SAAFL: Secure Aggregation for Label-Aware Federated Learning

Aftab Akram[1][0009-0003-2402-4058], Harry N. H. Pham[1], Melek Önen[1][0000-0003-0269-9495], Clémentine Gritti[2][0000-0002-0835-8678]

[1] Department of Digital Security, EURECOM, 06410 Biot, France
{aftab.akram, huan.pham, melek.onen}@eurecom.fr
[2] CITI Lab, INSA Lyon – Inria, 69100 Villeurbanne, France
clementine.gritti@insa-lyon.fr

Abstract. Secure aggregation (SA) has emerged as a vital component of federated learning (FL), enabling collaborative training of a global machine learning model while safeguarding the privacy of clients' local datasets. Most existing SA protocols implement the privacy preserving-variant of federated averaging (FedAvg) as the aggregation technique and assume independent and identically distributed (IID) datasets across clients. This assumption makes FedAvg unsuitable for non-IID scenarios, where variations in client datasets lead to less effective global model. We propose SAAFL, a SA protocol specifically designed for non-IID settings and more specifically for the recently proposed federated label-aware aggregation (FedLA) protocol. SAAFL computes the weighted average of clients' inputs where weights depend on the label distributions and should remain confidential. SAAFL is resilient to client dropouts and supports client selection. Our experimental results show that it achieves comparable model accuracy with FedLA and remains efficient in terms of computation and communication.

Keywords: Federated Learning· Secure Aggregation · non-IID data.

1 Introduction

Federated learning (FL) [13] has received significant interest across various stakeholders nowadays. In such a framework, machine learning models are trained with the collaboration of multiple clients: FL clients train models locally with their data and send them to a server, which aggregates them into a global model, ensuring that data samples never leave clients' premises. As for the aggregation operation, Federated Averaging (FedAvg) [13] is widely used in FL which consists of simply computing the average of all clients' inputs. Unfortunately, recent findings have shown that FedAvg falls short in achieving good model accuracy level when datasets are heterogeneous or non-identically distributed (non-IID) [15, 19]. To address this, authors in [9] develop a Federated Label-Aware Aggregation (FedLA) protocol which assigns label-specific weights to improve the

accuracy of non-IID data aggregation. On the other hand, the literature has shown that sharing FL clients' inputs in plaintext can leak information about the training data [14, 17] and there has a been a plethora of secure aggregation (SA) solutions [12] to thwart such attacks. Indeed, SA consists of computing an aggregated value without discovering individual inputs. While existing SA methods seem very promising when the aggregation operation consists of FedAvg, these methods, unfortunately, cannot be directly employed for FedLA. Relying on FedLA with non-IID data makes the SA process much more complex since the aggregation is not a simple sum anymore, and the assigned weights should remain confidential, too. Client dropouts also pose a significant challenge in the context of FedLA. While with FedAvg, dropouts were already addressed by reconstructing an encrypted zero-value, such a solution unfortunately falls short in the context of FedLA and may drop the accuracy of the model significantly.

In this paper, we propose a new FL protocol, named SAAFL[3], that is customized for FedLA. This solution ensures, through a new SA building block, a privacy-preserving clients' weighted model update aggregation along with secure clients' weight computations. Moreover, at each training round, SAAFL enables online clients to reconstruct dropped clients' contributions as specific weighted updates based on the previous round's model parameters.

Section 2 identifies the challenges for the design of SA customized for FedLA. In Section 3, the main SA building block is described. Section 4 describes the entire SAAFL protocol which is further studied in terms of performance in Section 5.

2 Towards privacy-preserving FedLA

In this section, we remind the FedLA algorithm and further identify the main challenges to design its privacy-preserving variant. We finally introduce our solution in high-level, explaining how each of these challenges are addressed.

2.1 FedLA Algorithm

As described in [9], FedLA is a FL framework designed to address non-IID data distributions. Compared to the standard FedAvg [13] method whereby the average of FL clients' model updates is computed, in FedLA the server computes a weighted average of these updates, whereby weights translate the distribution of classification labels among FL clients' datasets. Indeed, each client is assigned a so-called FedLA-weight that depends on the distribution of labels among data samples. More specifically, with n clients participating in the training of the FL model, and k labels represented in the entire dataset, let s_u be the number of samples held by client u, s_l be the total number of samples with label l across all clients, and $s_{u,l}$ be the number of samples with label l held by client u. To compute the overall FedLA-weight of each client, the process begins by determining the proportion of data that u owns with label l defined as: $w_{u,l} = \frac{s_{u,l}}{s_l}$. These

[3] SAAFL means simple in Urdu

terms are further summed up across all labels to compute: $w_u = \sum_{l=1}^{k} w_{u,l}$. Finally, the FedLA-weight for each client u is determined as:

$$w_{FedLA,u} = \frac{w_u}{\sum_{u=1}^{n} w_u} \tag{1}$$

This FedLA-weight serves as the global model's averaging coefficient.[4]

2.2 Challenges when integrating secure aggregation in FedLA

Similar to traditional FL, FedLA is vulnerable to membership inference attacks (MIA) if gradients are sent in the clear and, unfortunately, integrating SA into FedLA is not straightforward and raises new challenges. We describe them below after introducing the aligned threat model as in [11].

Threat model: An untrusted FL server may collude with some clients, along with the possibility that honest clients unintentionally fail or drop out. These failures can occur at any stage, and the adversary (comprising the server and colluding clients) aims to extract private information about honest clients' individual inputs. Two adversarial settings are considered: in the *passive model*, the adversary follows the protocol correctly but tries to infer private information from the protocol transcript, while in the *active model*, the adversary manipulates messages to uncover private data. We consider attacks that alter the aggregated results or perform denial-of-service (DoS) attacks out of scope. Additionally, attacks, where adversaries impersonate clients, are not considered either, since these can be easily mitigated through the deployment of public key infrastructures (PKI).

FedLA-weight privacy (C_1): Unlike the traditional FL approach whereby the aggregator only receives private model updates and executes SA, FedLA firstly requires the private computation of each client's FedLA-weights (which depend on some private inputs from other clients as well). As described in the previous section, these FedLA-weights should remain confidential to the server [21]. As a result, FedLA introduces a dual-layer privacy requirement: one for the private computation of each client's weight based on other clients' inputs and one for the computation of the global model (through a dedicated SA protocol).

Client Dropout in FedLA (C_2): Another challenge that arises with secure FedLA as well as in the standard FL framework is client dropouts. In many real-world scenarios, clients may be intermittently unavailable due to network issues, device power limitations, or other reasons. When clients drop out, their model updates are not received, which can disrupt the training of the model. There already exist many solutions [2, 3, 8, 10, 11, 18] that are resilient to client dropouts in privacy-preserving variants of FL frameworks. In these schemes, online clients mainly reconstruct the encryption of a zero-value for dropped clients [2, 3, 11, 18]. Unfortunately, these solutions fall short in the case of FedLA

[4] The term $\sum_{u=1}^{n} w_u$ equals k in the dataset: $\sum_{u=1}^{n} w_u = \sum_{u=1}^{n}\sum_{l=1}^{k} w_{u,l} = \sum_{u=1}^{n}\sum_{l=1}^{k} \frac{s_{u,l}}{s_l} = \sum_{l=1}^{k}\sum_{u=1}^{n} \frac{s_{u,l}}{s_l} = \sum_{l=1}^{k} \frac{s_l}{s_l} = k$

because clients' FedLA weights depend on the contributions of all the other clients including the dropped ones. Hence, the reconstruction of the encryption of a zero-value for dropped clients remains insufficient because this simple addition could skew the FedLA aggregation, leading to inaccurate model updates and degraded performance. Therefore, FedLA requires a new, dedicated fault-tolerant SA scheme that can somehow reconstruct proper FedLA weights that would remove the impact of dropped clients.

Client selection in SA for FedLA (C_3): Another challenge in achieving privacy-preserving FedLA through the use of SA consists of addressing client selection at each round. Indeed, in SA, the aggregation key usually depends on all clients' individual keys, and at each FL round, SA should still be correct while non-selected clients' keys are not included. Since, in FedLA, the distribution of clients is assumed to be non-IID and this is reflected through the use of FedLA weights, these weights of non-selected clients also need to be omitted while still achieving SA correctly.

2.3 Our Solution - Overview

In this section, we explain our design goals to address each of the three challenges identified in the previous section are adressed in the newly proposed solution, named as SAAFL, that is further described in details in Section 4.

Privacy-Preserving FedLA-weight Computation (G_1): As previously mentioned, the new solution should incorporate two privacy layers: one to compute each client's FedLA-weight $w_{FedLA,u}$ and one to compute global (aggregated) model parameters during the actual FL round. To compute $w_{FedLA,u}$ in a privacy-preserving manner, we propose that each client u protects the local label distribution $s_{u,l}$ through a standard SA framework and transfers it to the FL server. The server aggregates these received values from all clients to calculate the overall weight per label s_l, which is subsequently returned to each client. Clients can now individually determine their $w_{FedLA,u}$ weight relative to the global label distribution as described in Section 2.1, and this, without having access to other clients label weights $s_{v,l}$ in cleartext. These weights should be computed only once in the *Registration* step and require only one communication round. Once weights are determined, clients can proceed to the SA of the weighted model parameters.

Fault-tolerant FedLA (G_2): Regarding C_2, because the inputs of each client are not independent of each other due to $w_{FedLA,u}$, to address client dropouts, we propose a simple, yet effective, fault-tolerant SA solution based on a variant of the threshold Joye-Libert scheme proposed in [11]. In this new solution named **TJL-A**, each online client sends the difference between their current and previous round weighted inputs.

Once the server receives all inputs, the result will correspond to the aggregation of online clients' weighted parameters together with the previous model parameters multiplied by the remaining weight (corresponding to the dropped clients). Hence dropout clients' inputs become previous model's parameters that are properly weighted. In Section 5, we experimentally demonstrate the performance of this solution.

Handling client selection in SA for FedLA (G_3): To address the impact of non-selected clients, we apply the same technique proposed in G_2 to handle dropout clients. Hence, non-selected clients are considered as dropouts.

In the sequel of the paper, we describe how SAAFL achieves these three design goals starting by describing the new **TJL-A** building block.

3 Building Blocks

In this section, we outline the key building blocks necessary for designing our solution SAAFL, including the newly proposed **TJL-A** aggregation solution.
Secret Sharing Over the Integers (ISS): ISS used to secret share clients' keying material and some contributions for dropped clients.

- $\{(u, [\Delta s]_u)\}_{u\in\mathcal{U}} \leftarrow$ **ISS.Share**$(s, t, \mathcal{U} = \{1, \ldots, n\})$: This algorithm splits secret $s \in [-I, I]$ into n shares $[\Delta s]_u$, assigning each share to a user $u \in \mathcal{U}$, with t being the reconstruction threshold and $\Delta = n!$. The algorithm creates a random polynomial $p(x)$ of degree $t-1$ where $p(0) = \Delta s$ and each share is $[\Delta s]_u = p(u)$.
- $s \leftarrow$ **ISS.Recon**$(\{(u, [\Delta s]_u)\}_{u\in\mathcal{U}'}, t)$: This algorithm reconstructs secret $s \in [-I, I]$ from at least t shares received from users in $\mathcal{U}' \subseteq \mathcal{U}$. Using the Lagrange interpolation, the secret is computed as:

$$s = \frac{\sum_{u\in\mathcal{U}'} \mu_u [\Delta s]_u}{\Delta^2}, \quad \text{such that } \mu_u = \frac{\Delta \cdot \prod_{v\in\mathcal{U}'\setminus\{u\}} v}{\prod_{v\in\mathcal{U}'\setminus\{u\}} (v-u)}.$$

  μ_u represents the Lagrange coefficient for user u.

Label-aware Threshold Joye-Libert Aggregation (TJL-A): We propose a new variant of the **TJL** scheme originally proposed in [11] that helps SAAFL take into account the FedLA weights of dropped clients during aggregation. The main difference between the newly proposed **TJL-A** scheme and **TJL** is that when FL clients drop, their inputs are not replaced with zero-values anymore. Indeed, the original **TJL** reconstructs the zero-value encrypted with the dropped clients' individual keys. As also shown in Figure 3 in Section 5, in the context of a non-IID distribution of FL clients, aggregating a zero-value for dropped clients significantly impacts the accuracy of the actual model. Therefore, we propose to replace this zero-value with a previous round's model parameters weighted with a specific coefficient. This carefully selected value compensates for the missing contributions of the dropped clients more accurately and ensures the correctness of the aggregation at the same time. Furthermore, the initial contribution of each FL client becomes a weighted differences of their model parameters and the previous model parameters: $x_{u,\tau} = w_{FedLA,u}\theta_u^{\tau} - w_{FedLA,u}\theta^{(\tau-1)}$. Therefore, at the aggregation phase, the server will reconstruct a weighted average of online clients' inputs and previous model parameters for dropped clients.

Accordingly, the proposed **TJL-A** scheme consists of the following PPT algorithms:

- $(sk_0, \{sk_u\}_{u\in\mathcal{U}}, N, H, \sigma) \leftarrow$ **TJL-A.Setup**(λ) : Given security parameter λ, this algorithm outputs one secret key per client sk_u, the aggregator key $sk_0 = -\sum_{u\in\mathcal{U}} sk_u$ and the public parameters $pp = (N, H)$ such that $N = pq$ (p and q being large and equal-size prime numbers) and H is a cryptographic hash function $H : \mathbb{Z}_N \to \mathbb{Z}_{N^2}$. Additionally, it sets security parameter σ for **ISS**.
- $\{(v, [\Delta sk_u]_v)\}_{v\in\mathcal{U}} \leftarrow$ **TJL-A.SKShare**$(sk_u, t, \mathcal{U})$: This algorithm invokes **ISS.Share** with $sk_u \in [-2^b, 2^b]$, with b representing the bit-length of modulus N and outputs the output of **ISS**, namely the secret shares of sk_u for each user $v \in \mathcal{U}$.
- $Y_{u,\tau} \leftarrow$ **TJL-A.Protect**$(pp, sk_u, \tau, x_{u,\tau})$: This algorithm encrypts private input $x_{u,\tau}$ as described in equation 2 for round τ using sk_u and outputs ciphertext $Y_{u,\tau}$ using the original JL algorithm [7]:

$$Y_{u,\tau} = (1 + Nx_{u,\tau}) \cdot H(\tau)^{sk_u} \mod N^2 \tag{2}$$

- $[Y'_\tau]_u \leftarrow$ **TJL-A.ShareProtect**$(pp, \{[\Delta sk_v]_u\}_{v\in\mathcal{U}_{\text{off}}}, \tau, x'_{u,\tau})$: As opposed to the original **TJL.ShareProtect**, this algorithm protects $x'_{u,\tau}$ which is a non-zero value using the shares of the keys of dropped clients $\mathcal{U}_{\text{off}}$ and **ISS** parameters Δ and μ_u:

$$[Y'_\tau]_u = (1 + Nx'_{u,\tau})^{\Delta^2\mu_u^{-1}} \cdot H(\tau)^{\sum_{v\in\mathcal{U}_{\text{off}}}[\Delta sk_v]_u} \mod N^2 \tag{3}$$

- $Y'_\tau \leftarrow$ **TJL-A.ShareCombine**$(\{u, [Y'_\tau]_u\}_{u\in\mathcal{U}_{\text{on}}}, \tau)$: This algorithm calls **ISS.Recon** and reconstructs the non-zero value that will be further used for the final aggregation. Hence, Y'_τ is computed as $\prod_{u\in\mathcal{U}_{\text{on}}}([Y'_\tau]_u)^{\mu_u} \mod N^2$, here $\mathcal{U}_{\text{on}}$ being the set of online clients, and we obtain:

$$Y'_\tau = \left(1 + N\sum_{u\in\mathcal{U}_{\text{on}}} x'_{u,\tau}\right)^{\Delta^2} \cdot H(\tau)^{\Delta^2\sum_{v\in\mathcal{U}_{\text{off}}} sk_v} \mod N^2 \tag{4}$$

- $x_\tau \leftarrow$ **TJL-A.Agg**$(pp, sk_0, \tau, \{Y_{u,\tau}\}_{u\in\mathcal{U}_{\text{on}}}, Y'_\tau)$: Given public parameters pp, aggregation key sk_0, the ciphertexts of online clients $u \in \mathcal{U}_{\text{on}}$, and the ciphertext of the non-zero value corresponding to dropped clients $v \in \mathcal{U}_{\text{off}}$, this algorithm first computes the aggregated ciphertext Y_τ for round τ as $Y_\tau = \left(\prod_{u\in\mathcal{U}_{\text{on}}}(Y_{u,\tau})\right)^{\Delta^2} \cdot Y'_\tau \mod N^2$ which corresponds to:

$$Y_\tau = \left[1 + N\Delta^2\left(\sum_{u\in\mathcal{U}_{\text{on}}} x_{u,\tau} + \sum_{u\in\mathcal{U}_{\text{on}}} x'_{u,\tau}\right)\right] \cdot H(\tau)^{\Delta^2\cdot\sum_{u\in\mathcal{U}} sk_u} \mod N^2 \tag{5}$$

Finally, to obtain the overall aggregate in cleartext, the algorithm computes:

$$V_\tau = H(\tau)^{\Delta^2\cdot sk_0} \cdot Y_\tau, \quad x_\tau = \frac{V_\tau - 1}{N\Delta^2} \mod N.$$

This recovers the sum of the inputs:

$$x_\tau = \sum_{u\in\mathcal{U}_{\text{on}}} w_{FedLA,u}\theta_u^\tau + \left(1 - \sum_{u\in\mathcal{U}_{\text{on}}} w_{FedLA,u}\right)\theta^{(\tau-1)} \tag{6}$$

4 SAAFL

In this section, we describe SAAFL, which uses **TJL-A** to compute each client's FedLA weight in a privacy-preserving manner, along with a traditional fault-tolerant SA scheme for aggregation. SAAFL also makes use of a pseudo-random generator (PRG), key aggreement scheme (KA), authenticated encryption (AE), and the Shamir's secret sharing Scheme (SS) which due to space constraints are not described in the paper but the reader can refer to [11] as these are the same protocols. SAAFL consists of two phases: (1) *Setup Phase* whereby FL clients are registered and each party receives its keying material and (2) *training phase* whereby model parameters are aggregated and updated in several *FL Rounds*.

Setup Phase. The protocol starts with the *Setup Phase* which comprises three steps: *Registration*, *Key Setup* and *FedLA-weight Calculation*. Each of these steps is executed only once. At the *Registration* step, a trusted dealer (TD) generates the secret keys sk_u for each client and aggregation key sk_0 for the server along with relevant public parameters pp required by the clients and server. Each client generates public-private key pair over the pp using **KA.Gen** and registers itself by sending the public key to the server. At the *Key Setup* step, once each client receives the list of all registered clients together with their public keys, they establish pairwise keys with each of them and use **TJL-A.SKShare** to generate the shares of their individual **TJL-A** key sk_u. Finally, at the *FedLA-weight Calculation* step, each FL client u encrypts $s_{u,l}$ of data samples with label l with the **TJL-A** secret key sk_u (generated in the previous *Key Setup* step) and sends them to the FL server. The server performs their aggregation using **TJL-A.Agg** and sends s_l of samples with label l to all FL clients. Then, each client calculates its FedLA-weight $w_{FedLA,u}$ (see Eq. 1).

The training phase. The training phase consists of the execution of several *FL Rounds* until reaching a certain level of accuracy. Each *FL Rounds* consists of two steps: *Protection* and *Aggregation*. At the *Protection* step, all selected clients encrypt their inputs and send them to the server. Then, at the *Aggregation* step, clients interact with the server to account for the dropped and non-selected clients so that the server can perform the aggregation correctly and accurately. Each FL round begins with the selection of clients $\mathcal{U}^\tau \subseteq \mathcal{U}$ that will participate in the aggregation, and this is performed based on selection ratio k_τ as in the FedLA and ensures that all parties are aware of which clients will participate in the current *FL Rounds* τ in advance.

At the *Protection* step, after training the local models, each client generates a random seed $b_{u,\tau}$ to generate a blinding mask $B_{u,\tau}$ for masking its input $x_{u,\tau}$ and then encrypts it using **TJL-A.Protect**. These blinding masks are used in the same way as in [11, 4] to protect the privacy of the clients' inputs in case the server receives the protected values of a given client (usually a straggler) twice using the same τ. Each client then secretly shares seed $b_{u,\tau}$ with other clients using **SS.Share** through the server and transmits its protected input $Y_{u,\tau}$ masked with $B_{u,\tau}$ to the server.

At the *Aggregation* step, the clients receive the list of encrypted shares of blinding mask seeds $b_{u,\tau}$ and deduce the list of online clients $\mathcal{U}^\tau_{\text{on}}$ from those

SAAFL - Setup Phase

Registration: $|\mathcal{U}| = n$. $|\mathcal{L}| = k$.

- **Trusted Dealer (TD):**
  1. Run $pp^{\text{KA}} \leftarrow$ **KA.Param**(λ) and $(sk_0, \{sk_u\}_{u\in\mathcal{U}}, N, H, \sigma) \leftarrow$ **TJL-A.Setup**(λ).
  2. Set $pp = (pp^{\text{KA}}, N, H, \sigma, t, n, m, R, \mathbb{F})$ where m is the number of model parameters defined in $\mathbb{Z}_R$, t is the threshold and $\mathbb{F}$ is the field for the **SS** scheme.
  3. Send (pp, sk_u) to each client u and (pp, sk_0) to server.
- **Clients** $(\forall u \in \mathcal{U})$**:**
  1. Receive (pp, sk_u) from TD.
  2. $(c_u^{\text{PK}}, c_u^{\text{SK}}) \leftarrow$ **KA.Gen**(pp^{KA}): Generate a key pair and send c_u^{PK} to the server.
- **Server:**
  1. Receive (pp, sk_0) from TD and c_u^{PK} from each client $u \in \mathcal{U}$.
  2. Broadcast to all clients the list $\{u, c_u^{\text{PK}}\}_{u\in\mathcal{U}}$.

Key Setup:

- **Clients** $(\forall u \in \mathcal{U})$**:**
  1. Collect $\{v, c_v^{\text{PK}}\}_{v\in\mathcal{U}}$ of all registered clients.
  2. $\forall v \in \mathcal{U} \setminus \{u\}$, $c_{u,v} \leftarrow$ **KA.Agree**$(\text{pp}^{\text{KA}}, c_u^{\text{SK}}, c_v^{\text{PK}}, H)$: Establish pairwise shared keys.
  3. $\{(v, [sk_u]_v)\}_{v\in\mathcal{U}} \leftarrow$ **TJL-A.SKShare**$(sk_u, t, \mathcal{U})$: Generate t-out-of-n shares of sk_u.
  4. $\forall v \in \mathcal{U} \setminus \{u\}$, $\epsilon_{u,v} \leftarrow$ **AE.Enc**$(c_{u,v}, u \,\|\, v \,\|\, [sk_u]_v)$: Encrypt each share with $c_{u,v}$ and send $\{(u, v, \epsilon_{u,v})\}_{v\in\mathcal{U}}$ to server.
- **Server:**
  1. Collect and forward $\{(u, v, \epsilon_{u,v})\}_{v\in\mathcal{U}}$ to registered clients.

FedLA-weight Calculation:

- **Clients** $(\forall u \in \mathcal{U})$**:**
  1. $\forall l \in \mathcal{L}, W_{u,l} \leftarrow$ **TJL-A.Protect**$(pp, sk_u, \tau_0, s_{u,l})$: Protect the local label distribution $s_{u,l}$ and send $W_{u,l}$ to the server.
- **Server:**
  1. Collect all $W_{u,l}$ from clients.
  2. $\forall l \in \mathcal{L}, s_l \leftarrow$ **TJL-A.Agg**$(pp, sk_0, \tau_0, W_{u,l}, Y'_{\tau_0})$: Aggregate all $W_{u,l}$ to obtain the total number s_l of samples with label l. Here, $Y'_{\tau_0} = 1$.
- **Clients** $(\forall u \in \mathcal{U})$:
  1. Compute $w_{FedLA,u}$ (see Eq. 1 in Section 2.1).

Fig. 1: Detailed description of the *Setup Phase* of SAAFL

shares. Online clients encrypt the relevant value $x'_{u,\tau}$ for non-selected and dropped clients $\mathcal{U} \setminus \mathcal{U}^{\tau}_{\text{on}}$ with **TJL-A.ShareProtect** and the key-shares of these clients. This is to ensure that the FedLA-weights of non-selected and dropped clients also appear in the final aggregation as discussed in Section 2.3. Then, each online client sends to the server the shares of the blinding mask seeds for other online clients and the share of the protected non-zero value $[Y'_{\tau}]_u$ for the non-selected and dropped clients. The server reconstructs the blinding masks using **SS.Recon** and **PRG** and combines the shares of the protected approximated value $[Y'_{\tau}]_u$ using **TJL-A.ShareCombine**. Finally, the server computes the masked sum c_{τ} of the weighted inputs of online clients $x_{u,\tau}$ and the approximated value $x'_{u,\tau}$ for non-selected and dropped clients with **TJL-A.Agg** and removes the clients' blinding masks to obtain the aggregated model x_{τ} for *FL Round* τ.

Security Analysis: SAAFL must guarantee the privacy of the clients' inputs against the passive and active adversaries. We sketch the proofs following [11].

SAAFL - FL Round (τ)

Client Selection: Clients are selected based on the parameter k_τ and previous round model $\theta^{\tau-1}$. Let $\mathcal{U}^\tau \subseteq \mathcal{U}$ represent the set of selected clients in the current FL round τ.
Protection:

- **Clients** ($\forall u \in \mathcal{U}^\tau$):
  1. $b_{u,\tau} \xleftarrow{R} \mathbb{F}$: Sample a seed for **PRG**.
  2. $B_{u,\tau} \leftarrow \mathbf{PRG}(b_{u,\tau})$: Extend the seed using **PRG** to produce the blinding mask.
  3. $x_{u,\tau} \leftarrow w_{FedLA,u}\theta_u^\tau - w_{FedLA,u}\theta^{(\tau-1)}$: Compute client u's input.
  4. $Y_{u,\tau} \leftarrow \mathbf{TJL\text{-}A.Protect}(pp, sk_u, \tau, x_{u,\tau} + B_{u,\tau})$: Mask and protect the input $x_{u,\tau}$.
  5. $\{(v, [b_{u,\tau}]_v)\}_{v\in\mathcal{U}^\tau} \leftarrow \mathbf{SS.Share}(b_{u,\tau}, t, \mathcal{U}^\tau)$: Generate t-out-of-n shares of $b_{u,\tau}$.
  6. $\forall v \in \mathcal{U}^\tau \setminus \{u\}$, $e_{(u,v),\tau} \leftarrow \mathbf{AE.Enc}(c_{u,v}, u\|v\|[b_{u,\tau}]_v)$: Encrypt shares.
  7. If any of the above steps fails, abort. Otherwise, send $(\{e_{(u,v),\tau}\}_{v\in\mathcal{U}^\tau}, Y_{u,\tau})$ to the server.
- **Server:**
  1. Collect $(\{e_{(u,v),\tau}\}_{v\in\mathcal{U}^\tau}, Y_{u,\tau})$ from remaining clients.
  2. Define the set of online clients $\mathcal{U}^\tau_{\text{on}} \subseteq \mathcal{U}^\tau$ based on received tuples. If $|\mathcal{U}^\tau_{\text{on}}| < t$, then abort. Otherwise, send $\{(u, v, e_{(u,v),\tau})\}_{u,v\in\mathcal{U}^\tau_{\text{on}}}$ to online clients.

Aggregation:

- **Clients** ($\forall u \in \mathcal{U}^\tau_{\text{on}}$):
  1. Receive $\{(v, u, e_{(v,u),\tau})\}_{v\in\mathcal{U}^\tau_{\text{on}}}$ from the server and deduce $\mathcal{U}^\tau_{\text{on}}$.
  2. $\forall v \in \mathcal{U}^\tau_{\text{on}} \setminus \{u\}$, $v'\|u'\|[b_{v,\tau}]_u \leftarrow \mathbf{AE.Dec}(c_{u,v}, e_{(v,u),\tau})$: Decrypt secret shares. Assert that $v' = v$ and $u' = u$.
  3. $x'_{u,\tau} = \frac{1}{|\mathcal{U}^\tau_{\text{on}}|}\theta^{(\tau-1)}$: Compute the approximation for non-selected and dropped clients.
  4. $[Y'_\tau]_u \leftarrow \mathbf{TJL\text{-}A.ShareProtect}(\text{pp}, \{[\Delta sk_v]_u\}_{v\in\mathcal{U}\setminus\mathcal{U}^\tau_{\text{on}}}, \tau, x'_{u,\tau})$: Protect $x'_{u,\tau}$ with client u's key shares of non-selected and dropped clients.
  5. If any of above steps fails, abort. Otherwise, send $(\{[b_{v,\tau}]_u\}_{v\in\mathcal{U}^\tau_{\text{on}}}, [Y'_\tau]_u)$ to the server.
- **Server**
  1. Collect $(\{[b_{v,\tau}]_u\}_{v\in\mathcal{U}^\tau_{\text{on}}}, [Y'_\tau]_u)$ from online clients.
  2. $\forall u \in \mathcal{U}^\tau_{\text{on}}$, $b_{u,\tau} \leftarrow \mathbf{SS.Recon}(\{[b_{u,\tau}]_v\}_{v\in\mathcal{U}^\tau_{\text{on}}}, t)$: Reconstruct blinding mask seeds.
  3. $\forall u \in \mathcal{U}^\tau_{\text{on}}, B_{u,\tau} \leftarrow \mathbf{PRG}(b_{u,\tau})$: Reconstruct the blinding masks.
  4. $Y'_\tau \leftarrow \mathbf{TJL\text{-}A.ShareCombine}(\{[Y'_\tau]_u\}_{u\in\mathcal{U}^\tau_{\text{on}}}, t)$: Reconstruct the protected approximations.
  5. $c_\tau \leftarrow \mathbf{TJL\text{-}A.Agg}(\text{pp}, sk_0, \tau, \{Y_{u,\tau}\}_{u\in\mathcal{U}^\tau_{\text{on}}}, Y'_\tau)$: Aggregate all inputs from the online clients and all approximations from the dropped and non-selected clients.
  6. $x_\tau = c_\tau - \sum_{v\in\mathcal{U}^\tau_{\text{on}}} B_{u,\tau} = \sum_{u\in\mathcal{U}^\tau_{\text{on}}} x_{u,\tau} + \sum_{u\in\mathcal{U}^\tau_{\text{on}}} x'_{u,\tau}$: Calculate the final result by removing the blinding masks.

Fig. 2: Detailed description of the FL Round in SAAFL

Security in the Passive Model: In the honest-but-curious model, the server follows the protocol but colludes with up to $n - t$ clients. Let $\mathcal{U}_C$ represent the corrupted clients, and $\mathcal{C} = \mathcal{U}_C \cup \mathcal{S}$ where $\mathcal{S}$ is the server. The view of $\mathcal{C}$ is computationally indistinguishable from a simulated view if $|\mathcal{U}_C| = n - t < t$. Therefore, $t > n/2$, allowing recovery from up to $n/2 - 1$ client failures. This relies on the security of the underlying cryptographic primitives. KA ensures that entities in $\mathcal{C}$ cannot distinguish the honest clients' pairwise keys from random values. The **TJL-A** scheme ensures that protected inputs and honest clients' shares appear random if entities in $\mathcal{C}$ have fewer than t shares. SS (and ISS) guarantees that entities in $\mathcal{C}$ cannot distinguish the shares from random values if they have no more than $t - 1$ shares.

Security in the Active Model: In the active model, the server can manipulate its inputs. The server only distributes clients' public keys and encrypted shares, which it cannot alter due to the AE scheme. The server's power is limited to withholding shares, potentially misleading clients about which are online. Importantly, the server can provide different views to clients, convincing some to send a protected approximation Y'_τ for client u, while others send shares of the blinding mask $b_{u,\tau}$. If the server obtains both Y'_τ and $b_{u,\tau}$ for the same client u in round τ, it can unmask the input. Colluding with $n-t$ corrupted clients, the server can gain $n-t$ shares of Y'_τ and $b_{u,\tau}$. For the remaining t honest clients, it can manipulate half to send shares of Y'_τ and the other half shares of $b_{u,\tau}$, learning up to $n-t+t/2$ shares. Hence, $n-t+t/2 < t$, implying $t > 2n/3$, allowing recovery from up to $n/3-1$ client failures.

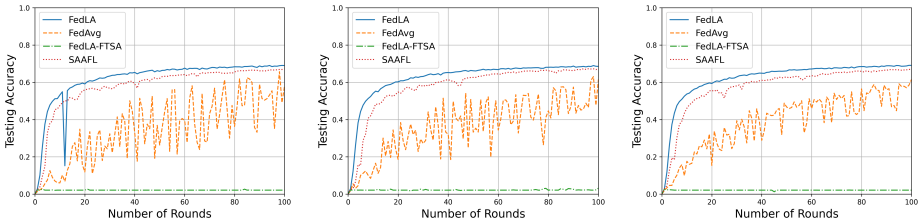
5 Performance Evaluation

Experimental Setup. We have implemented SAAFL[5] and to evaluate its performance, experiments are conducted on a machine equipped with an AMD EPYC 7272 processor, 128 GB of RAM, and 4 NVIDIA GeForce RTX 3090 graphics cards. The underlying cryptographic primitives consist of the same building blocks and libraries as in [11]. We use the EMNIST-balanced dataset [6] with $k = 47$ labels, each of which represented by 2400 samples. This is one of the two datasets utilized in [9] to evaluate the original FedLA. We also employ the same CNN model of [9] which consists of two convolutional layers, one dropout layer, and two linear layers. The model is configured with a batch size of 256 and a learning rate of 0.01.

To evaluate the accuracy of the resulting model, similar to [9], the EMNIST-balanced dataset is distributed among $n = |\mathcal{U}| = 10$ clients according to two parameters: the number of unique labels that each non-IID client has, denoted as $unique_c$ and $noniid_s$ which is the proportion of non-IID clients. We vary $unique_c$ across $\{1, 2, 3, 4, 6\}$ and $noniid_s$ across $\{0.1, 0.3, 0.5, 0.7\}$. Each IID client holds 2400 samples while each non-IID client holds all samples of the labels to which it is assigned: e.g., if $unique_c = 2$, each non-IID client holds $2 \times 2400 = 4800$ samples. At each FL round, we define the selection ratio $k_\tau \in \{0.7, 0.8, 0.9\}$. Each setting is evaluated by running experiments over 100 FL rounds each with 10 local epochs. In our protocol, non-selected clients are treated as dropped. We set the **TJL-A** threshold to $t = 6$ and allow a maximum dropout rate f of 0.1, 0.2, or 0.3 when k_τ is 0.7, 0.8, or 0.9, respectively. We further evaluate the performance of SAAFL in terms of computation and communication while considering 100 users and $k_\tau = 0.8$, $f = 0.2$.

Experimental Results. First, we compare testing accuracy of SAAFL under a non-IID setting with that of FedAvg, original FedLA, and FedLA-FTSA [11] which executes **TJL** with dropout clients' inputs set to zero. We evaluate worst-case scenario for SA, i.e., a maximum dropout rate that each method can handle.

[5] The code is available in https://github.com/Aftab201/SAAFL.git.



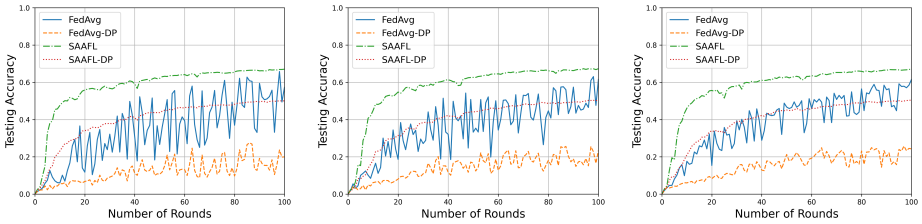
(a) $k_\tau = 0.7, f = 0.1$ (b) $k_\tau = 0.8, f = 0.2$ (c) $k_\tau = 0.9, f = 0.3$

Fig. 3: Testing accuracy for FedLA, FedAvg, FedLA-FTSA, and SAAFL with $unique_c = 1$, $noniid_s = 0.7$.

Since FedLA has been proved to have superior accuracy to FedAvg in data distribution in which the non-IID clients are the majority [9], we choose to discuss the results of the experiments in which $unique_c = 1$ and $noniid_s = 0.7$. As shown in Figure 3 and as expected, FedAvg suffers more unstable performance than FedLA and SAAFL because of the highly non-IID data distribution. We also observe that FedLA-FTSA yields worst performance mainly because dropped clients' inputs are considered as zero and this does not help achieve aggregation correctness in FedLA. Meanwhile, SAAFL achieves almost as good accuracy level as FedLA thanks to the proposed **TJL-A** scheme which sets dropped clients' inputs as model parameters of the previous round weighted with specific coefficients. Furthermore, in Figure 3a, FedLA has significant accuracy drop in one early round in which all selected clients have non-IID data distribution. This may happen when proportion of non-IID clients $noniid_s$ is equal to the selection ratio k_τ. On the contrary, SAAFL avoids this issue by adding to FedLA-weighted average $\sum_{u \in \mathcal{U}^\tau_{\text{on}}} w_{FedLA,u} \theta^\tau_u$ an approximated value $\left(1 - \sum_{u \in \mathcal{U}^\tau_{\text{on}}} w_{FedLA,u}\right) \theta^{(\tau-1)}$ that compensates for the impact of non-IID clients.

Furthermore, we also propose to integrate differentially private stochastic gradient descent (DP-SGD) [1] in the local training of each client using Opacus[20] in order to add another layer of privacy protection to protect against global membership inference attacks. We compare the performance of FedAvg and SAAFL. DP-SGD requires three extra parameters: the maximum gradient norm C, privacy budget per client per FL round (ϵ, δ). Fine-tuning of these parameters is required to balance the negative impact of DP to the performance and the privacy protection level. We choose optimized parameters $C = 7.0, \epsilon = 2.0, \delta = 10^{-5}$ for these experiments. Figure 4 shows the accuracy of the FedAvg and SAAFL when applying DP-SGD. While, due to the noise added to the models, both methods experience a performance decrease, compared to the results without DP-SGD, FedAvg has a more significant drop in testing accuracy than SAAFL does.

We finally evaluate the performance of SAAFL in terms of computation time and communication cost. We set the number of clients to 100. As shown in Table 1, with $k_\tau = 0.8$, and $f = 0.2$, each client takes 249 ms to run the *Key Setup* whereas the server takes 1.55 ms. The calculation of *FedLA-weights*, takes 38 ms



(a) $k_\tau = 0.7, f = 0.1$ (b) $k_\tau = 0.8, f = 0.2$ (c) $k_\tau = 0.9, f = 0.3$

Fig. 4: Testing accuracy for FedAvg, and SAAFL with DP-SGD ($C = 7.0, \epsilon = 2.0, \delta = 10^{-5}$) with $unique_c = 1$, $noniid_s = 7$.

at the client and 1085 ms at the server. Finally, at each *FL Round*, each client takes 6463 ms for the SA of model updates and the server, 46475 ms. The data transfer at each client in *Key Setup*, *FedLA-weight Calculation*, and *FL Round* are measured as 14.27 KB, 14.64 KB, and 24.88 KB, respectively.

Table 1: Computation and Communication Cost of SAAFL with $n = 100$, $k_\tau = 0.8$, $f = 0.2$.

SAAFL	**Server**	**Client**	**BW**
Key Setup	1.55 ms	249 ms	14.27 KB
FedLA-weight Calculation	1085 ms	38 ms	14.64 KB
FL Round	46457 ms	6463 ms	24.88 KB

When compared with the solution in [11], SAAFL incurs the same complexity, namely: $O(n + m)$ for the client and $O(n^2 + nm)$ for the server both for communication and storage, and $O(n^2 + m)$ for the client and $O(n^2 + nm)$ for the server in terms of computation. (n and m being the number of clients and the number of model parameters, respectively).

6 Related Work

We study the related work according to the three challenges that we have identified in Section 2. Many SA schemes have been proposed for FL [12]: The majority of them unfortunately omit the case with non-IID clients (C_1) and use the standard FedAvg solution to obtain the aggregated model. Such solutions become ineffective in the non-IID case. Regarding client dropouts (C_2), existing solutions either (i) rely on online clients to reconstruct offline clients' inputs [4, 11] or, (ii) define dedicated parties such as *decryptors* in [10], to ensure aggregation correctness, or (iii) consider online clients' inputs only and define aggregation keys on the fly [18] (this also seems to address client selection). SAAFL, relies on online

clients but to achieve a good level of accuracy in the non-IID context, the reconstructed value for dropped clients is not zero but a customized one. Recently, [16] develops a FL scheme that uses a dedicated polynomial integer neural network to address both non-IID distribution and client dropouts. Regarding C_3, most of the existing solutions, including [16], assume that client selection has already happened and usually omit this important operation, whereas in SAAFL, non-selected clients are considered as dropouts too. Finally, the very recent work in [5] considers the use of DP mechanisms for non-IID clients. This work can be considered as independent as it does not implement SA. A combination of this work with SAAFL can be an interesting future work direction.

7 Conclusion and Future Work

We have developed a new privacy-preserving FL framework named SAAFL that is suitable to environments where clients datasets are non-IID and runs in two communication rounds, only. SAAFL is based on a customized SA protocol **TJL-A** where the dropped clients' shares depend on the previous round's parameters. Thanks to **TJL-A** and as shown by our experimental results, SAAFL exhibits an acceptable level of accuracy. Finally, the DP-SGD mechanism is used to add another protection level for global model parameters. For future work, we aim to address stronger threat models, ensuring the privacy and correctness of the aggregated value even in the presence of malicious users and/or aggregators.

Acknowledgments

This work has been partially supported by the TRAIN project number ANR-22-FAI1-0003, the 3IA Côte d'Azur programme reference number ANR-23-IACL-0001 and the project number ANR-23-CPJ1-0060-01.

References

1. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep Learning with Differential Privacy. In: ACM SIGSAC Conference on Computer and Communications Security (ACM CCS) (2016)
2. Bell, J.H., Bonawitz, K.A., Gascón, A., Lepoint, T., Raykova, M.: Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. In: ACM SIGSAC Conference on Computer and Communications Security (CCS) (2020)
3. Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, B., et al.: Towards federated learning at scale: System design (2019)
4. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical Secure Aggregation for Privacy-Preserving Machine Learning. In: ACM SIGSAC Conference on Computer and Communications Security (CCS) (2017)
5. Chen, L., Ding, X., Bao, Z., Zhou, P., Jin, H.: Differentially Private Federated Learning on Non-iid Data: Convergence Analysis and Adaptive Optimization. IEEE Transactions on Knowledge and Data Engineering **36**, 4567 – 4581 (2024)

6. Cohen, G., Afshar, S., Tapson, J., Van Schaik, A.: Emnist: Extending mnist to handwritten letters. In: International Joint Conference on Neural Networks (IJCNN) (2017)
7. Joye, M., Libert, B.: A scalable scheme for privacy-preserving aggregation of time-series data. In: International Conference on Financial Cryptography and Data Security (FC) (2013)
8. Kadhe, S., Rajaraman, N., Ramchandran, K.: Fastshare: Scalable secret sharing by leveraging locality. In: 2021 IEEE International Symposium on Information Theory (ISIT) (2021)
9. Khalil, A., Wainakh, A., Zimmer, E., Parra-Arnau, J., Anta, A.F., Meuser, T., Steinmetz, R.: Label-aware aggregation for improved federated learning. In: International Conference on Fog and Mobile Edge Computing (FMEC) (2023)
10. Ma, Y., Woods, J., Angel, S., Polychroniadou, A., Rabin, T.: Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In: IEEE Symposium on Security and Privacy (SP) (2023)
11. Mansouri, M., Önen, M., Ben Jaballah, W.: Learning from Failures: Secure and Fault-Tolerant Aggregation for Federated Learning. In: 38th Annual Computer Security Applications Conference (ACSAC) (2022)
12. Mansouri, M. and Önen, M. and Ben Jaballah, W. and Conti, M.: Sok: Secure aggregation based on cryptographic schemes for federated learning. In: Proceedings on Privacy Enhancing Technologies (PoPETS) (2023)
13. McMahan, B., Moore, E., Ramage, D., Hampson, S., Aguera y Arcas, B.: Communication-efficient learning of deep networks from decentralized data. In: International Conference on Artificial Intelligence and Statistics (AISTATS) (2017)
14. Nasr, M., Shokri, R., Houmansadr, A.: Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In: IEEE Symposium on Security and Privacy (SP) (2019)
15. Oza, P., Patel, V.M.: Federated learning-based active authentication on mobile devices. In: IEEE international joint conference on biometrics (IJCB) (2021)
16. Shao, J., Sun, Y., Li, S., Zhang, J.: DReS-FL: dropout-resilient secure federated learning for non-IID clients via secret data sharing. In: 36th International Conference on Neural Information Processing Systems (NIPS) (2022)
17. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: IEEE Symposium on Security and Privacy (SP) (2017)
18. Taiello, R., Önen, M., Gritti, C., Lorenzi, M.: Let them drop: Scalable and efficient federated learning solutions agnostic to client stragglers. In: International Conference on Availability, Reliability and Security (ARES) (2024)
19. Xiao, P., Cheng, S., Stankovic, V., Vukobratovic, D.: Averaging is probably not the optimum way of aggregating parameters in federated learning. Entropy **22**(3), 314 (2020)
20. Yousefpour, A., Shilov, I., Sablayrolles, A., Testuggine, D., Prasad, K., Malek, M., Nguyen, J., Ghosh, S., Bharadwaj, A., Zhao, J., Cormode, G., Mironov, I.: Opacus: User-friendly differential privacy library in PyTorch (2021)
21. Zari, O., Xu, C., Parra-Arnau, J., Ünsal, A., Önen, M.: Link Inference Attacks in Vertical Federated Graph Learning. In: 40th Annual Computer Security Applications Conference (ACSAC) (2024)