FFL prevents the semi-honest server from correctly predicting the membership score of any target data point.

D. Robustness Against Property Inference Attacks

Property inference attacks [13], [50] try to infer specific properties about the training data by recognizing patterns within a participant’s local model. Melis et al. [13] showed how to infer properties of a participant’s training data that are uncorrelated to the main task features. The idea of the attack is to use a participant’s update to infer properties that characterize a subset of her training data. A semi-honest server can use some auxiliary data, with and without the property, to generate updates with and without that property. Then, it uses the gradients of the generated updates as input features to train a binary classifier. The binary classifier is used to distinguish if an input gradient was computed on data with the same target property. Finally, when the server receives a
target participant’s update $W_k$, it extracts its gradient as
\[
\nabla W_k = \frac{W - W_k}{\eta}.
\]

Then the server passes $\nabla W_k$ to the binary classifier to determine if the participant’s data have the target property.

We can notice that like the classifier in the MIA attack, the classifier in this attack mainly depends on the original gradient vectors. FFL prevents the server from obtaining the whole original gradient of the target participant. In general, her mixed gradient is inconsistent with the pattern the binary classifier was trained on. Thus, the classifier decision will most likely be inaccurate.

E. Robustness Against Reconstruction Attacks

Reconstruction attacks [14], [15], [16] are much stronger than the previous ones, since they can extract both the original training inputs and the labels from a participant’s local gradient. The idea behind these attacks is that a participant’s update $W_k$ is computed based on both the global model $W$ and the participant’s training data $(x_k, y_k)$. Since a semi-honest server has both $W$ and $W_k$, it can obtain the training data by inverting the update gradient. First, the server computes participant $k$’s gradient $\nabla W_k$ using Expression (10). After that, it tries to invert the gradient and to find the unknown training data $(x_k, y_k)$ that results in the same extracted gradient: it starts by randomly initializing a dummy input $x^*$ and a label input $y^*$ and feeds these “dummy data” to the global model $W$ to get “dummy gradients” $\nabla W^*$ as
\[
\nabla W^* = \nabla \mathcal{L}(W, (x^*, y^*)).
\]

Then, the server repeatedly modifies the dummy data in an adversarial perturbation way, based on the difference between the dummy gradient $\nabla W^*$ and participant $k$’s original gradient $\nabla W_k$. A small distance between $\nabla W^*$ and $\nabla W_k$ means that the dummy data are similar to the original data. Zhu and Han [14] and Zhao et al. [15] used the Euclidean distance between $\nabla W_k$ and $\nabla W^*$ as the objective function to modify the dummy data, whereas [16] employ the cosine similarity. The objective function used in [16] is given by
\[
\arg\min_{x^*, y^*} \left( 1 - \frac{\langle \nabla W_k, \nabla W^* \rangle}{\| \nabla W_k \| \| \nabla W^* \|} \right). \tag{12}
\]

Based on the above we can state the following proposition.

Proposition 4: A mixed gradient $\langle \nabla W_k \rangle_{\text{mixed}}$, sent by a participant $k$, cannot be leveraged by the server to estimate a target data point $(x, y)$ in $k$’s local data.

Proof: When using a mixed gradient $\langle \nabla W_k \rangle_{\text{mixed}}$ instead of the original $\nabla W_k$, the objective functions used in the reconstruction attacks will in general result in a random reconstructed data point because in general there is no original data point corresponding to that mixed gradient.

Proposition 4 guarantees that by providing the server with mixed updates instead of the original ones, FFL effectively prevents the semi-honest server from performing reconstruction attacks.

VII. Security Analysis

When exchanging fragments with another honest participant $k$, an attacker $j'$ can follow one of three strategies.

1) Strategy 1: Exchange her poisoned fragment with $k$ and send a poisoned mixed update to the server containing her other poisoned fragment and the good fragment of $k$.

With an expected number of $n/5$ attackers in a training round, there is a chance of poisoning at most $2n' = 2n/5$ mixed updates and at most $n' = n/5$ coordinates.

2) Strategy 2: Exchange her poisoned fragment with $k$ and send a fully poisoned mixed update to the server. Thus, there is a chance of poisoning at most $2n' = 2n/5$ mixed updates and at most $2n' = 2n/5$ coordinates.

3) Strategy 3: Exchange her poisoned fragments with the other participants and submit fully good updates to the server. The attacker does this to increase his global reputation and discredit the honest participants in front of the server. This strategy poisons fewer updates than Strategies 1 or 2: at most $n' = n/5$ mixed updates and at most $n' = n/5$ coordinates.

Now, let us see how FFL can counter the above strategies and neutralize the impact of poisoned mixed updates on the global model aggregation. In Strategy 1, since the number of untouched good coordinates is a majority (4$n/5$), the poisoned mixed updates will have less similarity to the centroid of the mixed update. This lower similarity will decrease the global reputations of both attackers and some honest participants, and the local reputations of the attackers. But since an honest participant is more likely to select another honest participant for the exchange, honest participants will find an opportunity to increase their global reputations and offset the harm caused by the attackers’ poisoned fragments. That is because the probability of an honest participant selecting another honest participant is $(n-n')/(n-1) = (4n-5)/(5n-5)$, whereas the probability of selecting an attacker is $(n')/(n-1) = n/(5n-5)$. Moreover, as the training evolves, the attackers will obtain smaller and smaller local reputations (less than the first quartile in the local reputation vectors). As a result, honest participants will not accept exchanging fragments with them. This will force attackers to send their poisoned updates directly to the server or keep them. If they send them, their global reputations will decrease more and more to be below the first quartile $Q_{1,\gamma}$, which will completely neutralize their influence on the global model because they will not be selected for future training. On the other hand, if they refrain from sending their poisoned updates, they will neutralize themselves. Note that, even if some attackers managed to have global reputations slightly greater than the first quartile in the early training rounds, they would have less influence on the global model aggregation than honest participants, because they will have small trust scores. Fig. 2 shows an example of how the average trusts of honest participants and attackers evolve as the training evolves when the attackers follow Strategy 1. In Strategy 2, the situation does not differ much. The attackers will get still lower similarity values than under Strategy 1, because they send fully poisoned updates, which will be farther from the centroid than the poisoned mixed updates sent by the honest participants. This will be reflected in their global and local
reputations, which will lead to attackers getting neutralized. As for Strategy 3, the attackers will get good global reputations because they always send good mixed updates; however, they will get low local reputations with honest participants. This will deter honest participants from making future exchanges with the attackers. Hence, the attackers will end up being unable to poison the mixed updates sent by honest participants.

VIII. EXPERIMENTAL ANALYSIS

In this section, we report empirical results on three real data sets for the most relevant security and privacy attacks discussed above. We used the PyTorch framework to implement the experiments on a computer with an AMD Ryzen 5 3600 6-core CPU, 32 GB RAM, an NVIDIA GTX 1660 GPU with 6 GB RAM, and Windows 10 OS. As said above, with PyTorch, the parameter bitlength is $\lambda = 32$. Our code and data are available for reproducibility purposes.\(^1\)

In all the experiments, the Acceptor, respectively, the Initiator, looped through all the layers of her update and generated a random binary mask $\mathbf{m}_l$ for each layer $l \in [1, L]$, where $L$ is the total number of layers in the global model $W$. At the end, the Acceptor, resp. the Initiator, set $\text{mask} = \mathbf{mask}_1 \ldots || \mathbf{mask}_L$, where $||$ is the concatenation operator, and used mask to exchange a random fragment.

We used the Diffie–Hellman key exchange protocol with the secure 2048-bit MODP group [59] to generate the secret shared seeds of the fragments’ masks, and we used [60] to encrypt and decrypt the OTP seeds with the recommended 3072-bit key size. We used a value of $\alpha = 0.2$ because we found it gives better simultaneous detection of untargeted and targeted attacks. That is because targeted attacks are stealthier than untargeted ones.

A. Data Sets and Models

We tested the proposed method on three ML tasks: 1) tabular data classification; 2) image classification; and 3) sentiment analysis. Table II summarizes the data sets and models we used. To evaluate the effectiveness of FFL at defeating reconstruction attacks, we used the code provided by the authors of [16], who perform reconstruction attacks with the ConvNet64 model described in their paper (with about 3 million parameters) and some images from the CIFAR10 validation set.

1) Tabular Data Classification: We used the Adult tabular data set\(^2\) that contains 48,842 records of census income information with 14 numerical and categorical attributes. The class label is the attribute income that classifies records into either $>50$K or $\leq 50$K. We used 80% of the data as training data and the remaining 20% as validation data. We randomly and uniformly split the 80% training examples among 20 FL participants. We used a multilayer perceptron (MLP) with one input layer, one hidden layer, and one output layer that contains about 5K learnable parameters. The output layer is followed by a Sigmoid function to produce the final predicted class for an input record. The MLP was trained during 100 rounds. In each round, the FL server selected 10 participants and asked them to train the model for 1 local epoch and a local batch size 64. The participants used the binary cross-entropy loss function with logit loss function and the Adam optimizer with a learning rate $= 0.001$ to train their models.

2) Image Classification: We used two data sets for this task.

1) MNIST data set. It contains 70K handwritten digit images from 0 to 9 (i.e., 10 classes) [61]. The images are in grayscale with size $28 \times 28$ pixels, and they are divided into a training set (60K examples) and a testing set (10K examples). We randomly and uniformly split the 60K training examples among 100 simulated participants of an FL setting. We used a two-layer convolutional neural network (CNN) with two fully connected layers. The CNN model was trained during 200 rounds. In each round, the FL server randomly chose 50 participants and asked them to train the model for 3 local epochs and a local batch size 64. The participants used the cross-entropy loss function and the stochastic gradient descent (SGD) optimizer with a learning rate $= 0.001$ and momentum $= 0.9$ to train their models.

2) CIFAR10 data set. It consists of 60K colored images of 10 different classes [62]. The data set is divided into 50K training examples and 10K testing examples. We randomly and uniformly split the 50K training examples among 20 FL participants. We used the VGG16 CNN model with one fully connected layer [63]. The VGG16 model was trained during 100 rounds. In each round, the FL server randomly chose 10 participants and asked them to train the model for 3 local epochs and a local batch size 32. The participants used the cross-entropy loss function and the SGD optimizer with a learning rate $= 0.01$ and momentum $= 0.9$ to train their models.

3) Sentiment Analysis: We used the IMDB large movie review data set [64] for this binary sentiment classification task. This data set is a collection of 50K movie reviews and

---

**TABLE II**

| Data Sets and Models Used in the Experiments |
|---------------------------------------------|
| **Task** | **Data set** | **# Examples** | **Model** | **# Parameters** |
|----------|--------------|----------------|-----------|-----------------|
| Tabular classif. | Adult | 48,842 | MLP | $\sim 5K$ |
| Image classif. | CIFAR10 | 60K | VGG16 | $\sim 15M$ |
| Sent. analysis | IMDB | 50K | BiLSTM | $\sim 13M$ |

---

\(^1\)https://github.com/anonymized30/FFL

\(^2\)https://archive.ics.uci.edu/ml/datasets/adult

---

Fig. 2. Participant average trust evolution during training in the CIFAR10-VGG16 benchmark.
We divided the data set into their corresponding sentiment binary labels (either positive or negative). We used the bidirectional long/short-term memory (BiLSTM) model, which has an embedding layer of 100 dimensions for each token. The model ends with a linear layer followed by a Sigmoid function to produce the final predicted sentiment for an input review. The BiLSTM was trained during each round, the FL server randomly chose 100 training examples among K training examples for each participant. We randomly and uniformly split the 40K training examples among 20 FL participants. We used a standard FL system with systematic aggregates all received updates. We evaluated TE and All-Acc under the Gaussian noise untargeted attack, and TE, Src-Acc, and ASR under the label-flipping targeted attack. We used standard FL with FedAvg when no attacks were performed as a baseline to show the impact of attacks on the model performance. In our experiments, the percentage of attackers was 20% for all four benchmarks.

1) Gaussian Noise Attack: The attackers added Gaussian noise to their updates to prevent the model from converging [19], [57]. Specifically, they added noise with 0 mean and 0.5 standard deviation for the MLP and CNN model parameters, and 0.2 standard deviation for VGG16 and BiLSTM parameters.

Table III shows the results under this attack. First, we can see the significant negative impact of the attack on the performance of FedAvg regarding both the TE and the All-Acc. The case was even worse with FoolsGold because the added noise made the attackers’ last layers more diverse than the honest participants’ FoolsGold assumes participants with similar last layers to be attackers and those with diverse last layers to be honest. Thus, it considered poisoned updates and excluded good updates in the model aggregation. The rest of the methods, including FFL, achieved comparable results to the baseline. Since the added noise made the magnitudes of the poisoned updates different from those of good updates, we observe that: 1) the median and the trimmed mean were able to neutralize the poisoned parameters in model aggregation; 2) multi-Krum was able to exclude the poisoned updates due to the larger Euclidean distances they had; and 3) FFL was able to exclude poisoned mixed updates because their deviations from their medians were larger than those of good mixed updates. We can also see that, in most cases (Adult-MLP, MNIST-CNN, and CIFAR10-VGG16), FFL achieved the lowest TE among all methods. As the training evolved, FFL set the trust values of honest participants to 1 and thus fully considered their contributions. Note that FFL achieved similar performance under attack strategies 1 and 2.

2) Label-Flipping Attack: In the label-flipping attack, attackers flip the labels of correct training examples from one class (also known as the source class) to another class and train their models according to the latter [20], [29]. For Adult, the attackers flipped each example with the label “≥50K” to “≤50K,” while they flipped each example with the label “7” to “1” for MNIST. For CIFAR10, the attackers flipped each example with the label “Cat” to “Dog.” For IMBD, they flipped the “positive” reviews to “negative.”

Table IV shows the results under the label-flipping attack. For the Adult-MLP benchmark, the FedAvg performance

| Benchmark/Method | Adult-MLP | MNIST-CNN | CIFAR10-VGG16 | IMDB-BiLSTM |
|------------------|-----------|-----------|---------------|-------------|
|                  | TE | All-Acc | TE | All-Acc | TE | All-Acc | TE | All-Acc |
| FedAvg (no attacks) | 0.349 | 82.56 | 0.112 | 96.69 | 0.881 | 80.77 | 0.475 | 88.63 |
| FedAvg | 1.138 | 75.08 | 2.322 | 9.65 | 10.334 | 10.40 | 0.618 | 67.7 |
| Median | 0.339 | 82.87 | 0.115 | 96.65 | 1.020 | 78.91 | 0.325 | 88.33 |
| Trimmed mean | 0.330 | 82.61 | 0.114 | 96.64 | 1.046 | 80.23 | 0.457 | 88.79 |
| Multi-Krum | 0.330 | 82.69 | 0.126 | 96.18 | 0.998 | 78.86 | 0.365 | 87.72 |
| FoolsGold | 44.053 | 69.47 | 2.541 | 8.92 | 12.813 | 9.02 | 2.694 | 51.61 |
| Strategy 1 | 0.349 | 82.84 | 0.113 | 96.67 | 0.934 | 79.74 | 0.336 | 88.15 |
| Strategy 2 | 0.330 | 82.86 | 0.113 | 96.61 | 0.938 | 79.26 | 0.342 | 87.48 |
under the attack significantly degraded for Src-Acc and ASR. However, TE slightly increased and kept close to that of the baseline (FedAvg - no attacks). The reason for that is, the adult data set is highly imbalanced, with a skew toward the "≤50K" class label. The median and the trimmed mean achieved low TE, but saw degraded performance for Src-Acc and ASR because they discarded a large number of coordinates in the global model aggregation. Multi-Krum, FoolsGold, and FFL achieved similar results to the baseline with slightly greater Src-ACC and slightly lower ASR. Since the Adult data set’s training data were randomly and uniformly distributed among the participants, some of them had larger percentages of the target class examples compared to the original class distribution of the data set. The original percentage of the examples belonging to the target class ≤50K in the data set is about 75.22%, which caused bias in the global model against the minority class >50K. This made the updates of those honest participants having higher percentages of the target class close to the attackers’ updates. Since multi-Krum, FoolsGold, and FFL excluded or penalized some honest participants with updates close to the attackers’ updates, the global model became less biased against the minority class >50K and, hence, the Src-Acc slightly increased. For MNIST-CNN, the performance of FedAvg, median, and trimmed mean slightly degraded compared to the baseline. An interesting note is that FedAvg was not highly affected by the attack. The reasons for that are the small size of the model and the simple and balanced distribution of the data set. That was also observed in [65], where the authors argued that, in some cases, FedAvg could be robust against poisoning attacks. For CIFAR-VGG16, Src-Acc degraded from 70.73% to 53.82%, and ASR increased from 14.51% to 29.34% with FedAvg. On the other hand, FedAvg achieved TE lower than that of the median, the trimmed median, and FoolsGold. That is because the attackers flipped the labels in the examples for only one class and kept the labels of the other classes unchanged. The median and the trimmed mean decreased Src-Acc and increased ASR because of the large size of the VGG16 model. Chang et al. [40] have shown that the estimation errors of the median and the trimmed mean scale up with the size of the model in a square-root manner. Multi-Krum achieved the worst performance regarding Src-Acc and ASR because the small impact of the attack was not detectable in the large model. Therefore, multi-Krum identified some attackers as honest while identifying some honest participants as attackers, which led to its poor performance. FFL achieved the best performance among all the methods for all three metrics. FoolsGold scored the second-best after FFL regarding Src-Acc and ASR. FFL and FoolsGold achieved such good performance because they analyzed the last-layer gradients, which contain more useful information for detecting the behavior of targeted poisoning attacks. However, FoolsGold achieved the greatest TE among the other methods because it did not consider the full contributions of the honest participants. FFL, however, considered almost all the full contributions of the honest participants and thus achieved the lowest TE.

For the IMDB-BiLSTM benchmark, FFL and FoolsGold achieved the best performance among all methods, most of which were negatively impacted by the large size of the BiLSTM model. FFL and FoolsGold outperformed the other methods by a large margin in achieving good values for all metrics. FoolsGold performed well in this benchmark because it was its ideal setting: updates from honest participants were somewhat different due to the different reviews they gave, whereas updates for attackers became very close to each other because they shared the same objective.

To summarize, the results show that FFL can effectively defend against untargeted and targeted poisoning attacks while preserving the benign model performance. Moreover, FFL outperforms the state-of-the-art defenses in achieving good model performance while preventing the attackers from mounting successful security attacks and hindering the semi-honest server from mounting privacy attacks.

C. Protection Against the Reconstruction Privacy Attack

We next report results on the protection offered by FFL against the most powerful privacy attack, namely the reconstruction attack proposed in [16]. Notice that this attack does not require auxiliary data and can estimate the private training data by inverting their corresponding gradients. Fig. 3 shows the results when two participants, k and j, sent their updates computed on just their private images (left column of the figure) to the server. In the FL setting (middle column of the figure), the server was able to reconstruct their private images with high accuracy. However, when they mixed fragments of their updates before sending them (FFL setting, right column of the figure), the server was only able to obtain noise instead of the original images.

Similarly, Fig. 4 shows the results when two participants k and j sent their updates computed on a batch of 8 images to the server. The figure exemplifies on two different batches of input images; the first example is given in the three upper rows and the second one in the three lower rows. In the FL setting, the server was able to recover a lot of information about the participants’ training data. However, when they mixed their
Table V reports the CPU runtimes in seconds per training round for mixed update decryption, global reputation calculation, and model aggregation on the server side, and local model computation, fragment exchanging, and mixing on the participants’ side. Note that the reported runtimes for exchanging and mixing fragments are the average for a single participant. We computed the runtime for the three benchmarks with 10, 50, and 100 updates per training round to illustrate how the runtime scales with the number of updates. On the server side, we report the total runtime for all methods; in contrast, on the participant side, we report the overhead (extra runtime) with respect to standard FL.

It can be seen that, as the number of updates and the model size increase, the FFL server’s runtime grows less than for the other methods, which confirms our computation cost analysis of Appendix S.II-A in the supplementary materials. In particular, with CIFAR10-VGG16, FFL achieved the lowest runtime among all nonbaseline methods. Furthermore, unlike FFL, the other methods are unable to thwart privacy attacks or provide adequate protection against security attacks.

Regarding the runtime overhead incurred by each participant, the maximum runtime overhead resulting from exchanging and mixing fragments was about 1.319 s for the largest model we used, which was VGG16. Also, it is worth noting that the increase in the number of updates had little impact on the overhead of the participants because their computations essentially depend on the model dimensionality.
Nevertheless, the FFL runtime is very small if we compare it with that of [36], which provides a level of privacy similar to ours but without being able to neutralize poisoned updates. Specifically, for an LSTM model containing only 4.02 million parameters, [36] report a runtime overhead for each participant of about 176 s and a total runtime for the server of 174 s to aggregate 50 local updates. Note also that the hardware specifications employed by [36] are superior to ours. Therefore, the participants in FFL can turn a blind eye to the computational overhead they incur in exchange for protecting their privacy, countering security attacks, and learning a more accurate global model.

IX. CONCLUSION

In this article, we have presented FFL, a novel approach based on cooperation between participants in FL systems to preserve their privacy without renouncing robust and accurate aggregation of their updates to the global model. FFL offers a practical solution where participants privately and efficiently exchange random fragments of their updates before sending them to the server. We have also proposed a novel reputation-based defense tailored for FFL that builds trust in the participants based on the quality of the mixed updates they send and the fragments they exchange. We have demonstrated that the proposed framework can effectively counter privacy and security attacks. All the above is achieved while obtaining a global model’s accuracy similar to that of standard FL (when no attack is performed) and imposing affordable communication cost and computation overhead on the participating parties. The efficiency of FFL makes it applicable to large-scale FL systems.

As future work, we plan to evaluate the robustness of FFL against backdoor attacks. Also, we plan to test its performance when working with non-identical and independently distributed (non-iid) data.

ACKNOWLEDGMENT

Thanks go to Ashmeet Singh for help with some figures. The authors are with the UNESCO Chair in Data Privacy, but the views in this article are their own and are not necessarily shared by UNESCO.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, “Communication-efficient learning of deep networks from decentralized data,” in Proc. 20th Int. Conf. Artif. Intell. Statist., 2017, pp. 1273–1282.
[2] K. Bonawitz et al., “Towards federated learning at scale: System design,” 2019, arXiv:1902.01096.
[3] H. Yu, Y. Gao, F. Yu, and T. Darrell, “End-to-end learning of driving models from large-scale video datasets,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 2174–2182.
[4] A. Bhownick, J. Duchi, J. Freudiger, G. Kapoor, and R. Rogers, “Protection against reconstruction and its applications in private federated learning,” 2018, arXiv:1812.00984.
[5] T. S. Briximi, R. Chen, T. Mela, A. Olshovsky, I. C. Paschalidis, and W. Shi, “Federated learning of predictive models from federated electronic health records,” Int. J. Med. Inform., vol. 112, pp. 59–67, 2018.
[6] A. Hard et al., “Federated learning for mobile keyboard prediction,” 2018, arXiv:1811.03604.
[7] V. Motthukuri, P. Khare, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, “Federated-learning-based anomaly detection for IoT security attacks,” IEEE Internet Things J., vol. 9, no. 4, pp. 2545–2554, 2022.
[8] P. Voigt and A. Von dem Busche, “The EU general data protection regulation (GDPR),” in A Practical Guide, vol. 10, 1st ed. Cham, Switzerland: Springer, 2017.
[9] P. Kairouz et al., “Advances and open problems in federated learning,” 2019, arXiv:1912.04977.
[10] V. Motthukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, “A survey on security and privacy of federated learning,” Future Gener. Comput. Syst., vol. 115, pp. 619–640, Feb. 2021.
[11] H. Brendan McMahan, D. Ramage, K. Talwar, and L. Zhang, “Learning differentially private recurrent language models,” 2017, arXiv:1710.06963.
[12] M. Nasr, R. Shokri, and A. Houmansadr, “Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning,” in Proc. IEEE Symp. Secur. Privacy (SP), May 2019, pp. 739–753.
[13] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, “Exploiting unintended feature leakage in collaborative learning,” in Proc. IEEE Symp. Secur. Privacy (SP), May 2019, pp. 691–706.
[14] L. Zhu and S. Han, “Deep leakage from gradients,” in Federated Learning, Berlin, Germany: Springer, 2020, pp. 17–31.
[15] B. Zhao, K. Reddy Mopuri, and H. Bilen, “IDLG: Improved deep leakage from gradients,” 2020, arXiv:2001.02610.
[16] J. Geising, H. Bauermeister, H. Dröge, and M. Moeller, “Inverting gradients—How easy is it to break privacy in federated learning?” 2020, arXiv:2003.14053.
[17] A. Blanco-Justicia, J. Domingo-Ferrer, S. Martínez, D. Sánchez, A. Flanagan, and K. E. Tan, “Achieving security and privacy in federated learning systems: Survey, research challenges and future directions,” Eng. Appl. Artif. Intell., vol. 106, Nov. 2021, Art. no. 104468.
[18] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, “Machine learning with adversaries: Byzantine tolerant gradient descent,” in Proc. 31st Int. Conf. Neurol. Inf. Process. Syst., 2017, pp. 118–128.
[19] Z. Wu, Q. Ling, T. Chen, and G. B. Giannakis, “Federated variance-reduced stochastic gradient descent with robustness to byzantine attacks,” IEEE Trans. Signal Process., vol. 68, pp. 4583–4596, 2020.
[20] B. Biggio, B. Nelson, and P. Laskov, “Poisoning attacks against support vector machines,” 2012, arXiv:1206.6389.
[21] H. Fu, C. J. Yoon, and L. Beschastnikh, “The limitations of federated learning in sybil settings,” in Proc. 23rd Int. Symp. Res. Attacks, Intrusions Defenses (RAID), 2020, pp. 301–316.
[22] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” in Proc. Int. Conf. Artif. Intell. Statist., 2020, pp. 2938–2948.
[23] C. Dwork and A. Roth, “The algorithmic foundations of differential privacy,” Found. Trends Theor. Comput. Sci., vol. 9, nos. 3–4, pp. 211–407, 2013.
[24] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in Proc. 41st Ann. ACM Symp. Symp. Theory Comput. (STOC), 2009, pp. 169–178.
[25] A. C. Yao, “Protocols for secure computations,” in Proc. 23rd Ann. Conf. Found. Comput. Sci. (FOCS), Nov. 1982, pp. 160–164.
[26] J. Domingo-Ferrer, D. Sánchez, and A. Blanco-Justicia, “The limits of differential privacy (and its misuse in data release and machine learning),” Commun. ACM, vol. 64, no. 7, pp. 33–35, 2021.
[27] A. Blanco-Justicia, D. Sánchez, J. Domingo-Ferrer, and K. Muralidhar, “A critical review on the use (and Misuse) of differential privacy in machine learning,” ACM Comput. Surv., to be published.
[28] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, “Byzantine-robust distributed learning: Towards optimal statistical rates,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 5650–5659.
[29] C. Fung, C. J. M. Yoon, and L. Beschastnikh, “Mitigating sybils in federated learning poisoning,” 2018, arXiv:1808.04866.
[30] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions,” IEEE Signal Process. Mag., vol. 37, no. 3, pp. 50–60, May 2020.
[31] Y. Liu, B. Guo, X. Zou, Y. Li, and S. Shi, “Machine learning assisted materials design and discovery for rechargeable batteries,” Energy Storage Mater., vol. 31, pp. 434–450, Oct. 2020.
[32] K. Bonawitz et al., “Practical secure aggregation for privacy-preserving machine learning,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2017, pp. 1175–1191.
[33] J. So, B. Guler, and A. S. Avestimehr, “Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning,” IEEE J. Sel. Areas Inf. Theory, vol. 2, no. 1, pp. 479–483, Mar. 2021.
[34] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, “Privacy-preserving deep learning via additively homomorphic encryption,” IEEE Trans. Inf. Forensics Security, vol. 13, no. 5, pp. 1333–1345, May 2018.
A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, “Analyzing BatchCrypt: Efficient homomorphic encryption for cross-silo federated learning,” in Proc. USENIX Annu. Tech. Conf., 2020, pp. 493–506.

R. C. Geyer, T. Klein, and M. Nabi, “Differentially private federated learning: A client level perspective,” 2017, arXiv:1712.07557.

E. Mahdi El Mhamdi, R. Guerraoui, and S. Rouault, “The hidden vulnerability of distributed learning in byzantium,” 2018, arXiv:1802.07927.

L. Chen, H. Wang, Z. Charles, and D. Papaioliopoulos, Draco: Byzantine-resilient distributed training via redundant gradients, in Proc. 35th Int. Conf. Mach. Learn., 2018, pp. 903–912.

H. Chang, V. Shejwalkar, R. Shokri, and A. Hounamsadr, Crounus: Robust and heterogeneous collaborative learning with black-box knowledge transfer, 2019, arXiv:1912.11279.

M. Naseri, J. Hayes, and E. De Cristofaro, “Local and central differential privacy for robustness and privacy in federated learning,” 2020, arXiv:2009.03561.

X. Liu, H. Li, G. Xu, Z. Chen, X. Huang, and R. Lu, “Privacy-enhanced federated learning against poisoning adversaries,”IEEE Trans. Inf. Forensics Security, vol. 16, pp. 4571–4588, 2021.

Z. Ma, J. Ma, Y. Miao, Y. Li, and R. Deng, “ShieldFL: Mitigating model poisoning attacks in privacy-preserving federated learning,” IEEE Trans. Inf. Forensics Security, vol. 17, pp. 1639–1654, 2022.

J. So, B. Güler, and A. S. Avetisian, “Byzantine-secure federated learning,”IEEE J. Sel. Areas Commun., vol. 39, no. 7, pp. 2168–2181, Jul. 2020.

X. Ma, X. Sun, Y. Wu, Z. Liu, X. Chen, and C. Dong, “Differentially private byzantine-robust federated learning,”IEEE Trans. Parallel Distrib. Syst., vol. 33, no. 12, pp. 3690–3701, Dec. 2022.

Z. Zhang, J. Li, S. Yu, and C. Makaya, “SAFELearning: Enable backdoor detectability in federated training with secure aggregation,” 2021, arXiv:2102.02402.

J. Domingo-Ferrer, A. Blanco-Justicia, J. Manjon, and D. Sanchez, “Secure and privacy-preserving federated learning via co-utility,”IEEE Internet Things J., vol. 9, no. 5, pp. 3988–4000, Mar. 2022.

Y. Chen, F. Luo, T. Li, T. Xiang, Z. Liu, and J. Li, “A training-integrity privacy-preserving federated learning scheme with trusted execution environment,”Inf. Sci., vol. 522, pp. 69–79, Jun. 2020.

Y. Zhang, Z. Wang, J. Cao, R. Hou, and D. Meng, “ShuffleFL: Gradient-preserving federated learning using trusted execution environment,” in Proc. 18th ACM Int. Conf. Comput. Frontiers, 2021, pp. 161–168.

K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, “Property inference attacks on fully connected neural networks using permutation invariant representations,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2018, pp. 619–633.

B. Hitaj, G. Atieniese, and F. Perez-Cruz, “Deep models under the GAN: Information leakage from collaborative deep learning,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., 2017, pp. 603–618.

M. Fang, X. Cao, J. Jia, and N. Gong, “Local model poisoning attacks to Byzantine-robust federated learning,” in Proc. 29th USENIX Secur. Symp. (USENIX Secur.), 2020, pp. 1605–1622.

M. Jagielko, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, “Manipulating machine learning: Poisoning attacks and countermeasures for regression learning,” in Proc. IEEE Symp. Secur. Privacy (SP), May 2018, pp. 19–35.

A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, “Analyzing federated learning through an adversarial lens,” in Proc. 36th Int. Conf. Mach. Learn., 2019, pp. 634–643.

A. Shafahi et al., “Poison frogs! Targeted clean-label poisoning attacks on neural networks,” 2018, arXiv:1804.00792.

C. Davenport and C. Davenport. (Aug. 2018). Gboard Passes One Billion Installs on the Play Store. accessed: 2022-Apr. 2, 2022. [Online]. Available: https://www.androidpolice.com/2018/08/22/gboard-passes-one-billion-installs-play-store/