Complex & Intelligent Systems (2024) 10:677–690
https://doi.org/10.1007/s40747-023-01184-3

ORIGINAL ARTICLE

Fedlabx: a practical and privacy-preserving framework for federated learning

Yuping Yan[1,2] · Mohammed B. M. Kamel[1,4,5] · Marcell Zoltay[2] · Marcell Gál[2] · Roland Hollós[2] · Yaochu Jin[3] · Ligeti Péter[1] · Ákos Tényi[2]

Received: 8 January 2023 / Accepted: 11 July 2023 / Published online: 31 July 2023

Abstract
Federated learning (FL) draws attention in academia and industry due to its privacy-preserving capability in training machine learning models. However, there are still some critical security attacks and vulnerabilities, including gradients leakage and interference attacks. Meanwhile, communication is another bottleneck in basic FL schemes since large-scale FL parameter transmission leads to inefficient communication, latency, and slower learning processes. To overcome these shortcomings, different communication efficiency strategies and privacy-preserving cryptographic techniques have been proposed. However, a single method can only partially resist privacy attacks. This paper presents a practical, privacy-preserving scheme combining cryptographic techniques and communication networking solutions. We implement Kafka for message distribution, the Diffie–Hellman scheme for secure server aggregation, and gradient differential privacy for interference attack prevention. The proposed approach maintains training efficiency while being able to addressing gradients leakage problems and interference attacks. Meanwhile, the implementation of Kafka and Zookeeper provides asynchronous communication and anonymous authenticated computation with role-based access controls. Finally, we prove the privacy-preserving properties of the proposed solution via security analysis and empirically demonstrate its efficiency and practicality.

Keywords Federated learning · Kafka · Secure aggregation · Differential privacy

Introduction

The increasing demand to process the generated data on the Internet has led to the growing use of Machine Learning (ML) methods. Traditionally, a single server or data center stores data collected from different parties and centrally performs model training. This paradigm is called centralized learning. The centralized learning mode is vulnerable to leakage of personal and private information. Numerous studies [16, 36] have shown that by analyzing the output of machine learning mode, attackers can reversely infer sensitive information about individuals in the training dataset, e.g., bank transaction records and personal medical data.

Without proper regulations and privacy consents, it is challenging to effectively aggregate datasets from different industry domains for privacy concerns in practice. Countries worldwide have introduced a series of legal regulations to protect user privacy. In March 2018, the EU General Data Protection Regulation [34] came into force, setting precise requirements for companies to process user data. Health Insurance Portability and Accountability (HIPAA) [3] also

✉ Yuping Yan
yupingyan@inf.elte.hu

Mohammed B. M. Kamel
mkamel@inf.elte.hu; mkamel@hs-furtwangen.de

Yaochu Jin
yaochu.jin@uni-bielefeld.de

Ligeti Péter
ligetipeter@inf.elte.hu

1 Department of Computer Algebra, Eötvös Loránd University, Budapest, Hungary

2 Smart Data Group, E-Group ICT Software Zrt, Budapest, Hungary

3 Faculty of Technology, Bielefeld University, 33619 Bielefeld, Germany

4 Institute for Data Science, Cloud Computing and IT Security, IDACUS, Furtwangen University, Furtwangen im Schwarzwald, Germany

5 Department of Computer Science, University of Kufa, Najaf, Iraq

defines how medical data can be used and released. These regulations guide how companies can use data properly and define unauthorized access to the data as illegal. It is forbidden to deliberately and unlawfully attempt to access or gain entry to computer systems, networks, or data without a proper authorization or permission. Therefore, a far-reaching and urgent issue is how to give full play to the potential of machine learning and other artificial intelligence methods while ensuring user privacy and data security.

Recently, some researchers have tried to train global models while keeping data from all participants locally. A typical case is Federated Learning (FL), which was proposed by McMahan et al. [26]. FL shows its advantages in providing user privacy protection without sharing data among participants. It is a good solution for the data silos problem, which can learn from different datasets and improve the generalization effect of the model. Meanwhile, clients can adopt different machine learning training algorithms flexibly. Technically, the FL architecture has the potential to apply to any industry, but it is essential to consider the security issues. The basic Federated Averaging (FedAvg) scheme can not guarantee data privacy. A malicious participant can infer sensitive information about original data sets from the shared models and hyper-parameters. This gradients leakage problem [35] is one of the significant security concerns in FL. Similarly, data reconstruction attacks [24] and interference attacks [31] can be conducted on the classic FL schemes. Even though cryptographic primitives such as homomorphic encryption (HE), differential privacy (DP), and secure multi-party computation (MPC) schemes are widely applied to FL, there are trade-offs between accuracy, efficiency, and learning performance.

The typical attack points in an FL scenario can be the compromised server and malicious clients. On the one hand, the server deployed by a service provider is considered a passive attacker with honest-but-curious security models. The server usually provides its services strictly following the established learning protocols, but it may also leak some sensitive information about the user from local model updates. On the other hand, participants are considered active attackers who try to recover sensitive information about the other participants from the global model parameters shared by the training data formation.

There are several open-access FL frameworks and libraries available for academia and industry, such as TensorFlow Federated (TFF) [1] proposed by Google, FedML [15], FATE [38] by WeBank, Flower [5] structure, among others. Different platforms use various strategies. However, implementing these libraries requires a lot of effort to re-engineer to accommodate new ML models. Meanwhile, they are all client–server structures and do not support managed and access-controlled networks. We will conduct a comparative analysis with these proposed schemes, especially regarding privacy mechanisms in the "Comparisons of different FL frameworks".

In this paper, we propose a Kafka-based privacy-preserving framework for FL. In this scheme, we improve the network reliability by using Kafka for message distribution and ensuring data privacy with the Diffie-Hellman scheme for secure server aggregation and DP on gradients to prevent interference attacks. The contributions of this paper are summarized as follows:

1. A combined framework of partial Diffie-Hellman protocol and DP scheme is proposed to minimize the main security threats in FL.
2. A practical FL scheme with Apache Kafka and Zookeeper is designed to achieve anonymous authentication with role-based ACLs and support independent and asynchronous model distributions.
3. The proposed method is compared and evaluated with the state-of-the-art with respect to security, efficiency, effectiveness and accuracy via security analysis and empirical studies.

The structure of the paper is as follows. In "Related work and primitives", we will introduce the primitives related to the proposed framework, including Apache Kafka, secure aggregation scheme, and DP. We then propose the main framework and protocol in "Proposed framework". Subsequently, security analysis and performance analysis are conducted in "Security analysis" and "Evaluations", respectively. Finally, we conclude the paper with a conclusion and future work in "Conclusion and future work".

Related work and primitives

This section will introduce the main primitives related to our framework. They include the FL algorithm, Apache Kafka block, secure aggregation scheme, and the DP primitives.

Federated learning

McMahan et al. [26] first introduced the concept of federated learning, which eliminates the need to centrally store the data and allows each participant to update the gradient in a distributed way. It gained popularity for its capability of addressing the data privacy issues, which was missing in the previous distributed machine learning schemes. Generally, FL works with both independent and identically distributed (IID) and non-IID data. FL can be categorized into three groups based on the data features: horizontal, vertical, or hybrid data partition. Horizontal federated learning (HFL) shares feature space but has a different sample space in the data of the clients. By contrast, vertical federated learning

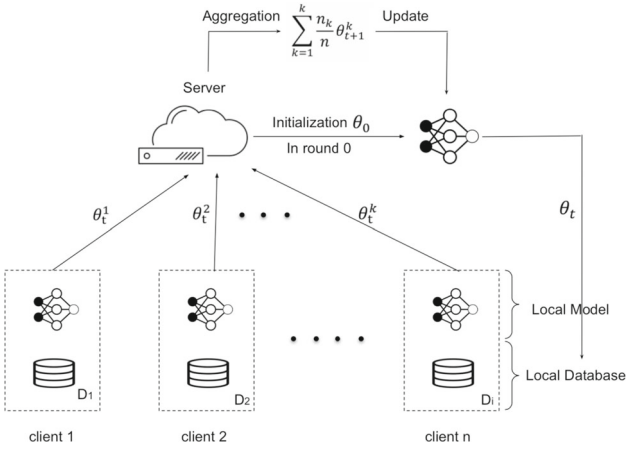

Fig. 1 Training process of federated learning (round r)

(VFL) shares the same sample space but different feature spaces. Hybrid federated learning uses a dataset that has a diverse sample and different feature spaces [38]. A definition of FL is given in Definition 1 and a typical HFL training process is described in Fig. 1.

Definition 1 (*Federated learning*) Let $U = \{U_1, U_2, \ldots, U_n\}$ be a set of n participants, and each participant U_i has its own local dataset D_i. Let U_c be the central server (or aggregator). The task of FL is to compute the global model M_{global} on node U_c based on all uploaded local models $\forall i \in U : M_i$, while each of the local model is generated from the local client dataset D_i.

In the first round, the server will generate a model with random parameters θ_0 and send them to all clients. After receiving the model sent by the server, k out of n participating clients will locally compute the training gradients based on their own dataset and send the updated model to the server. Then, the server aggregates gradients of clients and computes the global parameters $\theta_r = \sum_{i=1}^{k} \theta_i / k$. After a round of updates is completed, the clients check whether the accuracy of the local model meets the requirements and stops training if it does; otherwise, it is ready for the next round of training.

During the FL training process, it is vital to minimize the global accuracy f_{FL}, i.e.,

$$\min_{w} \frac{1}{K} \sum_{n=1}^{N} \sum_{k=1}^{k_n} f_k(w), \tag{1}$$

where w and K represent the global model weights and the number of participating clients, respectively.

The most typical algorithm for local model training is Federated Averaging, or FedAvg for short, whose pseudo code is listed in Algorithm 1. In FL, the server chooses a fraction of clients $0 \leq C \leq 1$ and set the learning rate to η. For each client k, it computes the gradient $w \leftarrow w - \eta \nabla(w, b)$ and sends it to the server. The server will update the parameter $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$ by the weighted average of the gradients.

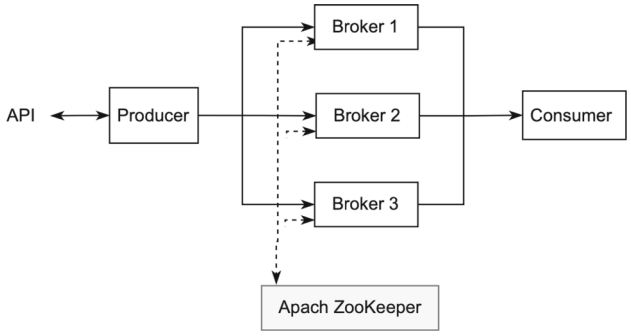

Fig. 2 Communication topology of Kafka cluster

Algorithm 1: FedAvg

Input: The K clients are indexed by k, B is the local mini batch-size, E is number of local epochs, and η is the learning rate
Output: Updated model

```
Server Side: weights initialization w_o
for each round t = 1, 2... do
    Select S_t ← m ← (C * K, 1) clients randomly.
    for each client k ∈ S_t in parallel do
        w_{t+1}^k ← ClientUpdate (k, w_t)
        w_{t+1} ← Σ_{k=1}^{K} (n_k / n) w_{t+1}^k
    end for
end for
Return w_{t+1}
Local update(k, w)
B ← split d_k into batches of size B
for each local epoch e from 1 to E do
    for b ∈ B do
        w ← w − η ∇(w, b)
    end for
end for
Return w
```

Apache Kafka

Apache Kafka [22] is a popular publish/subscribe (pub/sub) system for data pipelines, streaming analytics, data integration, and mission-critical applications. A basic structure of a Client-Kafka/Zookeeper-Producer architecture is shown in Fig. 2.

The basic components of a simple Kafka platform include Zookeeper nodes, Kafka broker nodes, producers, and consumers. There are logical channels on Kafka for separating the message flow, which is named "*topic*". The producers produce the messages, store them in brokers and send messages to the topic, while the consumers who subscribe to this topic

can listen and read messages from the broker. Meanwhile, Kafka implements Zookeeper for robust synchronization. As a centralized service, Zookeeper can trace clients' states, remove/add the nodes, distribute messages, and even make the access list. These components communicate with each other through secure and encrypted communication channels. The current Kafka includes some security supports such as channel security, authentication, and authorization (ACL), which are described in the following:

- **Channel security**: To safely transmit messages through channels between different components, Secure Sockets Layer (SSL)/Transport Layer Security (TLS) secure communication channel is implemented.
- **Authentication**: Simple Authentication and Security Layer (SASL) or SSH (Secure Shell) authentication are used to authenticate clients against brokers or, optionally, against Zookeepers.
- **Authorization (ACL)**: Authorization is a process where the application decides which user has access to the resources and which resources. In this case, one practical example is that one client has the right to publish a message on a topic, and the other one has the right to read messages from this topic. No other action is allowed. These rules are defined in Kafka through Access Control Lists (ACLs).

These unique features make Kafka an excellent solution to handle and mediate communication between two applications. For FL structure, Apache Kafka can provide a secure communication channel for different clients and the server, store and distribute messages. Meanwhile, it can authenticate clients without extra security strategies, such as VPN or username-password authentication. Its distributed and scalable nature makes Kafka well suited for FL. It fits the FL environment for the following reasons:

1. Scalability: Kafka excels in handling high-throughput, large-scale data streaming with low latency, which aligns well with the requirements of real-time processing applications, such as FL. Meanwhile, comparing to other pub/sub systems, such as Amazon Kinesis, Apach Kafka can easily be integrated with other platforms and services.
2. Data integration and distribution: Kafka's publish-subscribe model allows for the integration and distribution of data streams among the participants and the server, which fits the server-client structure of FL.
3. Reliability: Kafka provides reliable communication channels and ensures that data streams are highly available. In FL, where communication is a bottleneck and devices may go offline during the communication, Kafka's resilience ensures that data updates and model synchronization can be reliably managed even during network disruptions.

Secure aggregation

Secure aggregation protocol is proposed by Segal et al. [29] to prevent the gradient leakage attack in FL. It builds on secret sharing [30], public-key cryptography, and pseudo-random number generation. The core idea of the algorithm is to add random noise to the updates that change the values completely but cancels out at the aggregation.

These masks are generated by pairwise *Diffie–Hellmann Key Agreement* [9] that uses pseudo-random generators for creating the noises, whose seeds are given by shared secret keys g^{ab}. Here g^a and g^b are the public keys of the two clients with the secret keys a and b, respectively, and g is a public generator element. The scheme is secure under the Computational Diffie–Hellman assumption.

Definition 2 (*Computational Diffie–Hellman (CDH) assumption*) Consider a cyclic group G of order q, a generator $g \in G$ and random $a, b \in \mathbb{Z}_q^*$, given (g, G, p, g^a, g^b) it is computationally intractable to compute the value g^{ab}.

The protocol of *Diffie–Hellmann Key Agreement* is defined as follows:

1. Alice picks a random natural number a with $1 < a < q$ as her secret key, and sends the element g^a of G to Bob;
2. Bob picks a random natural number b with $1 < b < q$ as hse secret key, and sends the element g^b of G to Alice;
3. Alice computes the element $\left(g^b\right)^a = g^{ba}$ of G;
4. Bob computes the element $(g^a)^b = g^{ab}$ of G.

Thus $g^{ab} = g^{ba}$ will be their shared secret (this is done for every pair of clients). To address the problem of clients' dropout and unresolved noises, sharing the private key a and b via *by Shamir's k-out-of-n threshold secret sharing* [30] is implemented. It results that at least k shares of a secret can reconstruct it perfectly. By having less than k shares, an entity can reconstruct the secret with negligible probability only.

Differential privacy

DP was first proposed by Dwork et al. [10] and is considered the most secure perturbation-based privacy protection method. DP provides statistical privacy guarantees for individual records and prevents interference attacks on the model without incurring additional computational overhead compared to encryption methods.

In the context of FL, the adjacency means the one dataset $\mathcal{D}'$ can be obtained for the other $\mathcal{D}$ by removing the data of

a single client [2]. DP can be achieved by applying *differentially private transformation*. The most common form of these transformations is adding random noise to the data.

There are two categories of DP: the central DP (or global DP) and the local DP, according to where the noise is added. In the local model of DP, the noise is directly added to the local datasets of clients. After collecting these noisy data, the aggregator can compute some statistics and publish them. The significant advantage of this mode is that it does not ask for a trusty aggregator. However, it reduces the accuracy by adding noise to the raw data. By contrast, in the central mode, clients send their trained models to the aggregator, and the aggregator publishes the results after masking them. In this scenario, it compromises with a trusty aggregator, but the final results remain private.

Definition 3 (*Differential privacy*) In *differential privacy* [12], it quantifies and limits information disclosure about an individual with a privacy loss parameter (ϵ, δ). A random algorithm $\mathcal{A}$ is (ϵ, δ)-*differentially private*, if for all $\mathcal{S} \subseteq \text{Range}(\mathcal{A})$ and for all $\mathcal{D}$, and $\mathcal{D}'$ *adjacent* datasets

$$P(\mathcal{A}(\mathcal{D}) \in S) \leq e^{\epsilon} P(\mathcal{A}(\mathcal{D}') \in S) + \delta \tag{2}$$

where S means all possible outcomes of $\mathcal{A}$. If ϵ, the difference between the probabilities for getting the result from dataset $\mathcal{D}$ or $\mathcal{D}'$ is small, that means that it is hard to guess on which has been $\mathcal{A}$ run.

Definition 4 (*Global sensitivity*) The random algorithm $\mathcal{A}$ satisfies the global sensitivity, which tells us how much noise is to be added to the results. Given a sequence of counting queries $\mathcal{Q}$, global sensitivity measures the maximal change on the result when removing one record from the dataset $\mathcal{D}$.

For $\mathcal{Q} : \mathcal{D} \rightarrow \mathcal{R}$, the global sensitivity of $\mathcal{Q}$ is defined as:

$$GS = \max_{D, D'} \left\| Q(D) - Q(D') \right\| \tag{3}$$

To satisfy the differential privacy definition, there are two primary noise mechanisms in DP, namely Laplace mechanism (LM) and exponential mechanism (EM).

Theorem 1 (Laplace mechanism) *Given an function* $Q : D \rightarrow R$*, for an arbitrary domain* D*, the random algorithm* $\mathcal{A}$ *provides* ϵ*-DP, if* $\mathcal{A}$ *satisfies:*

$$\mathcal{A} = Q(D) + \left(\text{Lap}\left(\frac{GS}{\epsilon} \right) \right),$$

where the noise $\text{Lap}\left(\frac{GS}{\epsilon}\right)$ *is drawn from a Laplace distribution, and* d *is the dimension of the query* Q*.*

Theorem 2 (Exponential mechanism) *Given a function* Q*, where its input is dataset* D*, and the output is an entity object* $r \in$ *Range. Let* $q(D, r)$ *be a score function to assign each output* r *a score, and* GS *be the sensitivity of the score function. Then, the mechanism* $\mathcal{A}$ *maintains* ϵ*-DP, if:*

$$\mathcal{A}(r, q) = \left\{ \textit{return } r \textit{ with probability } \propto \exp\left(\frac{\epsilon Q(D, r)}{2GS} \right) \right\}.$$

Proposed framework

This section will explain the FedlabX: Kafka-based privacy-preserving framework in detail. The framework overview is presented in Fig. 3, with the implementations of DP on the trained model gradients, Kafka brokers with Zookeeper for communication, and secure aggregation on the server side.

Security model

We follow the semi-honest threat model in which all the participating parties, including the curious adversaries, follow the protocol properly. Additionally, the parties in the semi-honest model can passively gather and analyze the publicly available data in the system. We assume Probabilistic Polynomial Time (PPT) computational power of the parties in the system. Based on the scheme, we can define the following security and privacy requirements:

Definition 5 (*Correctness*) The proposed model is correct if the server conducts aggregation following the protocol honestly without accuracy loss.

Definition 6 (*Data Privacy*) Any PPT adversary party can infer the original data of the clients from the proposed model with a negligible probability only.

Definition 7 (*Client Unlinkability*) Any PPT adversary party can link the client local data to the aggregated data with a negligible probability only.

FedlabX framework

Our framework has three parties: the server, clients, and the Apache Kafka message queue. In this framework, the server initializes a model and sends the untrained model to the request broker in Kafka. We assume Kafka is embedded in the trusted third-party side. Once the clients request the model from the Kafka request broker, Kafka relies upon the untrained model for the clients. The proposed model allows the central broker to set a policy that permits only specified clients based on a 1-out-of-n verification scheme [18] with predefined n attributes to join the system. We assume that there is an established secure channel between the central server and the clients. A lightweight encryption [19], [20]

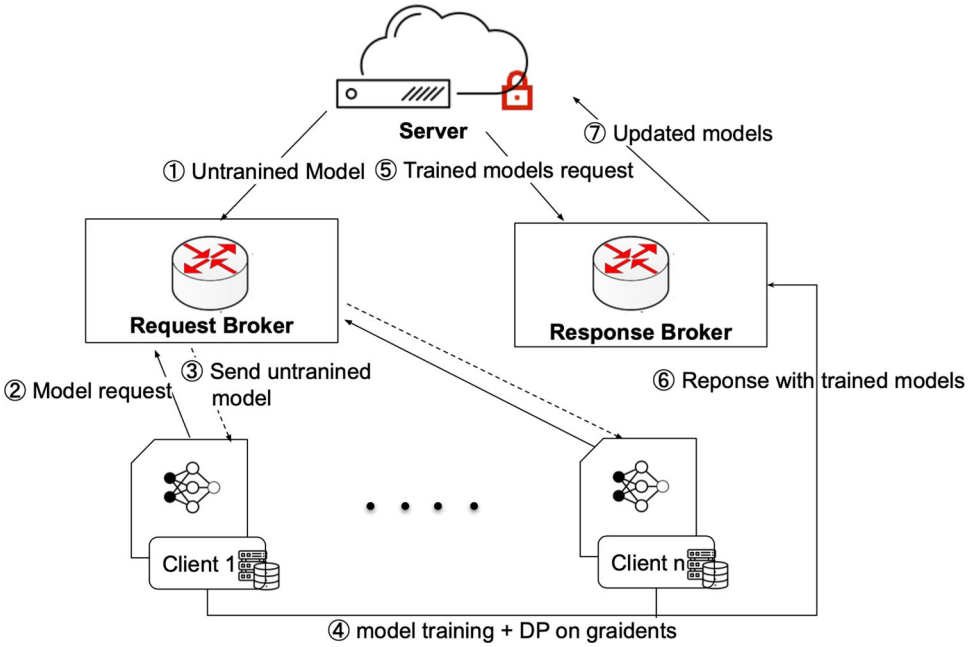

Fig. 3 Overview framework of the FedlabX

during the verification can be adopted by the central broker. Each client will train the model based on their local dataset. Once the server requests the trained model, clients will respond to the response broker in Kafka with the trained and masked model. Finally, the response broker will send the updated models to the server, and the server will conduct the secure aggregation algorithm to get the final result. This framework is ideal for cross-silos FL scenarios with few clients but large databases. It is more practical in the real industry with the Kafka client–server communication structure and message storage and transmission function. Meanwhile, from the security perspective, secure aggregation and DP schemes make FedlabX privacy-preserving and prevent most attacks when the server and clients are semi-honest. We implement the variant of secure aggregation, which makes it more efficient and computation cost-saving.

This scheme contains the following phases: setup and key generations, model updates, and secure aggregation. The main structure can be found in Fig. 4, and we will go for details of each phase in the following protocol.

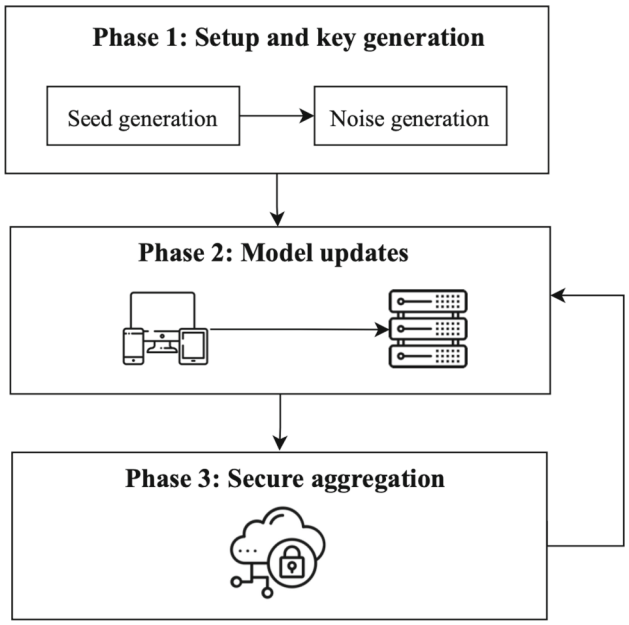

Fig. 4 Structure of the phases

Phase 1 (Setup and key generations)

This phase includes the key generation between the clients. Let the set of clients be $C = 1, 2, \ldots, n$. Then the server chooses an appropriate group G, where the computational Diffie–Hellman problem is hard. G will be a cyclic group of q elements with generator g, following general parameters will be generated:

$$(G,), |G| = q, g \in G : G =< g >$$

Let $i \neq j \in C$ be a fixed pair of clients. Then the following protocol that includes seed and noise generation must be run for every pair of clients:

Seed generation

1. Client i chooses an exponent $a_i \in_R \mathbb{Z}_q^*$, and sends to the server with the value of g^{a_i}. Then the server sends the pair (i, g^{a_i}) to the client j.

2. Client j chooses an exponent $a_j \in_R \mathbb{Z}_q^*$ and sends to the server with the value of g^{a_j}. Then the server sends the pair (j, g^{a_j}) to the client i.
3. The common key of client i and j is $key_{i,j} = g^{a_i a_j}$. At the end of this phase, every client $i \in C$ stores a set of common keys with other clients:

$$K_i = \text{key}_{i,1}, \text{key}_{i,2}, \ldots, \text{key}_{i,n}.$$

Noise generation

To prevent interference attacks, clients will implement DP by adding noises to the gradients after the model training, which is based on DP-SDG [2]. In our case, we use the Laplace noise generation mechanism [11] of DP. The Laplace mechanism preserves $(\epsilon, 0)$-differential privacy or ϵ-differentially private.

Definition 8 Lets Q be an aggregated query. The results of Q is α differentially private if it is perturbed by a random variable $\mathcal{R} \sim \text{Lap}(\theta, \alpha)$, where θ is the mean, and α is a scale.

A random variable $\mathcal{R} \sim \text{Lap}(0, \Delta_Q/\alpha)$ follows a Laplace distribution if:

$$Pr(\mathcal{R} = r) = \frac{\alpha}{2\Delta_Q} e^{-|r|\alpha/\Delta_Q} \tag{4}$$

When we add a Laplacian noise to the gradients, we exploit the infinitely divisible property of the distribution, where the random variable $\mathcal{R}$ can be computed by summing up n of other random variables. We generate the partial noise using the following:

$$\mathcal{R} = \sqrt{\mathcal{B} \cdot \sum_{i=1}^{n} \mathcal{R}_i}, \text{ for } \mathcal{R}, \mathcal{R}_i \sim \text{Lap}(0, \alpha), \text{ and} \tag{5}$$

$$\mathcal{B} \sim \text{Beta}(1, n-1) \text{ is a single r.v.,} \tag{6}$$

Where $\Pr(\mathcal{B} = x) = (n-1)(1-x)^{n-2}$.

Phase 2 (model updates)

As a result of the execution of the first phase, where the seed and noise will be generated, all participating clients will be ready for the model masked with noise and key pairs. Continually, clients will update the model by responding to the request sent from the central server. Models will arrive and be stored in the Kafka brokers connected with Zookeeper. In this case, even if clients have different settings of hardware conditions and network bandwidth, this framework can finally reach synchronous communication.

Phase 3 (secure aggregation)

In this phase, the server computes the aggregated client data for a given update. Let $l \in \mathbb{N}$ be the length of the users' data of a new update. This length can be different for every update but fixed during a single update. Let $x_i \in \{0,1\}^l$ be the data of client i, and P_l be a pseudorandom generator of output size l.
The following protocol describes the phase 2 and phase 3 of the framework:

Protocol 1 *1: The server chooses a random salt* $r \in_R \{0,1\}^n$.
2: The server sends the chosen r *to every client* i.
3: Every client $i \in C$ *computes*

$$s_i = x_i + \sum_{j \le i} P_l(r|\text{key}_{i,j}) - \sum_{i \le j} P_l(r|\text{key}_{j,i})$$

4: Every client i *sends* s_i *to the server.*
5: The server computes $\sum_{i \in c} s_i$ *to get the aggregation of the individual data.*

The result of the $\sum_{i \in c} s_i$ in step 5 of Protocol 1 is our final result of the training request.

Security analysis

FedlabX shows its advantage in data privacy preservation. With a modified federated ML scheme structure, we focused on the privacy and security properties of this framework. In this section, we analyze the security of the proposed model. During analysis, we consider two types of threat models in our scheme, internal and external attacks. Table 1 briefly summarize the threats, source of vulnerability, and countermeasures in the FedlabX framework. We do not go into the details of attack prevention with DP and secure aggregation here, as other papers have proved them [12, 14, 21, 29].

Remark 1 (Data privacy) If the secure aggregation and Laplace noise generation have been implemented, then the proposed model satisfies the *data privacy* requirement, where the original data can be protected from the gradient leakage attacks and the inference attacks.

Below are a few arguments about this Remark 1. We assume that the central server and clients are semi-honest in the proposed model. In this case, they will follow the protocol properly but try to exploit and infer useful information from the training process for their purpose. Therefore, for an insider attack, the data reconstruction from the gradients is the leading security issue. In this attack, the semi-honest nodes get the gradient update during iterative training to reconstruct the local training data using leaked gradients.

Table 1 Summarize of threats and countermeasures in FedlabX framework

	Threats	Source of vulnerability	Counter measures
Internal	Gradient-leakage	Semi-honest server	Secure aggregation
	Inference attack	Semi-honest client semi-honest server	Differential privacy
External	Man in the middle	Insecure channel	Kafka
	Sybil attacks	Malicious outside users	Kafka

Deep leakage from gradient (DLG) [42] and improved deep leakage (iDLG) [41]) use different methods. That is, they run optimization on the pixels of a randomly generated image (or text) matching the gradients on the random image to those of the real data point. If the training uses frequent updates computed from gradients over few data points or a small number of epochs, the training data can be reconstructed completely. As FedlabX uses secure aggregation, this scheme prevents gradient-leakage attacks [6, 13, 37].

External attacks such as man-in-the-middle-attack, Sybil attacks, and interference attacks happen when attackers from the outside of the architecture want to hack the system to steal some data or crash the protocol. The man-in-the-middle attack is one of the most typical attacks. An attacker sniffs the traffics between clients and the central server and tries to infer the private information of clients. However, as the proposed model assumes the existence of secure communication channels [7, 17], we do not consider man-in-the-middle attacks in our scheme. The main security issue, in this case, is the interference attack. In an interference attack, an attacker trains shadow models similar to the attacked one, which performs a similar task over similar datasets. Different input data feed these malicious shadow models, and over their outputs, a binary classifier attack model will be trained whose task is to decide whether the input is part of the training data. Despite its complexity trade-off [33], DP can prevent membership interference attacks as proved in [8, 27, 28].

Theorem 3 *The proposed model is computationally correct by following the protocol honestly.*

Proof based on Definition 5, the proposed model is correct if the server conducts aggregation without considering the accuracy loss. In other words, the central server in Protocol 1 should finally compute $\sum_{i\in c} x_i$. Each client i computes $s_i = x_i + \sum_{j\le i} P_l(r|\text{key}_{i,j}) - \sum_{i\le j} P_l(r|\text{key}_{j,i})$. Since the pseudo-random generator P takes the $r|\text{key}_{i,j}$ seed from each pair of clients (as a result of phase 1 of the proposed protocol), in each pair of clients, the same random number will be generated. Considering that the central server will receive these two random numbers in two opposite signs, they will cancel each other after adding them. As a result, only x_i values of each client will remain, therefore

$$\sum_{i\in c} x_i + \sum_{j\le i} P_l(r|\text{key}_{i,j}) - \sum_{i\le j} P_l(r|\text{key}_{j,i}) = \sum_{i\in c} x_i \qquad \square$$

Theorem 4 *If the function P_l is PRNG in phase 3 of the protocol is, then the Protocol 1 satisfies client unlinkability.*

Proof Assume that the function P_l is PRNG in phase 3 of the Protocol 1. A PPT adversary party can access the securely aggregated client data s_i from step 4 of Protocol 1. To link the client identity to the client data, an adversary needs x_i that is blinded with

$$\sum_{j\le i} P_l(r|\text{key}_{i,j}) - \sum_{i\le j} P_l(r|\text{key}_{j,i})$$

As a result of the PRNG with a secure seed concatenated random salt r, the above blinding factor seems random from any PPT adversary point of view, hence the adversary can learn x_i from aggregated s_i with negligible probability only. $\square$

Evaluations

In this section, we compare Kafka-based privacy preserving Federated Learning (FedlabX) with other typical frameworks we mentioned before, Flower, FATE, TFF, and FedML, regarding privacy mechanisms, system scalability and interoperability and the computational complexity. Furthermore, we also evaluate the performance of the proposed FedlabX framework in terms of computational cost and accuracy.

Comparisons of different FL frameworks

Comparisons of privacy mechanisms

We list the privacy mechanisms of the most popular FL frameworks in Table 2.

The comparison shows that the FATE framework applies different cryptographic mechanisms compared to other frameworks. FATE implements secure computation protocols based on homomorphic encryption (HE) of the Pallier system and multi-party computation (MPC). It provides strong privacy protection but relies on heavy computation. The Flower is one of the most user-friendly FL libraries among these open-access frameworks, and it provides DP for privacy-preserving. From the official website, we can find out that their "LightSecAgg protocol," which is for secure aggre-

Table 2 Comparisons of different FL frameworks

Frameworks	Privacy mechanisms				
	SA	DP	HE	MPC	AA
FATE 1.5.0	✓		✓	✓	
Flower		✓			
TFF 0.17.0	✓	✓			
FedML		✓			
FedlabX	✓	✓			✓

SA secure aggregation, *DP* differential privacy, *HE* homomorphic encryption, *MPC* multi-party computation, *AA* anonymous authentication

gation, has not been implemented yet, so its diagram and abstraction may not be accurate in practice. Same for other mechanisms. TFF applies very similar strategies to our framework in implementing cryptographic mechanisms with SA and DP schemes. FedML only provides DP, which can guarantee a certain level of privacy. However, it can not solve the gradient leakage risks in FL.

Incredibly, only our framework reaches anonymous authentication. It is not allowed to use anti-detoxification and anomaly detection to user data or proposed updates in the FL settings. That is why FL has no defense against data poisoning and no anomaly detection. Most other schemes apply standard security communication channels to exchange messages, but they do not mention how the clients are authenticated to prevent malicious clients. However, in our framework, it is clear that we reach anonymous authentication with Kafka ACLs and 1 out of n attribute verification.

Comparisons of system scalability and interoperability

The comparisons of system scalability and interoperability of different frameworks can be found in Table 3.

As observed, FATE has the capability to support various data structures, including HFL, VFL, and Hybrid FL, whereas other frameworks may only support HFL. Additionally, FATE offers a comprehensive one-step federation solution that encompasses assessment and evaluation. This aspect enhances its operability while potentially impacting its scalability. Flower is predominantly used in stand-alone cases, which can result in average operability and scalability. TFF focuses on edge computing and emphasizes ease of cooperation with other platforms and structures. However, the complexity in operating TFF may arise due to its reliance on command-line operations rather than a graphical user interface. FedML demonstrates its versatility in various applications, including topology-based mode, stand-alone mode, edge computing, and non-IID case. Its user interface and API offer ease of use, allowing for straightforward implementation. However, extending the framework to integrate with other platforms can be challenging. FedLabX supports both stand-alone mode and edge computing. It facilitates seamless collaboration with different strategies and platforms, offering enhanced interoperability. However, its operability is rated as average due to the absence of a user interface.

Comparisons of computational complexity

All calculations below assume a single server and n users, where each user holds a data vector of size m. The computational complexity of the FL libraries is influenced by various factors, including the size and complexity of the data and models, the algorithms and protocols used, and the efficiency optimizations employed within the library. However, here we only discuss the computational complexity, which is increased by the privacy mechanisms, especially SA, DP, and HE strategies. We analyze the computational complexity of FedlabX in the following.

On the client side

Compared to the basic FL, the main differences in computational cost lie in the key generation at the initialization stage, PRNG function for the secure aggregation, and the Laplacian noise generation for the DP. We can divide the computation cost of each client i into three parts. (1) Performing the $n-1$ key agreements, which takes $O(n-1)$ computation time, (2) Generating noise $P_l(r|\text{key}_{i,j})$ with the PRNG function for every other user j, which takes $O(mn)$ time in total, (3) Generating local noise with the Laplace mechanism, which takes $O(mn)$ computation time. Overall, the computation cost on a standard client is $O(mn+n)$.

On the server side

There is no key set up on the server side. Thus, we only need to consider the computation cost during the federated process. The server's computation cost can be broken down as: performing the aggregation function by removing the $P_l(r|\text{key}_{i,j})$ values, which takes time $0(mn^2)$ in the worst case.

The computational complexity of different FL frameworks is summarized in Table 4.

The most significant computation cost of FATE lies in the HE, which is mainly determined by the encryption and decryption of the message size. For each client, it encrypts the predicted value and sends the resulting ciphertext to the server. Therefore, the computational complexity for each client is $O(m^2)$, where m is the size of the message and m is much larger than the client number n. The computational complexity of the FATE is the highest one. On the server side,

Table 3 System scalability and interoperability comparisons of different FL frameworks

Framework	Structures	Applications	Operability	Scalability
FATE 1.5.0	HFL, VFL, Hybrid FL	One-step federation solution Assessment, evaluation...	High	Low
Flower	HFL	Stand-alone mode	Average	Average
TFF 0.17.0	HFL	Edge computing	Low	High
FedML	HFL	Topology-based mode Stand-alone mode Edge computing Non-IID	High	Low
FedLabX	HFL	Stand-alone mode Edge computing	Average	High

Table 4 Comparisons of computational complexity of different FL frameworks

Frameworks	Computational complexity	
	Client side	Server side
FATE 1.5.0	$O(m^2)$	$O(mn^2)$
TFF 0.17.0	$O(mn+n)$	$O(mn^2)$
Flower & FedML	$O(m)$	$O(m)$
FedlabX	$O(mn+n)$	$O(mn^2)$

calculations are performed based on the ciphertext, resulting in a computational complexity of $O(m)$. However, the SA strategy is also employed, which increases the computational complexity to $O(mn^2)$. In conclusion, the overall computational complexity is $O(mn^2)$.

The Flower library and FedML have SAME complexity since they both utilize the DP strategy. On the client side, the computation complexity is determined by the Laplace mechanism, which has a complexity of $O(m)$. Similarly, on the server side, where aggregation is performed, the computation complexity is also based on the message size, resulting in a complexity of $O(m)$ as well. Thus, both the client and server components exhibit a computational complexity of $O(m)$. TFF and FedlabX employ the same privacy strategies, resulting in comparable complexity on both the client and server sides.

Prototype performance

In this sub-section, we first clarify the hardware and public datasets for prototype evaluation. Then, the accuracy results with DP and without DP are shown and explained.

Hardware: Both the central server and clients have the same hardware settings in this experiment. All experiments were executed on 2x Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz, 1 A100 GPU, and 256 GB of RAM. Every component was run in Docker containers, and resources were allocated to them dynamically. One container was dedicated to MongoDB for storing the trained models, and another container served as the message queue. These containers had minimal resource usage, except for disk I/O. Accuracy testing was performed separately using a Python script. The experiment was designed in a way that the clients' VRAM usage was slightly below 4GB, ensuring that the 40GB GPU was not bottlenecked when using 10 clients.

Dataset: We use standard MNIST [25] and CIFAR-10 [23] datasets to conduct experiments. MNIST is a handwritten digital recondition dataset consisting of 60,000 training examples and 10,000 testing examples, and each is a 28 × 28 size gray-level image. MNIST is divided into four sub-datasets for FL, where the size of each sub-dataset is 100 samples. Compared to the MNIST, CIFAR-10 has a more complicated combination consisting of 60,000 32 × 32 color images in 10 classes, with 6000 images per class.The MNIST and CIFAR-10 datasets are widely used in machine-learning for tasks such as image classification. While these datasets do not contain personally identifiable information (PII), they can still be subject to specific privacy attacks. One of the most severe attacks is membership inference, where an attacker may attempt to infer whether a specific data point was part of the original dataset [31]. Meanwhile, although MNIST and CIFAR-10 datasets do not directly contain PII, it is possible to re-identify individuals by combining the dataset with external information. There is a high risk that an attacker may attempt to re-identify those individuals based on their unique characteristics.

Accuracy results without DP: We take the FedAvg algorithm 1 in MNIST and CIFAR-10 databases with IID and non-IID data structures as our baseline. Table 5 shows the accuracy of centralized learning and FL on MNIST and CIFAR-10 databases. The mode details results can be found in Fig. 5. It plots the accuracy of the FedlabX algorithm in the IID and non-IID data structure without the added DP. The training sets are evenly partitioned into ten clients. Each client is randomly assigned a uniform distribution over ten classes for the IID setting. For the non-IID setting, the data is sorted by class and divided to create an extreme 1-class non-IID, where each client receives data partition from only a

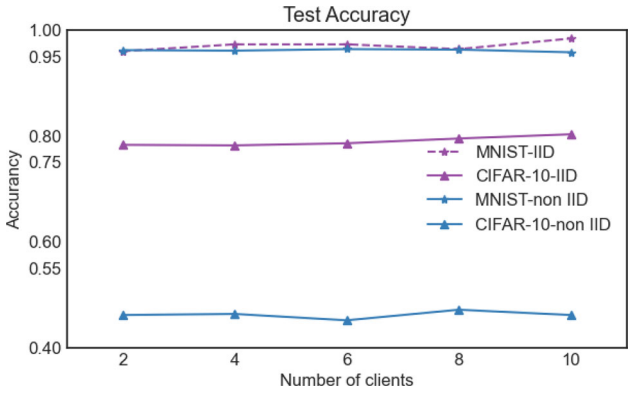

Fig. 5 Test accuracy of FedlabX framework

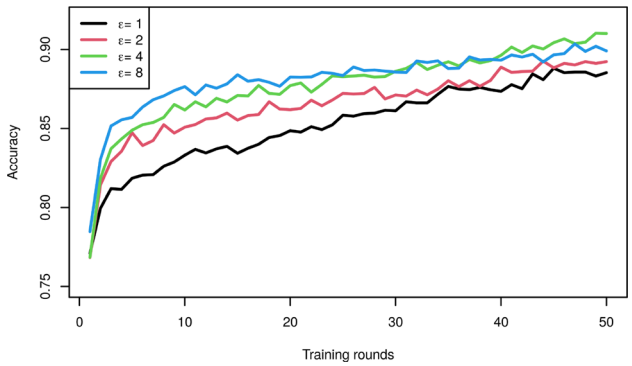

Fig. 6 Accuracy of different levels of ϵ setting with different rounds of training on MNIST

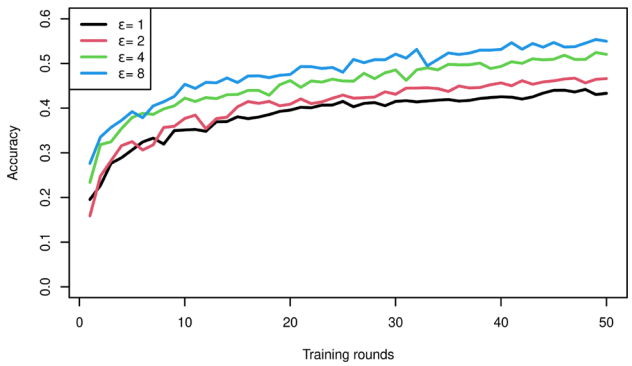

Fig. 7 Accuracy of different levels of ϵ setting with different rounds of training on CIFAR-10

Table 5 Baseline accuracy

	MNIST (%)	CIFAR-10 (%)
Centralized learning	98.2	89.3
Federated learning (iid)	98.4	80.3
Federated learning (non-iid)	96.4	47.2

single class. We perform 50 communication rounds of training on MNIST and CIFAR-10. We use the same notations for the FedAvg algorithm as Algorithm 1: B, the batch size, and E, the number of local epochs. The following parameters are used for FedlabX framework: for MNIST, $B = 32$, $E = 25$, $\eta = 0.01$ and decay rate = 0.995; for CIFAR-10, $B = 128$, $E = 60$, $\eta = 0.1$ and decay rate = 0.992.

The following observations can be made: the accuracy on the MNIST dataset can reach even higher than the centralized accuracy in the IID data structure. There is a slight drop in the non-IID case on MNIST with 2%. However, on the CIFAR-10 database, the accuracy reaches around 80% in IID, and drops sharply in the non-IID scenario, which can only reach almost 50% of accuracy.

Accuracy with DP: From the definition of DP (Definition 3), we conclude that the parameter ϵ measures the ability of the random algorithm $\mathcal{A}$ to resist attacks, and the smaller the parameter ϵ, the greater the privacy protection it provides. This is because the smaller the ϵ is, the bigger Laplace noise is, i.e., $b = \frac{\Delta q}{\epsilon}$. In this case, the accuracy will increase with the growth of ϵ value. In this experiment, we fix the parameter of θ to 10^{-5}. In most of the literature, the algorithms have been evaluated with ϵ ranging from as little as 0.1 to as much as 8. In our case, we analyze the accuracy of MNIST and CIFAR-10 with the ϵ value from 1 to 8. The main results can be found in Figs. 6 and 7.

On the MNIST database, there are a few differences with the increase of ϵ value. The highest accuracy ranges from 89 to 91%. However, with more training rounds, the performance on $\epsilon = 4$ is better than when the $\epsilon = 8$. It reflects that with higher epsilon, there is a need for as many training rounds. Most clearly, this effect is observable around $\epsilon = 1$. This plot Fig. 6 suggests we have to focus more on sub-zero and $\epsilon = 1$–2 ranges.

On the CIFAR-10 database, there is a significant drop after adding the noise to the model gradients with DP. When the ϵ equals 1, the best accuracy only reaches around 44%. With the increase of ϵ, the accuracy also improves. The details can be found in Fig. 7.

We examined the client training loss with respect to the number of epochs. The training loss serves as a monitoring metric for assessing the progress of local training on each client and evaluating the convergence and performance of models across different clients. In our case, we fixed the number of clients to 4 and averaged all the training losses to obtain the results. As observed, a larger ϵ value of DP leads to more loss, particularly when the number of epochs is small. Additionally, the proposed framework suffers from less training loss on MNIST than on CIFAR-10 due to its smaller size. However, the framework converges on both datasets as the number of epochs increases. More details can be found in Fig. 8.

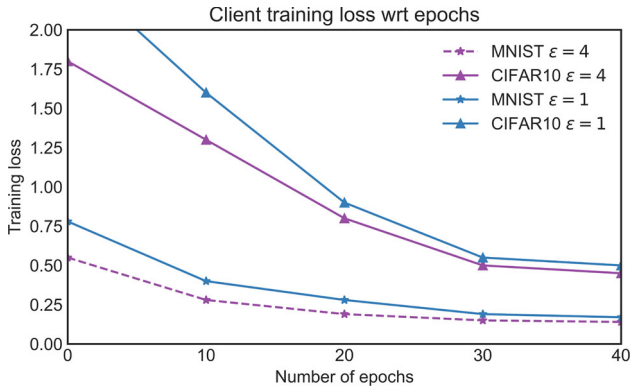

Fig. 8 Client training loss over the epochs

Comparisons with some state-of-the-art FL frameworks

Table.6 depicts the accuracy of various differential private training methods on MINIST and CIFAR-10 databases with different privacy losses. LDP-FL [32] adds noises directly into the raw database, which significantly drops the accuracy in CIFAR-10. For a more rigorous comparison, we have included FedAvg-LDP and FedAvg-CDP [39], two state-of-the-art algorithms in the comparison. The difference is that LDP adds noise directly into the database, while CDP adds noise after the model aggregation. These two algorithms have a structure that closely resembles our framework, making them more suitable for comparative analysis. We also include the version of DP-SGD [4], which modifies the activation functions to prevent activations from exploding in differential privacy. DP-SGD applies DP in calculating the gradients instead of adding noise directly to the model or the dataset, which leads to better accuracy results. ρ-zero-centralized DP-SGD [40] gets more attention and is also applied in the 2020 Census Conflict of interest Avoidance System for higher privacy. However, the accuracy is not satisfied even in the MNIST database. In our approach, FedlabX can not reach the best accuracy but can reach the median.

Limitations of FedlabX

Based on the analysis and evaluations above, it is apparent that the FedlabX framework still has certain limitations that can be attributed to the cryptographic components adopted in this work.

1. **Size limitation and latency of Kakfa**: Kafka configuration limits the size of communication messages, which is 1MB by default. However, the model we trained in the FL framework might exceed this limitation. To solve this problem, we can modify the Kafka broker configuration file parameter to increase the allowed maximum message size. Still, processing large messages consumes more CPU and memory of our producer and consumer, which might cause high latency to the end-user.
2. **Computation cost**: As we can found the comparison of the computational complexity in Table 4, the secure aggregation protocol, especially the key agreement process and the PRNG generation process, increases the computational complexity of our framework.
3. **Accuracy drop**: The choice of a DP scheme will undoubtedly have an impact on the accuracy of the framework. This is evident from the results presented in Table 6, where the accuracy consistently decreases as more noise is added to the algorithm.

Conclusion and future work

This paper proposed a practical privacy-preserving framework FedlabX based on the basic FL. We proposed adopting secure aggregation and differential privacy mechanisms to achieve a privacy-preserving scheme. Meanwhile, Kafka supports anonymous authentication and asynchronous message queue. Our security analysis proves that FedlabX can protect from the semi-honest server and semi-honest clients, which is enough for a cross-silos FL scenario. From the experiment results, the FedlabX accuracy is higher than the LDP-FL and centralized DP-FL. However, some improve-

Table 6 Test accuracy of various differentially private training methods on MNIST and CIFAR-10 with different privacy loss and $\theta = 10^{-5}$

DP algorithms	MNIST			CIFAR-10		
	$\epsilon = 1$	$\epsilon = 3$	$\epsilon = 8$	$\epsilon = 1$	$\epsilon = 3$	$\epsilon = 8$
LDP-FL	78.76%	89.10%	NA	27.18%	28.00%	NA
FedAvg-CDP	NA	69.20%	82.00%	NA	51.00%	71.00%
FedAvg-LDP	NA	71.10%	92.00%	NA	52.00%	72.00%
DP-SGD	89.00%	89.40%	**92.00%**	NA	**68.00%**	**73.00%**
ρ-zCDP-SGD	10.12%	65.33%	NA	NA	NA	NA
FedlabX	**88.97%**	**89.23%**	90.35%	44.01%	48.70%	55.72%

Bold represent the best accuracy results in each situation

ments can still be made compared to TensorFlow DP-SGD. The following future work are worthy of considering:

1. How can we set a reasonable ϵ with an explanation or justification? If the relationship between the parameter ϵ and the probability of the attacker's success can be obtained, it will undoubtedly give an intuitive understanding of the strength of differential privacy protection.
2. Based on the current scheme, we can add more privacy-preserving features and comprehensively analyze different schemes, such as MPC and HE schemes.
3. Non-IID impacts the accuracy massively in FL. Another important research direction is how we can improve the accuracy of our FedlabX framework.

Funding Open access funding provided by Eötvös Loránd University. Funding was provided by EIT Digital (2021-1.2.1-EIT-KIC) and Alexander von Humboldt-Stiftung. This research was supported in part by the project No. 2019-1.3.1-KK-2019-00011 financed by the National Research, Development and Innovation Fund of Hungary under the Establishment of Competence Centers, Development of Research Infrastructure Programme funding scheme.

References

1. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, et al (2016) {TensorFlow}: a system for {Large-Scale} machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI 16). pp. 265–283
2. Abadi M, Chu A, Goodfellow I, McMahan HB, Mironov I, Talwar K, Zhang L (2016) Deep learning with differential privacy. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. ACM, pp 308–318
3. Annas GJ (2003) Hipaa regulations: a new era of medical-record privacy? N Engl J Med 348:1486
4. Arachchige PCM, Bertok P, Khalil I, Liu D, Camtepe S, Atiquzzaman M (2019) Local differential privacy for deep learning. IEEE Internet Things J 7(7):5827–5842
5. Beutel DJ, Topal T, Mathur A, Qiu X, Parcollet T, de Gusmão PP, Lane ND (2020) Flower: a friendly federated learning research framework. arXiv preprint arXiv:2007.14390
6. Bonawitz K, Ivanov V, Kreuter B, Marcedone A, McMahan HB, Patel S, Ramage D, Segal A, Seth K (2017) Practical secure aggregation for privacy-preserving machine learning. In: proceedings of the 2017 ACM SIGSAC conference on computer and communications security. pp 1175–1191
7. Buchmann J, Karatsiolis E, Wiesmaier A, Karatsiolis E (2013) Introduction to public key infrastructures, vol 36. Springer, Berlin
8. Chen J, Wang WH, Shi X (2020) Differential privacy protection against membership inference attack on machine learning for genomic data. In: BIOCOMPUTING 2021: proceedings of the pacific symposium. World Scientific, pp 26–37
9. Diffie W, Hellman M (1976) New directions in cryptography. IEEE Trans Inf Theory 22(6):644–654
10. Dwork C (2008) Differential privacy: a survey of results. In: International conference on theory and applications of models of computation. Springer, pp 1–19
11. Dwork C, McSherry F, Nissim K, Smith A (2006) Calibrating noise to sensitivity in private data analysis. In: Halevi S, Rabin T (eds) Theory of cryptography. Springer, Berlin, pp 265–284
12. Dwork C, Roth A et al (2014) The algorithmic foundations of differential privacy. Foundations and Trends®. Theor Comput Sci 9(3—–4):211–407
13. Elkordy AR, Zhang J, Ezzeldin YH, Psounis K, Avestimehr S (2022) How much privacy does federated learning with secure aggregation guarantee? arXiv preprint arXiv:2208.02304
14. Goryczka S, Xiong L (2015) A comprehensive comparison of multiparty secure additions with differential privacy. IEEE Trans Dependable Secur Comput 14(5):463–477
15. He C, Li S, So J, Zeng X, Zhang M, Wang H, Wang X, Vepakomma P, Singh A, Qiu H et al (2020) Fedml: a research library and benchmark for federated machine learning. arXiv preprint arXiv:2007.13518
16. Jin X, Chen PY, Hsu CY, Yu CM, Chen T (2021) Cafe: catastrophic data leakage in vertical federated learning. Adv Neural Inf Process Syst 34:994–1006
17. Kamel MBM, George LE (2016) Secure model for SMS exchange over GSM. Int J Comput Netw Inf Secur 8(1):1
18. Kamel MBM, Yan Y, Ligeti P, Reich C (2022) Attribute verifier in internet of things. In: 2022 32nd international telecommunication networks and applications conference (ITNAC). IEEE, pp 1–3
19. Kamel MB, Ligeti P, Reich C (2022) Odabe: outsourced decentralized cp-abe in internet of things. In: Applied cryptography and network security workshops: ACNS 2022 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, SiMLA, Rome, Italy, June 20–23, 2022, Proceedings. Springer, pp 611–615
20. Kamel MB, Ligeti P, Reich C (2022) Sdabe: efficient encryption in decentralized cp-abe using secret sharing. In: 2022 International conference on electrical, computer and energy technologies (ICECET). IEEE, pp 1–6
21. Kerkouche R (2021) Differentially private federated learning for bandwidth and energy constrained environments. Ph.D. thesis, Université Grenoble Alpes [2020-....]
22. Kreps J, Narkhede N, Rao J et al (2011) Kafka: a distributed messaging system for log processing. In: Proceedings of the NetDB. Athens, Greece vol. 11. pp 1–7
23. Krizhevsky A, Hinton G et al (2009) Learning multiple layers of features from tiny images. Toronto, ON, Canada
24. Lacharité MS, Minaud B, Paterson KG (2018) Improved reconstruction attacks on encrypted data using range query leakage. In: 2018 IEEE symposium on security and privacy (SP). IEEE, pp 297–314
25. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
26. McMahan HB, Moore E, Ramage D, Arcas BA (2016) Federated learning of deep networks using model averaging. arXiv preprint arXiv:1602.05629 **2**

27. Rahimian S, Orekondy T, Fritz M (2021) Differential privacy defenses and sampling attacks for membership inference. In: Proceedings of the 14th ACM workshop on artificial intelligence and security. pp 193–202
28. Rahman MA, Rahman T, Laganière R, Mohammed N, Wang Y (2018) Membership inference attack against differentially private deep learning model. Trans Data Priv 11(1):61–79
29. Segal A, Marcedone A, Kreuter B, Ramage D, McMahan HB, Seth K, Bonawitz K, Patel S, Ivanov V (2017) Practical secure aggregation for privacy-preserving machine learning. In: proceedings of the 2017 ACM SIGSAC conference on computer and communications security. pp 1175–1191
30. Shamir A (1979) How to share a secret. Commun ACM 22(11):612–613
31. Shokri R, Stronati M, Song C, Shmatikov V (2017) Membership inference attacks against machine learning models. In: 2017 IEEE symposium on security and privacy (SP). IEEE, pp 3–18
32. Sun L, Qian J, Chen X (2020) Ldp-fl: practical private aggregation in federated learning with local differential privacy. arXiv preprint arXiv:2007.15789
33. Truex S, Liu L, Gursoy ME, Wei W, Yu L (2019) Effects of differential privacy and data skewness on membership inference vulnerability. In: 2019 First IEEE international conference on trust, privacy and security in intelligent systems and applications (TPS-ISA). IEEE, pp 82–91
34. Voigt P, Von dem Bussche A (2017) The eu general data protection regulation (gdpr). A practical guide, vol 10(3152676), 1st edn. Springer International Publishing, Cham, pp 10–5555
35. Wang J, Liu Q, Liang H, Joshi G, Poor HV (2020) Tackling the objective inconsistency problem in heterogeneous federated optimization. Adv Neural Inf Process Syst 33:7611–7623
36. Wang Z, Song M, Zhang Z, Song Y, Wang Q, Qi H (2019) Beyond inferring class representatives: User-level privacy leakage from federated learning. In: IEEE INFOCOM 2019-IEEE conference on computer communications. IEEE, pp 2512–2520
37. Yang CS, So J, He C, Li S, Yu Q, Avestimehr S (2021) Lightsecagg: Rethinking secure aggregation in federated learning. arXiv preprint arXiv:2109.14236
38. Yang Q, Liu Y, Chen T, Tong Y (2019) Federated machine learning: concept and applications. ACM Trans Intell Syst Technol (TIST) 10(2):1–19
39. Yang Y, Hui B, Yuan H, Gong N, Cao Y Privatefl: accurate, differentially private federated learning via personalized data transformation (2013) PrivateFL: accurate, differentially private federated learning via personalized data transformation
40. Yu L, Liu L, Pu C, Gursoy ME, Truex S (2019) Differentially private model publishing for deep learning. In: 2019 IEEE symposium on security and privacy (SP). IEEE, pp 332–349
41. Zhao B, Mopuri KR, Bilen H (2020) idlg: improved deep leakage from gradients. arXiv preprint arXiv:2001.02610
42. Zhu L, Liu Z, Han S (2019) Deep leakage from gradients. arxiv:1906.08935