AegisFL: Efficient and Flexible Privacy-Preserving Byzantine-Robust Cross-silo Federated Learning

Dong Chen [1 2] **Hongyuan Qu** [1 2] **Guangwu Xu** [1 2 3 4]

Abstract

Privacy attacks and poisoning attacks are two of the thorniest problems in federated learning (FL). Homomorphic encryption (HE), which allows certain mathematical operations to be done in the ciphertext state, provides a way to solve these two problems simultaneously. However, existing Paillier-based and CKKS-based privacy-preserving byzantine-robust FL (PBFL) solutions not only suffer from low efficiency but also expose the final model to the server. Additionally, these methods are limited to one robust aggregation algorithm (AGR) and are therefore vulnerable to AGR-tailored poisoning attacks. In this paper, we present AegisFL, an efficient PBFL system that provides the flexibility to change the AGR. We first observe that the core of the existing advanced AGRs is to calculate the inner products, L_2 norms and mean values for vectors. Based on this observation, we tailor a packing scheme for PBFL, which fits perfectly with RLWE-based fully homomorphic encryption. Under this packing scheme, the server only needs to perform one ciphertext multiplication to construct any required AGR, while the global model only belongs to honest clients. Finally, we conduct extensive experiments on different datasets and adversary settings, which also confirm the effectiveness and efficiency of our scheme.

[1]School of Cyber Science and Technology, Shandong University, Qingdao, China [2]Key Laboratory of Cryptologic Technology and Information Security of Ministry of Education, Shandong University, Qingdao, China [3]Shandong Institute of Blockchain, Jinan, China [4]Quan Cheng Laboratory, Jinan, China. Correspondence to: Guangwu Xu <gxu4sdq@sdu.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

Federated learning (FL) (McMahan et al., 2017) is an emerging machine learning paradigm that was proposed to solve the data privacy problem when multiple clients jointly train machine learning models. However, there are two fatal shortcomings in traditional FL. First, a large number of studies (Nasr et al., 2019; Geiping et al., 2020; Zhu et al., 2019; Zhang et al., 2021) have shown that the server can launch privacy attacks based on intermediate results uploaded by clients, thereby recovering some sensitive information of clients, which makes federated learning meaningless. Second, FL is vulnerable to poisoning attacks that some malicious clients may upload manipulated intermediate results to the server (Fang et al., 2020; Shejwalkar & Houmansadr, 2021; Baruch et al., 2019; Bagdasaryan et al., 2020; Wang et al., 2020). Even a single malformed intermediate result can significantly alter the final model in FL training (Roy Chowdhury et al., 2022). As a result, the final model will have poor performance or be left backdoors by malicious clients.

Unfortunately, mitigating both privacy attacks and poisoning attacks simultaneously is a very challenging problem: in general, to defend against privacy attacks, clients' intermediate results will be encrypted or masked, which makes the intermediate results invisible to the server. To guard against poisoning attacks, using AGR on the server is an effective method because AGR can filter out abnormal intermediate results. However, this method requires the server to access the clients' intermediate results and perform comprehensive similarity measurements in plaintext. Therefore, they are mutually orthogonal aspects.

Homomorphic encryption not only provides strong privacy guarantees, but also allows certain mathematical operations (such as addition and multiplication) to be performed directly on encrypted data without prior decryption. When decrypted, the output is the same as that produced from plaintext data. Therefore, some works, such as Paillier-based solutions (Liu et al., 2021; Ma et al., 2022; Lu et al., 2023) and CKKS-based solutions (Miao et al., 2022; Rahulamathavan et al., 2023), explore using HE as a privacy protection method to build privacy-preserving byzantine-robust FL (PBFL), which can simultaneously solve privacy

Schemes	Privacy-preserving technology	Similarity measurement method	Efficiency	Ownership of the final model
PEFL (Liu et al., 2021)	Paillier	Pearson correlation coefficient	Low	Clients
ShieldFL (Ma et al., 2022)	Paillier	Cosine similarity	Low	Clients
Scheme (Lu et al., 2023)	Paillier	Adjusted cosine similarity	Low	Clients and Server
Scheme (Miao et al., 2022)	CKKS	Cosine similarity	Moderate	Clients and Server
FheFL (Rahulamathavan et al., 2023)	CKKS	Euclidean distance	Moderate	Clients and Server
AegisFL (ours)	RLWE-based HE	Various	High	Clients

Table 1. Comparison between our scheme and previous schemes.

attacks and poisoning attacks. However, this leads to several serious problems when trying to use these schemes in practice, as shown below.

Firstly, heavy computational burden. When using Paillier (Liu et al., 2021; Ma et al., 2022; Lu et al., 2023), each element in an intermediate result needs to be quantified and encrypted one by one, and measuring the similarity between intermediate results in Paillier ciphertext also requires a lot of additional calculation and communication, which makes their methods very inefficient. When using CKKS (Miao et al., 2022; Rahulamathavan et al., 2023), thanks to the single-instruction-multiple-data (SIMD), the efficiency is increased to a certain extent compared with the Paillier-based scheme. However, calculating the inner products and L_2 norms for N-dimensional vectors under CKKS ciphertext involves $\log_2 N$ rotations, which is very time-consuming. Furthermore, in (Miao et al., 2022), they use the ReLU function to realize their AGR. However, when evaluating in CKKS, an iteration algorithm is used to approximate the ReLU function, which requires a high multiplication depth (possibly requiring bootstrapping), severely degrading efficiency. **Secondly, lack of flexibility**. Existing PBFL solutions are all designed for a specific AGR. The AGR and privacy protection technology are bound together. It is difficult to change the AGR in the existing PBFL methods. However, due to the influence of factors such as the type of poisoning attack and the distribution of clients' data (maybe iid or non-iid), different AGRs may exhibit varying levels of effectiveness, so it is difficult to determine an optimal AGR. Even worse, if malicious clients know the adopted AGR, they can launch targeted advanced poisoning attacks so that the poisoned models completely bypass the detection of the adopted AGR. **Thirdly, the final model is exposed to the server**. In cross-silo FL, the final model should only be visible to the clients (Zhang et al., 2020; Kairouz et al., 2021). However, some schemes (Lu et al., 2023; Miao et al., 2022; Rahulamathavan et al., 2023) expose the final model to the server, which does not meet the requirements of cross-silo FL.

In this paper, we strive to build an efficient PBFL system that can flexibly change AGR while ensuring that the final model only belongs to honest clients. To achieve this goal, we take a closer look at the AGRs in existing PBFL and observe that almost all of them build upon three basic operations: *inner product, L_2 norm and mean value*. Based on this observation, we propose AegisFL, whose core idea is to use a special packing scheme to encode the intermediate results into polynomials, while protecting the polynomials with a RLWE-based HE. With the help of polynomial ring structure of HE scheme, only one polynomial modular multiplication is performed to get some meaningful results, such as the inner product of two vectors or the L_2 norm and mean value of a vector. Then, a variety of state-of-the-art AGRs can be flexibly constructed based on these results. In addition, in order to reduce the communication burden, we also compress the ciphertext without affecting the results. In a word, we tailor a HE-based privacy preserving scheme for PBFL, while retaining high efficiency and flexibility. TABLE 1 gives a comparison of AegisFL with the above-mentioned PBFL solutions.

The contributions of this paper can be summarized as follows:

- To the best of our knowledge, we propose the first PBFL scheme that can flexibly change the AGR.
- We use a special packing method that is perfectly compatible with RLWE-based HE and FL. In this packing method, the inner products, L_2 norms and mean values for vectors can be efficiently calculated.
- We have carried out our experiments on two dataset and three types of poisoning attacks, and these experiments show the high accuracy and efficiency of AegisFL.

2. Background

2.1. Federated Learning

In a standard cross-silo FL setting, with the help of a server, U clients $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_U\}$ aim to cooperatively train a global model $\boldsymbol{w}$ without leaking their own data. Each client has a local dataset $D_u, u \in [1, U]$, the training objective is $\text{argmin}_{\boldsymbol{w}} \sum_{u=1}^{U} \mathcal{L}_u(\boldsymbol{w}, D_u)$, where $\mathcal{L}_u(\boldsymbol{w}, D_u)$ is the loss for client u's local dataset.

At the beginning of each global iteration t, the server first sends the current global model $\boldsymbol{w}^{(t)}$ to each client. Secondly, after receiving the global model, each client $\mathcal{C}_u$ con-

ducts one or more local iterations based on its dataset to obtain a local model $\boldsymbol{w}_u^{(t)}$, then sends local model $\boldsymbol{w}_u^{(t)}$ or local model update $\boldsymbol{g}_u^{(t)} = \boldsymbol{w}^{(t)} - \boldsymbol{w}_u^{(t)}$ to the server. Finally, the server aggregates the intermediate results sent by all clients to obtain a new global model $\boldsymbol{w}^{(t+1)}$ according to a defined aggregation rule, such as FedAvg. Specifically, if clients send local models to the server, the new global model $\boldsymbol{w}^{(t+1)} = \frac{1}{U}\sum_{u=1}^{U} \boldsymbol{w}_u^{(t)}$. If clients send local model updates to the server, the new global model $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \boldsymbol{g}^{(t)}, \boldsymbol{g}^{(t)} = \frac{1}{U}\sum_{u=1}^{U} \boldsymbol{g}_u^{(t)}$ is global model update. The above process is repeated until the termination condition is reached.

2.2. Poisoning Attacks

Based on adversarial goals, poisoning attacks can also be divided into targeted poisoning attacks (Bagdasaryan et al., 2020; Xie et al., 2019; Wang et al., 2020) and untargeted poisoning attacks (Fang et al., 2020; Shejwalkar & Houmansadr, 2021; Baruch et al., 2019). Untargeted attacks, such as Krum attack (Fang et al., 2020) and Min-Max attack (Shejwalkar & Houmansadr, 2021), aim to corrupt the global model so that it makes wrong predictions indiscriminately for a large number of test examples, that is, the test error rate is high. Targeted attacks, such as Scaling attack (Bagdasaryan et al., 2020) and PGD attack (Wang et al., 2020), aim to destroy the integrity of the global model, resulting in incorrect predictions for specific input but maintaining correct predictions for other test samples.

2.3. Privacy Attacks

Privacy attacks refer to when the server gets the plaintext intermediate results sent by clients, the server can infer some sensitive information from them. In recent years, a variety of privacy attacks have been proposed, including member inference (Nasr et al., 2019), attribute inference (Geiping et al., 2020; Zhu et al., 2019), and distribution estimation (Zhang et al., 2021). It is even possible to reconstruct multiple independent input images from the average gradient (Geiping et al., 2020). Therefore, it is essential to protect the intermediate results.

3. Preliminaries

3.1. RLWE-based Homomorphic Encryption

Homomorphic encryption is a cryptographic technology that can perform arithmetic operations on ciphertext. In this paper, we use CKKS scheme (Cheon et al., 2017). We first define some parameters used in this scheme. Let $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ and $\mathbb{C}$ be the set of integers, rational numbers, real numbers and complex numbers respectively. For a power-of-2 N, let $\mathcal{R} = \mathbb{Z}[X]/\left(X^N+1\right)$, and $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$. Let L be the level of CKKS scheme, and $P \in \mathbb{Z}$ be the big integer used in the key switching stage. Let $Q_l = q_0 \cdot \Delta^l$ for $0 < l \leq L$. Let Δ be the scaling factor of CKKS scheme. Let $\chi_{\text{key}}, \chi_{\text{err}}, \chi_{\text{enc}}$ denote the small distributions over $\mathcal{R}$ for secret, error and encryption respectively. The following is a concrete description of CKKS scheme that we use.

- **KeyGen**. Sample a secret $s \leftarrow \chi_{\text{sec}}$, a random $a \leftarrow \mathcal{R}_{Q_L}$, and an error $e \leftarrow \chi_{\text{err}}$. Set secret key as $sk \leftarrow (1, s)$ and public key $pk \leftarrow (b, a) \in \mathcal{R}_{Q_L}^2$, where $b = -a \cdot s + e \pmod{Q_L}$. As for evaluation key, sample $a' \leftarrow \mathcal{R}_{P \cdot Q_L}$, $e' \leftarrow \chi_{\text{err}}$, set $evk \leftarrow (b', a')$, where $b' = -a' \cdot s + P \cdot s^2 + e' \pmod{P \cdot Q_L}$.

- **Encode**$(\boldsymbol{m})$. We do not use the packing scheme in CKKS. Here we first briefly describe our packing scheme, and Sect. 3.2 will give its details and functionality. For a vector $\boldsymbol{m} = [m_0, m_1, \ldots, m_{N-1}] \in \mathbb{R}^N$, we give the following two packing methods:
  - **pm1**$(\boldsymbol{m})$: Compute $m' = \sum_{i=0}^{N-1} m_i \cdot X^i \in \mathbb{R}[X]/(X^N+1)$, return an integer polynomial $m = \lfloor \Delta \cdot m' \rceil \in \mathcal{R}$.
  - **pm2**$(\boldsymbol{m})$: Compute $m' = -\sum_{i=0}^{N-1} m_i \cdot X^{N-i} \in \mathbb{R}[X]/(X^N+1)$, return an integer polynomial $m = \lfloor \Delta \cdot m' \rceil \in \mathcal{R}$.

- **Decode**(m). For $m \in \mathcal{R}$, compute $m' = m/\Delta = \sum_{i=0}^{N-1} m_i' \cdot X^i \in \mathbb{R}[X]/(X^N+1)$, return a vector $\boldsymbol{m} = [m_0', m_1', \ldots, m_{N-1}']$.

- **Enc**(m, pk). For $m \in \mathcal{R}$, sample $r \leftarrow \chi_{\text{enc}}$ and $e_1, e_2 \leftarrow \chi_{\text{err}}$. Then output $ct \leftarrow (r \cdot b + m + e_1, r \cdot a + e_2) \pmod{Q_L}$.

- **Dec**(ct, sk). For $ct = (c_1, c_2) \in \mathcal{R}_{Q_l}^2$, output $\tilde{m} \leftarrow c_1 + c_2 \cdot s \pmod{Q_l}$.

- **Add**$(ct; ct')$. For $ct, ct' \in \mathcal{R}_{Q_l}^2$, output $\hat{ct} \leftarrow ct + ct' \pmod{Q_l}$.

- **cAdd**(ct, m'). For $ct = (c_0, c_1) \in \mathcal{R}_{Q_l}^2$ and a plaintext polynomial $m' \in \mathcal{R}$, output $\hat{ct} \leftarrow (c_0 + m', c_1) \in \mathcal{R}_{Q_l}$.

- **Mul**(ct, ct', evk). For two level l ciphertexts $ct = (c_0, c_1)$, $ct' = (c_0', c_1')$, compute

$$\tilde{ct} = (d_0, d_1) + \lfloor P^{-1} \cdot d_2 \cdot evk \rceil \in \mathcal{R}_{Q_l}^2,$$

  where $(d_0, d_1, d_2) = (c_0 c_0', c_0 c_1' + c_1 c_0', c_1 c_1') \in \mathcal{R}_{Q_l}^3$, then return $\hat{ct} = \lfloor \Delta^{-1} \cdot \tilde{ct} \rceil \in \mathcal{R}_{Q_{l-1}}^2$.

- **cMul**$(ct, cons)$. For a level l ciphertext $ct = (c_0, c_1)$ and a constant $cons \in \mathbb{R}$, let $\tilde{ct} = (\lfloor cons \cdot \Delta \rceil \cdot c_0, \lfloor cons \cdot \Delta \rceil \cdot c_1) \in \mathcal{R}_{Q_l}^2$, then return $\hat{ct} = \lfloor \Delta^{-1} \cdot \tilde{ct} \rceil \in \mathcal{R}_{Q_{l-1}}^2$.

3.2. Practical Packing Scheme

In this section, we describe a practical encode scheme (Yasuda et al., 2014) that exploits the structure of polynomial rings to efficiently compute the L_2 norm and mean value of a vector, or the inner product of two vectors. We have introduced the packing algorithms **pm1** and **pm2** in Sect. 3.1.

Assume $\boldsymbol{a} = [a_0, a_1, \ldots, a_{N-1}] \in \mathbb{R}^N$, $p_{\boldsymbol{a}}^{(1)} \leftarrow$ **pm1**$(\boldsymbol{a}, \Delta)$ and $p_{\boldsymbol{a}}^{(2)} \leftarrow$ **pm2**$(\boldsymbol{a}, \Delta)$. Then:

$$\begin{aligned} p_{\boldsymbol{a}}^{(1)} \cdot p_{\boldsymbol{a}}^{(2)} &= \left\lfloor \sum_{i=0}^{N-1} \Delta a_i \cdot X^i \right\rceil \cdot \left\lfloor - \sum_{j=0}^{N-1} \Delta a_j \cdot X^{N-j} \right\rceil \\ &= \sum_{i=0}^{N-1} \lfloor \Delta a_i \rceil^2 + \text{(non-constant terms)} \in \mathcal{R}, \end{aligned} \tag{1}$$

we find that the constant of $p_{\boldsymbol{a}}^{(1)} \cdot p_{\boldsymbol{a}}^{(2)}$ is about Δ^2 times the squared L_2 norm of $\boldsymbol{a}$. The larger the Δ, the higher the precision.

For computing the mean value of $\boldsymbol{a}$, we need to define a polynomial $p = -\sum_{i=0}^{N-1} X^{N-i}$. Then:

$$p_{\boldsymbol{a}}^{(1)} \cdot p = \sum_{i=0}^{N-1} \lfloor \Delta a_i \rceil + \text{(non-constant terms)} \in \mathcal{R}, \tag{2}$$

we can calculate that the mean value of $\boldsymbol{a}$ is about $\frac{(p_{\boldsymbol{a}}^{(1)} \cdot p)_0}{\Delta \cdot N}$.

For the inner product of $\boldsymbol{a}$ and $\boldsymbol{b} = [b_0, b_1, \ldots b_{N-1}] \in \mathbb{R}^N$, $p_{\boldsymbol{b}}^{(2)} \leftarrow$ **pm2**$(\boldsymbol{b})$. Then:

$$p_{\boldsymbol{a}}^{(1)} \cdot p_{\boldsymbol{b}}^{(2)} = \sum_{i=0}^{N-1} \lfloor \Delta a_i \rceil \cdot \lfloor \Delta b_i \rceil + \text{(non-constant terms)} \in \mathcal{R}, \tag{3}$$

it is clear that the constant of $p_{\boldsymbol{a}}^{(1)} \cdot p_{\boldsymbol{b}}$ is about Δ^2 times the inner product of $\boldsymbol{a}$ and $\boldsymbol{b}$.

It is worth noting that with this special packing approach, computing the inner product, L_2 norm and mean value requires only a single polynomial multiplication. Conversely, in the original CKKS encoding scheme, requiring one polynomial multiplication, $\log_2 N$ rotations, and $\log_2 N$ polynomial additions for both inner product and L_2 norm, and $\log_2 N$ rotations and $\log_2 N$ polynomial additions for mean value.

4. Observation

In this section, let's review some AGRs in existing PBFL.

(1) AGR in ShieldFL (Ma et al., 2022): In ShieldFL, servers first check whether the L_2 norm of the local model update $\boldsymbol{g}_u^{(t)}$ is 1, if not, it is discarded directly. Then servers determine a baseline local model update $\boldsymbol{g}^*$ which has the lowest cosine similarity to the global model update $\boldsymbol{g}^{(t-1)}$ of the $(t-1)$-th global iteration. A confidence score is calculated for each local model update $\boldsymbol{g}_u^{(t)}$ with L_2 norm of 1 as $s_u = 1 - cos(\boldsymbol{g}^*, \boldsymbol{g}_u^{(t)})$. Lastly, updating the global model as follows:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \sum_{u=1}^{U} \frac{s_u}{s} \cdot \boldsymbol{g}_u^{(t)}, \tag{4}$$

where $s = \sum_{u=1}^{U} s_u$.

(2) AGR in (Liu et al., 2021) and (Lu et al., 2023): In these two AGRs, the coordinate-wise medians $\boldsymbol{g}^*$ of all local model updates is securely calculated first. Then, in (Liu et al., 2021) the trust score for each local model update $\boldsymbol{g}_u^{(t)}$ is defined as follows:

$$s_u = max\{0, \ln(\frac{1 + \rho_{\boldsymbol{g}^*, \boldsymbol{g}_u^{(t)}}}{1 - \rho_{\boldsymbol{g}^*, \boldsymbol{g}_u^{(t)}}}) - 0.5\},$$

where $\rho_{\boldsymbol{g}^*, \boldsymbol{g}_u^{(t)}} = \frac{Cov(\boldsymbol{g}^*, \boldsymbol{g}_u^{(t)})}{\sigma(\boldsymbol{g}^*)\sigma(\boldsymbol{g}_u^{(t)})}$ is the Person correlation coefficient between $\boldsymbol{g}^*$ and $\boldsymbol{g}_u^{(t)}$. Similarly, in (Lu et al., 2023), the trust score is defined as $s_u = (ReLU(acs(\boldsymbol{g}^*, \boldsymbol{g}_u^{(t)})))^2$, where $acs(\boldsymbol{g}^*, \boldsymbol{g}_u^{(t)})$ is the adjusted cosine similarity (Sarwar et al., 2001) between $\boldsymbol{g}^*$ and $\boldsymbol{g}_u^{(t)}$. Lastly, they update the global model as in Equation (4).

(3) AGR in (Miao et al., 2022): This AGR is actually based on FLTrust (Cao et al., 2020). In this AGR, the server requires to possess a clean trusted dataset and trains a server model update $\boldsymbol{g}_s^{(t)}$ with L_2 norm of 1. In each global iteration t, the server also checks whether the L_2 norm of local model update is 1. Then for each local model update $\boldsymbol{g}_u^{(t)}$ whose L_2 norm is 1, server defines a trust score as $s_u = ReLU(cos(\boldsymbol{g}_u^{(t)}, \boldsymbol{g}_s^{(t)}))$. Lastly, the global model of next round can be computed as in Equation (4).

(4) AGR in FheFL (Rahulamathavan et al., 2023): The basic idea of this AGR is to calculate the squared Euclidean distance between each local model $\boldsymbol{w}_u^{(t)}$ and the current global model $\boldsymbol{w}^{(t)}$: $d_u^{(t)} = \|\boldsymbol{w}^{(t)} - \boldsymbol{w}_u^{(t)}\|^2$. The confidence score for each local model $\boldsymbol{w}_u^{(t)}$ is then calculated as $s_u = 1 - \frac{d_u^{(t)}}{\sum_{u=1}^{U} d_u^{(t)}}$. We can find that $\sum_{u=1}^{U} s_u = U - 1$. Lastly, the global model of next round can be computed as follows:

$$\boldsymbol{w}^{(t+1)} = \frac{1}{U-1} \sum_{u=1}^{U} s_u \cdot \boldsymbol{w}_u^{(t)}.$$

(5) FLAME (Nguyen et al., 2022): FLAME is one of the most advanced AGRs. Firstly, the server filters out local models that are anomalous in direction based on the cosine similarity between local models:

$$\{\boldsymbol{w}_{a_1}, \boldsymbol{w}_{a_2}, \ldots, \boldsymbol{w}_{a_{U'}}\} \leftarrow Clustering(c_{11}, \ldots, c_{UU}),$$

where c_{ij} is the cosine distance between $\boldsymbol{w}_i^{(t)}$ and $\boldsymbol{w}_j^{(t)}$, $i, j \in [1, U]$, U' is the number of admitted local models, and $Clustering()$ is a clustering algorithm such as HDBSCAN (Campello et al., 2013). Secondly, the adaptive clipping bound S_t at round t is calculated for each local model based on the Euclidean distance between the local model and the current global model:

$$S_t \leftarrow \text{Median}(e_1, \ldots, e_U),$$

where e_u is the Euclidean distance between $\boldsymbol{w}_u^{(t)}$ and $\boldsymbol{w}^{(t)}$. Then the server clip the each admitted model by adaptive clipping bound S_t as follows:

$$\boldsymbol{w}_{a_u}^{(t)} = \boldsymbol{w}^{(t)} + (\boldsymbol{w}_{a_u}^{(t)} - \boldsymbol{w}^{(t)}) \cdot \text{Min}(1, S_t/e_{a_u}).$$

But this clipping formula is not friendly to our system framework because $\boldsymbol{w}^{(t)}$ is included in this clipping formula. In ciphertext form, the server cannot refresh $ct(\boldsymbol{w}^{(t)})$, so each global iteration requires one layer of multiplication depth, which is unrealistic. To overcome this problem, we transform the clipping formula into the following form:

$$\boldsymbol{w}_{a_u}^{(t)} = \boldsymbol{w}_{a_u}^{(t)} \cdot \text{Min}(1, S_t/e_{a_u}).$$

We believe that the modified clipping formula still works, because the purpose of clipping is to ensure that the L_2 norm of the local model does not exceed S_t, and our clipping formula can do this. We refer to FLAME using modified clipping formula as M-FLAME. In Sect. 7, our experiments also show that modified-FLAME has very superior robustness. Lastly, the server computes the global model of next round as follows:

$$\boldsymbol{w}^{(t+1)} = \frac{1}{U'} \sum_{u=1}^{U'} \boldsymbol{w}_{a_u}^{(t)} + noi.$$

where noi is the adaptive noising.

In Figure 1, we analyze all indicators used by the above AGRs. We find that for two N-dimensional vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, the Euclidean distance, cosine similarity, Pearson correlation coefficient, and adjusted cosine similarity between $\boldsymbol{a}$ and $\boldsymbol{b}$ can all be deduced when their inner products, L_2 norms and mean values are known. Recall the packing method introduced in Sect. 3.2, it fits our needs very well.

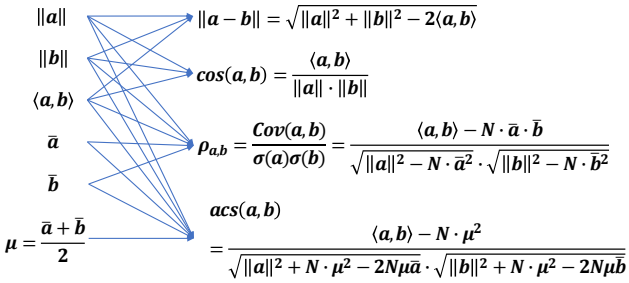


Figure 1. Indicators.

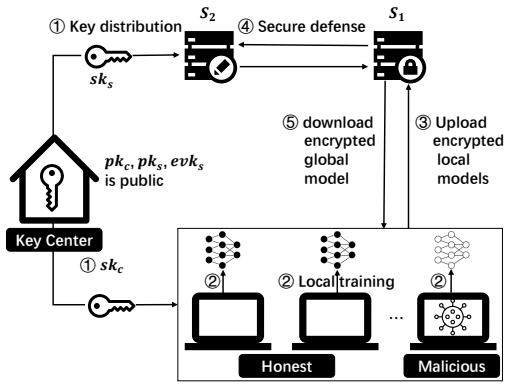


Figure 2. System model.

5. Problem Formulation

5.1. Adversarial Model

As depicted in Fig.2, our system consists of a *Key Center (KC)*, two servers and U clients. We assume *KC* is an independent and trustworthy organization. $\mathcal{S}_1$ and $\mathcal{S}_2$ are two honest-but-curious and non-colluding central servers, meaning they follow the established protocol honestly, but may launch inference attack (see Sect. 2.3) to deduce some sensitive information from the data they receive. Meanwhile, we consider honest clients and malicious clients, but the number of malicious clients is not more than 50%. For honest clients, they are eager to get a high-quality model by combining their datasets and therefore upload their local models correctly. For malicious clients, their goal is to launch various poisoning attacks (see Sect. 2.2) to affect the performance of the global model without being detected. In addition, we also assume malicious clients can collude with $\mathcal{S}_1$ to help it carry out inference attacks.

5.2. Design Goals

Considering the above adversarial model, we have the following five design goals:

- **Privacy**. The privacy of our scheme should have two aspects, one is to maintain the confidentiality of the

Notation	Description
T	The number of global iterations
ψ	The number of local iterations
μ	Local learning rate
U	The number of clients
$\|\mathcal{B}\|$	The batch size
$(p)_0$	the constant term of polynomial p
cos	Cosine similarity value
$\langle \boldsymbol{a}, \boldsymbol{b} \rangle$	The inner product of $\boldsymbol{a}$ and $\boldsymbol{b}$
$\overline{\boldsymbol{a}}$	The mean value of $\boldsymbol{a}$
$\|\boldsymbol{a}\|$	The L_2 norm of $\boldsymbol{a}$
$\|\boldsymbol{a} - \boldsymbol{b}\|$	The Euclidean distance between $\boldsymbol{a}$ and $\boldsymbol{b}$
$p^{(1)}_{\boldsymbol{a}}$	The polynomial obtained by **pm1**$(\boldsymbol{a})$
$p^{(2)}_{\boldsymbol{a}}$	The polynomial obtained by **pm2**$(\boldsymbol{a})$
$ct^{(1)}_{pk}(\boldsymbol{a})$	Ciphertext of **pm1**$(\boldsymbol{a})$ encrypted with pk
$ct^{(2)}_{pk}(\boldsymbol{a})$	Ciphertext of **pm2**$(\boldsymbol{a})$ encrypted with pk

Table 2. Notations.

intermediate results uploaded by clients, and the other is that the final model should be released exclusively to honest clients—other parties (such as $\mathcal{S}_1$, $\mathcal{S}_2$, and malicious clients) cannot access the final model.

- **Robustness**. Our scheme should be robust against poisoning attacks by malicious clients, which means that the final model will not be deteriorated or left backdoors by malicious clients. At the same time, our system can resist the collusion between malicious clients and $\mathcal{S}_1$
- **Efficiency**. Compared to other PBFL schemes, our scheme should significantly reduce the computational burden caused by encryption
- **Accuracy**. Our solution should not sacrifice accuracy to achieve other goals, which means that the accuracy of the final model should be comparable to traditional FL.
- **Flexibility**. Our system should be able to flexibly change AGR to cope with different poisoning attacks and the distribution of clients' data.

6. Design of AegisFL

In this section, we describe the concrete construction of AegisFL. The main notations are shown in Table 2. At a high level, AegisFL consists of key distribution, local training and secure defense. In key distribution phase, the KC generates keys and distributes them to entities according to the protocol. This process only needs to be performed once at the beginning of training. In local training phase, each client trains a local model based on its own local dataset, which is then encrypted and uploaded to $\mathcal{S}_1$. The secure defense is the core of AegisFL, to build AGRs, $\mathcal{S}_1$ and $\mathcal{S}_2$ cooperate to perform consistency check, secure inner product, secure L_2 norm, secure mean value, and secure key conversion.

6.1. Key Distribution (Step ① in Figure 2)

The KC generates $\{pk_s, sk_s, evk_s\}$ and $\{pk_c, sk_c\}$. pk_s, evk_s and pk_c are broadcast to all entities in the system, but sk_s is only sent to $\mathcal{S}_2$, and sk_c is only sent to clients. In addition, $\mathcal{S}_1$ initializes the parameters of the global model $\boldsymbol{w}^{(1)}$ randomly at the beginning of the protocol.

6.2. Local Training (Step ② and ③ in Figure 2)

We show this process in detail in Algorithm 1. We assume that the trained model is flattened into a vector. If the length of this vector is not a multiple of N, it is padded with 0.

Algorithm 1 Local_training

Input: Encrypted global model $ct^{(1)}_{pk_c}(\boldsymbol{w}^{(t)})$, local datasets $D = \{D_1, D_2, \dots D_U\}$, batch size $|\mathcal{B}|$, local learning rate μ, number of local iterations ψ.
Output: $\{ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}_u), ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)}_u)\}_{u=1}^{U}$.
for each client $\mathcal{C}_{u\in[1,U]}$ in parallel **do**
  $p^{(1)}_{\boldsymbol{w}^{(t)}} \leftarrow$ **Dec**$(ct^{(1)}_{pk_c}(\boldsymbol{w}^{(t)}), sk_c)$;
  $\boldsymbol{w}^{(t)} \leftarrow$ **Decode**$(p^{(1)}_{\boldsymbol{w}^{(t)}})$;
  $\boldsymbol{w}^{(t)}_u = \boldsymbol{w}^{(t)}$;
  if $\mathcal{C}_u$ is a honest client **then**
    for each $i \in \{1, 2, \dots, \psi\}$ **do**
      Randomly choose a batch $D_{|\mathcal{B}|}$ from D_u;
      $\boldsymbol{w}^{(t)}_u \leftarrow \boldsymbol{w}^{(t)}_u - \mu \nabla \mathcal{L}(\boldsymbol{w}^{(t)}_u, D_{|\mathcal{B}|})$;
    end for
  else
    $\mathcal{C}_u$ launches a poisoning attack, spawning a poisonous local model $\boldsymbol{w}^{(t)}_u$;
  end if
  $p^{(1)}_{\boldsymbol{w}^{(t)}_u} \leftarrow$ **pm1**$(\boldsymbol{w}^{(t)}_u)$, $p^{(2)}_{\boldsymbol{w}^{(t)}_u} \leftarrow$ **pm2**$(\boldsymbol{w}^{(t)}_u)$;
  $ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}_u) \leftarrow$ **Enc**$(p^{(1)}_{\boldsymbol{w}^{(t)}_u}, pk_s)$;
  $ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)}_u) \leftarrow$ **Enc**$(p^{(2)}_{\boldsymbol{w}^{(t)}_u}, pk_s)$;
  Sends $ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}_u)$ and $ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)}_u)$ to $\mathcal{S}_1$;
end for
return $\{ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}_u), ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)}_u)\}_{u=1}^{U}$.

6.3. Secure Defense (Step ④ and ⑤ in Figure 2)

Security defense essentially involves filtering and refining local models under ciphertext to ensure the performance of the final model. The primary algorithm constituting security

Algorithm 2 Consistency_check

Input: $\mathcal{S}_1$ holds $\{ct^{(1)}_{pk_s}(\boldsymbol{w}_u^{(t)}), ct^{(2)}_{pk_s}(\boldsymbol{w}_u^{(t)})\}_{u=1}^{U}$; $\mathcal{S}_2$ holds $sk_s = s_s$.
Output: Client list $\mathcal{C}^* = \{\mathcal{C}_{b_1}, \mathcal{C}_{b_2}, \ldots, \mathcal{C}_{b_{U^*}}\}$.
$\mathcal{S}_1$:
Generates a random vector $\boldsymbol{r} \in \mathbb{Z}^N$;
$p^{(1)}_{\boldsymbol{r}} \leftarrow \sum_{i=0}^{N-1} r_i \cdot X^i$, $p^{(2)}_{\boldsymbol{r}} \leftarrow -\sum_{i=0}^{N-1} r_i \cdot X^{N-i}$;
for each $u \in \{1, 2, \ldots, U\}$ **do**
  Computes $ct^{(1)}_{pk_s}(\boldsymbol{w}_u^{(t)}) \cdot p^{(2)}_{\boldsymbol{r}} = (c_0^{(1)}, c_1^{(1)}) \in \mathcal{R}^2_{Q_L}$, $ct^{(2)}_{pk_s}(\boldsymbol{w}_u^{(t)}) \cdot p^{(1)}_{\boldsymbol{r}} = (c_0^{(2)}, c_1^{(2)}) \in \mathcal{R}^2_{Q_L}$;
  Sends $((c_0^{(1)})_0, c_1^{(1)}), ((c_0^{(2)})_0, c_1^{(2)}) \in \mathbb{Z}_{Q_L} \times \mathcal{R}_{Q_L}$ to $\mathcal{S}_2$;
end for
$\mathcal{S}_2$:
Initializes client list $\mathcal{C}^* \leftarrow \varnothing$
for each $u \in \{1, 2, \ldots, U\}$ **do**
  Decrypts $res_1 = [(c_0^{(1)})_0 + (s_s \cdot c_1^{(1)})_0]_{Q_L}$ and $res_2 = [(c_0^{(2)})_0 + (s_s \cdot c_1^{(2)})_0]_{Q_L}$;
  if $|\frac{res_1 - res_2}{\Delta}| < \kappa$ **then**
    Adds client $\mathcal{C}_u$ to $\mathcal{C}^*$;
  end if
end for
Sends $\mathcal{C}^* = \{\mathcal{C}_{b_1}, \mathcal{C}_{b_2}, \ldots, \mathcal{C}_{b_{U^*}}\}$ to $\mathcal{S}_1$;
return $\mathcal{C}^*$.

defense include consistency check, secure inner product, secure L_2 norm, secure mean value, and secure key conversion. We present it in detail in Algorithm 3. There is a general and stationary assumption that in the first global iteration ($i.e., t = 1$), all clients are honest and they will not launch poison attacks (Ma et al., 2022; Nguyen et al., 2022).

6.3.1. Consistency Check

The purpose of consistency check is to ensure that the clients package their local model updates correctly according to the packing method. We present it in Algorithm 2. In our scheme, instead of sending $(c_0^{(1)}, c_1^{(1)}), (c_0^{(2)}, c_1^{(2)}) \in \mathcal{R}^2_{Q_L}$ directly, $\mathcal{S}_1$ sends $((c_0^{(1)})_0, c_1^{(1)}), ((c_0^{(2)})_0, c_1^{(2)}) \in \mathbb{Z}_{Q_L} \times \mathcal{R}_{Q_L}$ to $\mathcal{S}_2$. Upon receiving them, $\mathcal{S}_2$ decrypts with sk_s to get res_1 and res_2, and then checks whether $|\frac{res_1 - res_2}{\Delta}| < \kappa$, where κ is the error threshold caused by noise, if so, $\mathcal{C}_u$ is accepted by $\mathcal{S}_2$. We know from Sect. 3.2 that $\frac{res_1}{\Delta}$ and $\frac{res_2}{\Delta}$ are actually the approximate inner product of $\boldsymbol{w}_u^{(t)}$ and $\boldsymbol{r}$. If the client does not package its local model correctly, it will not pass this detection due to the randomness of $\boldsymbol{r}$. Finally, we assume that U^* clients pass the check, and $\mathcal{S}_2$ sends the client list $\mathcal{C}^* = \{\mathcal{C}_{b_1}, \mathcal{C}_{b_2}, \ldots, \mathcal{C}_{b_{U^*}}\}$ to $\mathcal{S}_1$.

Compared to sending $ct^{(1)}_{pk_s}(\boldsymbol{w}_u^{(t)}) \cdot p^{(2)}_{\boldsymbol{r}}$ and $ct^{(2)}_{pk_s}(\boldsymbol{w}_u^{(t)}) \cdot p^{(1)}_{\boldsymbol{r}}$ directly to $\mathcal{S}_2$, our approach has two advantages. Firstly, it reduces the communication cost between $\mathcal{S}_1$ and $\mathcal{S}_1$ from $2N \cdot \log_2 Q_L$ to $(N+1) \cdot \log_2 Q_L$. Secondly, after decryption, only the constant terms are meaningful, that is, $\mathcal{S}_2$ cannot get any valuable information from other terms, further enhancing data privacy.

6.3.2. Secure Inner Product, L_2 Norm and Mean Value

In lines 8-15 of Algorithm 3, $\mathcal{S}_1$ and $\mathcal{S}_2$ cooperate to calculate $\{\|\boldsymbol{w}_{b_u}^{(t)}\|, \langle \boldsymbol{w}_{b_u}^{(t)}, \boldsymbol{w}^{(t)} \rangle, \overline{\boldsymbol{w}}_{b_u}^{(t)}\}_{u=1}^{U^*}$, $\|\boldsymbol{w}^{(t)}\|$, and $\overline{\boldsymbol{w}}^{(t)}$. Then we can construct all AGRs described in Sect. 4. We define the ciphertext of the global model for the next round as: $ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t+1)}) = \sum_{u=1}^{U^*} \mathbf{cMul}(\frac{s_{b_u}}{s}, ct^{(1)}_{pk_s}(\boldsymbol{w}_{b_u}^{(t)}), \Delta)$, where s_{b_u} is the trust score for each client $\mathcal{C}_{b_u} \in \mathcal{C}^*$, $s = \sum_{u=1}^{U^*} s_{b_u}$. The global model is implicit in the coefficients of the polynomial $p^{(1)}_{\boldsymbol{w}^{(t+1)}}$. It is worth noting that our system only requires two multiplication levels. We show the specific process of secure inner product, L_2 norm and mean value in Appendix A.

6.3.3. Secure Key Conversion

Clients upload intermediate results encrypted with pk_s to ensure their confidentiality, but will get the global model encrypted with pk_c from $\mathcal{S}_1$. The purpose of this is that even if the malicious clients collude with $\mathcal{S}_1$ and leak the private key sk_c to $\mathcal{S}_1$, as long as $\mathcal{S}_1$ and $\mathcal{S}_2$ are non-colluding, $\mathcal{S}_1$ still cannot decrypt the encrypted information. We show the specific process of secure key conversion in Appendix B.

6.4. Security Analysis

The security of our system can depends on the following facts:

- **Fact 1**: CKKS has the IND-CPA security property.
- **Fact 2**: If a random polynomial r is uniformly distributed on $\mathcal{R}$ and also independent from any plaintext $x \in \mathcal{R}$, then $x \pm r$ is also uniformly random and independent from x.
- **Fact 3**: In our case, the system involves 5 equations (which are either linear forms or binary forms) for each client, but the number of variables is usually more than 8192. With a significant amount of freedom, it becomes impossible to guess a specific value of a variable.

Then we give the following Theorem:

Theorem 6.1. *$\mathcal{S}_1$, $\mathcal{S}_2$, and malicious clients can get nothing about the sensitive information of honest clients.*

Proof. In our scheme, we consider two honest-but-curious and non-colluding servers, honest clients and malicious

Algorithm 3 Secure_defense

Input: $\{ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}_u), ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)}_u)\}^U_{u=1}$.
Output: $ct^{(1)}_{pk_c}(\boldsymbol{w}^{(t+1)})$.
if $t == 1$ **then**
  $\mathcal{S}_1$ computes $ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t+1)}) = \sum_{u=1}^{U}$ **cMul**$(ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}_u), \frac{1}{U}, \Delta)$, $ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t+1)}) = \sum_{u=1}^{U}$ **cMul**$(ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)}_u), \frac{1}{U}, \Delta)$;
else
  $\mathcal{S}_1$ and $\mathcal{S}_2$ invoke Consistency_check$(\{ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}_u), ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)}_u)\}^U_{u=1}, sk_s)$ to get $\mathcal{C}^* = \{\mathcal{C}_{b_1}, \mathcal{C}_{b_2}, \ldots, \mathcal{C}_{b_{U^*}}\}$;
  $\mathcal{S}_1$ sets $ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)})$ and $ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)})$ as the baseline;
  $\mathcal{S}_1$ and $\mathcal{S}_2$:
  $\|\boldsymbol{w}^{(t)}\| \leftarrow$ Secure_L_2_norm$(ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}), ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)}), sk_s)$;
  $\overline{\boldsymbol{w}}^{(t)} \leftarrow$ Secure_mean_value$(ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}), sk_s)$;
  for each client $\mathcal{C}_{b_{u\in[1,U^*]}} \in \mathcal{C}^*$ **do**
    $\mathcal{S}_1$ and $\mathcal{S}_2$:
    $\|\boldsymbol{w}^{(t)}_{b_u}\| \leftarrow$ Secure_L_2_norm$(ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}_{b_u}), ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)}_{b_u}), sk_s)$;
    $\langle \boldsymbol{w}^{(t)}_{b_u}, \boldsymbol{w}^{(t)} \rangle \leftarrow$ Secure_inner_product$(ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}), ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)}_{b_u}), sk_s)$;
    $\overline{\boldsymbol{w}}^{(t)}_{b_u} \leftarrow$ Secure_mean_value$(ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}_{b_u}), sk_s)$;
  end for
  $\mathcal{S}_1$:
  Computes a trust score s_{b_u} for each client $\mathcal{C}_{b_u} \in \mathcal{C}^*$ based on $\{\|\boldsymbol{w}^{(t)}_{b_u}\|, \|\boldsymbol{w}^{(t)}\|, \langle \boldsymbol{w}^{(t)}_{b_u}, \boldsymbol{w}^* \rangle, \overline{\boldsymbol{w}}^{(t)}_{b_u}, \overline{\boldsymbol{w}}^{(t)}\}$;
  Computes $s = \sum_{i=1}^{U^*} s_{b_u}$;
  Computes $ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t+1)}) = \sum_{u=1}^{U^*}$ **cMul**$(\frac{s_{b_u}}{s}, ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t)}_{b_u}), \Delta)$, $ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t+1)}) = \sum_{u=1}^{U^*}$ **cMul**$(\frac{s_{b_u}}{s}, ct^{(2)}_{pk_s}(\boldsymbol{w}^{(t)}_u), \Delta)$;
end if
$\mathcal{S}_1$ and $\mathcal{S}_2$ invoke Secure_key_conversion$(ct^{(1)}_{pk_s}(\boldsymbol{w}^{(t+1)}), sk_s)$ to get $ct^{(1)}_{pk_c}(\boldsymbol{w}^{(t+1)})$, and send it to clients $\mathcal{C}_{b_{u\in[1,U^*]}}$;
return $ct^{(1)}_{pk_c}(\boldsymbol{w}^{(t+1)})$.

clients. And the malicious clients can collude with one of the servers. Therefore, there are three adversary cases:

- **Case 1**: Servers and malicious clients do not collude. In this case, for $\mathcal{S}_1$, it only gets the encrypted local models, based on Fact 1, it can not get information of clients' data. For $\mathcal{S}_2$, it gets $r + p^{(1)}_a$ in secure key conversion phase, based on Fact 2, it can't get any information from it. For a malicious clients, $\mathcal{S}_1$ does not return any data to them. In addition, when building a AGR, the L_2 norms, mean values, and inner products between clients' local models and the global model of the last round are public, for each client, the server can construct up to 5 equations, but there are N unknowns, $5 \ll N$, based on Fact 3, server can't push out valuable information.
- **Case 2**: Malicious clients collude with $\mathcal{S}_1$. In this case, the malicious clients will leak the private key sk_c to $\mathcal{S}_1$, but this is meaningless because the honest clients' models are encrypted with the public key pk_s. And although the malicious clients will get the global model from $\mathcal{S}_1$ and perform decryption, they cannot calculate the models of honest clients unless there are $U-1$ malicious clients. But $U-1$ malicious clients is an unrealistic assumption.
- **Case 3**: Malicious clients collude with $\mathcal{S}_2$. In this case, the malicious clients will get the private key sk_s from $\mathcal{S}_2$, but this is also meaningless because they can't get the ciphertexts of the honest clients' local models. And the private key sk_c is also useless to $\mathcal{S}_2$.

□

7. Experiments

7.1. Experiment Setup

7.1.1. Environment

Our experiments run on a Windows PC with Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz and 16 GB memory. We use PyTorch to train the neural network. For the implementation of HE, we first incorporated our encoding method into the SEAL library, and then utilized the SEAL-Python library to enable its execution in a Python environment. We set $N = 8192, \Delta = 2^{40}$ and Q_L as a 200-bit number, achieving 128-bit security.

7.1.2. Clients, Datasets, Models, and Attacks

We consider 30 clients in total, and three cases where the number of malicious clients are 3, 9, and 12. We adopt two widely used datasets HAR and MNIST. We use a simple neural network that contains an input layer, a fully connected layer, and an output layer. When using HAR, there are 72710 parameters; when using MNIST, there are 101770 parameters. Meanwhile, we examine three types of attacks: untargeted attack, label flipping attack, and scaling attack. Please see Appendix C.1 for details.

7.2. Effectiveness Analysis

We test the ability of FedAvg (baseline), M-FLAME and ShieldFL to resist poisoning attacks under both plaintext and ciphertext. We present the experimental results in Appendix C.2. Our experimental results reveal three key phenomena: First, M-FLAME is effective in defending against all types of attacks encountered in our experiments. Secondly, ShieldFL shows improved defense capabilities against untargeted attacks and label flipping attacks on the HAR (non-iid) dataset, yet it is less effective against more subtle scaling attacks. In contrast, M-FLAME performs better on the MNIST (iid) dataset and effectively counters scaling attacks. This indicates that different AGRs exhibit varying levels of effectiveness depending on the specific scenario. Third, the accuracy under encryption is on par with, or slightly exceeds, the accuracy in plaintext, indicating that at $\Delta = 2^{40}$, approximate HE does not adversely affect model training. This insight underscores the potential of HE in maintaining model performance while ensuring data privacy.

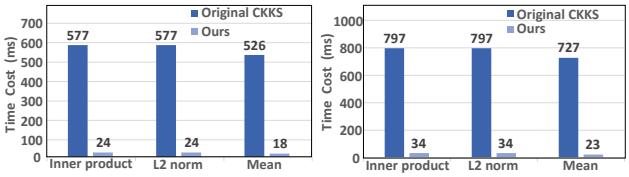
7.3. Computational Efficiency Analysis

We measure the time required to compute the inner product of two local models, the L_2 norm or mean value of one local model when trained separately with the MNIST and HAR datasets. The original CKKS scheme serves as our baseline. (Miao et al., 2022) has shown that both the Paillier cryptosystem and its batch computation variant (BatchCrypt) exhibit significantly lower efficiency in encryption, decryption, and plaintext-ciphertext multiplication compared to CKKS, we no longer include Paillier in our comparison.

As shown in Figure 3, compared to the original CKKS, our encoding approach achieves approximately 24 times speedup in computing inner products and L_2 norms, and about 30 times speedup in calculating mean values. This is because the original CKKS scheme involves multiple rotations in computing these values, and rotation is a relatively time-consuming operation.

7.4. Communication Efficiency Analysis

We measure the communication volume between clients and $\mathcal{S}_1$ during one global iteration on two datasets. We consider



(a) HAR (b) MNIST

Figure 3. The time cost of calculating inner product, L_2 norm and mean value.

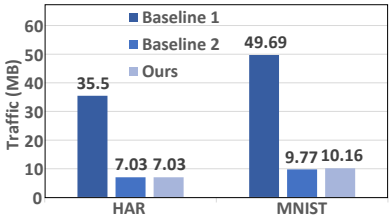


Figure 4. Traffic

the Paillier-based scheme ShieldFL (Ma et al., 2022) as baseline 1 and original CKKS-based scheme (Miao et al., 2022) as baseline 2. As shown in Figure 4, the traffic of our scheme is significantly lower than baseline 1 and roughly equivalent to or slightly higher than baseline 2. In original CKKS, there are at most $N/2$ slots for packaging data, but in our scheme all N slots can be used for packaging data. Therefore, although the client needs to upload twice ciphertexts to $\mathcal{S}_1$ in our scheme, the communication burden will not increase compared to baseline 2.

8. Conclusion and Open Problem

We introduce AegisFL, a novel PBFL framework. Compared to previous approaches, it offers two main advantages. First, it achieves high efficiency, thanks to our packaging scheme that requires only a single ciphertext multiplication to compute the inner products, L_2 norms and mean values of vectors. Second, it boasts high flexibility, our system is not limited to a single robust aggregation algorithm but can combine various robust aggregation methods to address different adversarial environments. Moreover, our experiments validate the effectiveness and efficiency of our method.

Our approach is compatible with the majority of similarity-based AGRs because key similarity metrics, derived from L_2 norms, mean value, and inner products, can be effectively applied. However, it is incompatible with statistics-based and feature extraction-based AGRs, this is because computing the coordinate-wise median/trimmed mean and the top right singular eigenvector under CKKS ciphertext remains a highly challenging open problem. For performance-based AGRs, they require access to the clients' models in plaintext, which violates the privacy requirements and falls outside the scope of this discussion. We leave the exploration of efficient compatibility with a wider range of aggregation schemes under HE for future work.

Acknowledgements

This work is partially supported by the National Natural Science Foundation of China (No. 12271306) and the National Key Research and Development Program of China (Grant No. 2018YFA0704702).

Impact Statement

In this paper, we present AegisFL, a novel federated learning system that enhances privacy and security against poisoning attacks. By using advanced homomorphic encryption, AegisFL offers a more efficient and flexible solution compared to existing methods. This improvement is significant for the broader societal application of federated learning, as it increases data protection and model integrity in various fields, including healthcare, finance, and more. While our work primarily focuses on advancing machine learning technology, we recognize the importance of considering the ethical use of such advancements. Therefore, we encourage responsible deployment and further discussion on the potential impacts of our work.

References

Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., and Shmatikov, V. How to backdoor federated learning. In *International conference on artificial intelligence and statistics*, pp. 2938–2948. PMLR, 2020.

Baruch, G., Baruch, M., and Goldberg, Y. A little is enough: Circumventing defenses for distributed learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Campello, R. J., Moulavi, D., and Sander, J. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pp. 160–172. Springer, 2013.

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

Cheon, J. H., Kim, A., Kim, M., and Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pp. 409–437. Springer, 2017.

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

Geiping, J., Bauermeister, H., Dröge, H., and Moeller, M. Inverting gradients-how easy is it to break privacy in federated learning? *Advances in Neural Information Processing Systems*, 33:16937–16947, 2020.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

Liu, X., Li, H., Xu, G., Chen, Z., Huang, X., and Lu, R. Privacy-enhanced federated learning against poisoning adversaries. *IEEE Transactions on Information Forensics and Security*, 16:4574–4588, 2021.

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

Ma, Z., Ma, J., Miao, Y., Li, Y., and Deng, R. H. Shieldfl: Mitigating model poisoning attacks in privacy-preserving federated learning. *IEEE Transactions on Information Forensics and Security*, 17:1639–1654, 2022.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Miao, Y., Liu, Z., Li, H., Choo, K.-K. R., and Deng, R. H. Privacy-preserving byzantine-robust federated learning via blockchain systems. *IEEE Transactions on Information Forensics and Security*, 17:2848–2861, 2022.

Nasr, M., Shokri, R., and Houmansadr, A. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*, pp. 739–753. IEEE, 2019.

Nguyen, T. D., Rieger, P., De Viti, R., Chen, H., Brandenburg, B. B., Yalame, H., Möllering, H., Fereidooni, H., Marchal, S., Miettinen, M., et al. {FLAME}: Taming backdoors in federated learning. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 1415–1432, 2022.

Rahulamathavan, Y., Herath, C., Liu, X., Lambotharan, S., and Maple, C. Fhefl: Fully homomorphic encryption friendly privacy-preserving federated learning with byzantine users. *arXiv preprint arXiv:2306.05112*, 2023.

Roy Chowdhury, A., Guo, C., Jha, S., and van der Maaten, L. Eiffel: Ensuring integrity for federated learning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2535–2549, 2022.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pp. 285–295, 2001.

Shejwalkar, V. and Houmansadr, A. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *NDSS*, 2021.

Wang, H., Sreenivasan, K., Rajput, S., Vishwakarma, H., Agarwal, S., Sohn, J.-y., Lee, K., and Papailiopoulos, D. Attack of the tails: Yes, you really can backdoor federated learning. *Advances in Neural Information Processing Systems*, 33:16070–16084, 2020.

Xie, C., Huang, K., Chen, P.-Y., and Li, B. Dba: Distributed backdoor attacks against federated learning. In *International conference on learning representations*, 2019.

Yasuda, M., Shimoyama, T., Kogure, J., Yokoyama, K., and Koshiba, T. Practical packing method in somewhat homomorphic encryption. In *Data Privacy Management and Autonomous Spontaneous Security: 8th International Workshop, DPM 2013, and 6th International Workshop, SETOP 2013, Egham, UK, September 12-13, 2013, Revised Selected Papers*, pp. 34–50. Springer, 2014.

Zhang, C., Li, S., Xia, J., Wang, W., Yan, F., and Liu, Y. {BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning. In *2020 USENIX annual technical conference (USENIX ATC 20)*, pp. 493–506, 2020.

Zhang, W., Tople, S., and Ohrimenko, O. Leakage of dataset properties in {Multi-Party} machine learning. In *30th USENIX security symposium (USENIX Security 21)*, pp. 2687–2704, 2021.

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

A. Secure Inner Product, L_2 Norm and Mean Value

Benefit from the knowledge of Sect. 3.2, for two N-dimensional vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, we show how $\mathcal{S}_1$ and $\mathcal{S}_2$ can safely calculate the L_2 norm of $\boldsymbol{a}$, the inner product of $\boldsymbol{a}$ and $\boldsymbol{b}$ and the mean value of $\boldsymbol{a}$ in Algorithm 4, Algorithm 5 and Algorithm 6 respectively.

Algorithm 4 Secure_L_2_norm

Input: $\mathcal{S}_1$ holds $ct^{(1)}_{pk_s}(\boldsymbol{a}), ct^{(2)}_{pk_s}(\boldsymbol{a}) \in \mathcal{R}^2_{Q_l}$; $\mathcal{S}_2$ holds $sk_s = s_s$.
Output: $\|\boldsymbol{a}\|$.
$\mathcal{S}_1$:
Mul$(ct^{(1)}_{pk_s}(\boldsymbol{a}), ct^{(2)}_{pk_s}(\boldsymbol{a}), evk) = (c_0, c_1) \in \mathcal{R}^2_{Q_{l-1}}$;
Sends $((c_0)_0, c_1) \in \mathbb{Z}_{Q_{l-1}} \times \mathcal{R}_{Q_{l-1}}$ to $\mathcal{S}_2$;
$\mathcal{S}_2$:
Decrypts $result = [(c_0)_0 + (s_s \cdot c_1)_0]_{Q_{l-1}}$;
return $\|\boldsymbol{a}\| = \sqrt{\frac{result}{\Delta}}$.

Algorithm 5 Secure_inner_product

Input: $\mathcal{S}_1$ holds $ct^{(1)}_{pk_s}(\boldsymbol{a})$, $ct^{(2)}_{pk_s}(\boldsymbol{b}) \in \mathcal{R}^2_{Q_l}$; $\mathcal{S}_2$ holds $sk_s = s_s$.
Output: $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$.
$\mathcal{S}_1$:
Mul$(ct^{(1)}_{pk_s}(\boldsymbol{a}), ct^{(2)}_{pk_s}(\boldsymbol{b}), evk) = (c_0, c_1) \in \mathcal{R}^2_{Q_{l-1}}$;
Sends $((c_0)_0, c_1) \in \mathbb{Z}_{Q_{l-1}} \times \mathcal{R}_{Q_{l-1}}$ to $\mathcal{S}_2$;
$\mathcal{S}_2$:
Decrypts $result = [(c_0)_0 + (s_s \cdot c_1)_0]_{Q_{l-1}}$;
return $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \frac{result}{\Delta}$.

Algorithm 6 Secure_mean_value

Input: $\mathcal{S}_1$ holds $ct^{(1)}_{pk_s}(\boldsymbol{a}) \in \mathcal{R}^2_{Q_l}$; $\mathcal{S}_2$ holds $sk_s = s_s$.
Output: $\overline{\boldsymbol{a}}$.
$\mathcal{S}_1$:
Generates a polynomial $p = -\sum_{i=0}^{N-1} X^{N-i}$;
Computes $ct^{(1)}_{pk_s}(\boldsymbol{a}) \cdot p = (c_0, c_1) \in \mathcal{R}^2_{Q_l}$;
Sends $((c_0)_0, c_1) \in \mathbb{Z}_{Q_l} \times \mathcal{R}_{Q_l}$ to $\mathcal{S}_2$;
$\mathcal{S}_2$:
Decrypts $result = [(c_0)_0 + (s_s \cdot c_1)_0]_{Q_l}$;
return $\overline{\boldsymbol{a}} = \frac{result}{\Delta \cdot N}$.

B. Secure Key Conversion

As illustrated by Algorithm 7, secure key conversion refers to converting ciphertext $ct^{(1)}_{pk_s}(\boldsymbol{a})$ to cihpertext $ct^{(1)}_{pk_c}(\boldsymbol{a})$, where $\boldsymbol{a}$ is a N-dimensional vector. Due to the randomness of $r \in \mathcal{R}$, $\boldsymbol{a}$ will not be exposed to $\mathcal{S}_2$.

Algorithm 7 Secure_key_conversion

Input: $\mathcal{S}_1$ holds $ct^{(1)}_{pk_s}(\boldsymbol{a}) = (c_0, c_1) \in \mathcal{R}^2_{Q_l}$; $\mathcal{S}_2$ holds sk_s.
Output: $ct^{(1)}_{pk_c}(\boldsymbol{a})$.
$\mathcal{S}_1$:
Generates a random polynomial $r \leftarrow \mathcal{R}$;
$\hat{ct} = (c_0 + r, c_1) \leftarrow \mathbf{cAdd}(ct^{(1)}_{pk_s}(\boldsymbol{a}), r)$;
Sends $\hat{ct}$ to $\mathcal{S}_2$;
$\mathcal{S}_2$:
$r + p^{(1)}_{\boldsymbol{a}} \leftarrow \mathbf{Dec}(\hat{ct}, sk_s)$;
$ct_{pk_c}(r + p^{(1)}_{\boldsymbol{a}}) \leftarrow \mathbf{Enc}(r + p^{(1)}_{\boldsymbol{a}}, pk_c)$;
Sends $ct_{pk_c}(r + p^{(1)}_{\boldsymbol{a}})$ to $\mathcal{S}_1$;
$\mathcal{S}_1$:
$ct^{(1)}_{pk_c}(\boldsymbol{a}) \leftarrow \mathbf{cAdd}(ct_{pk_c}(r + p^{(1)}_{\boldsymbol{a}}), -r)$;
return $ct^{(1)}_{pk_c}(\boldsymbol{a})$.

C. Experiments

C.1. Experiment Setup Details

The Human Activity Recognition (HAR) dataset comprises data collected from the smartphones of 30 real-world users. This dataset includes signals from multiple sensors located on the users' smartphones, with the objective to predict the user's activity out of six possible categories: walking, walking upstairs, walking downstairs, sitting, standing, and lying down. Each example in this dataset encompasses 561 features, amounting to a total of 10,299 instances. In our experiment, each user is considered as a distinct client, with 75% of their data utilized for training purposes and the remaining 25% serving as test cases. For HAR, we set the number of global iterations $T = 100$, the number of local iterations $\psi = 50$, and the batch size $|\mathcal{B}| = 100$. We note that due to individual variances, sensor positioning, and variations in the execution of activities, the data in the HAR dataset is non-independent and identically distributed (non-iid).

The MNIST dataset is a classic dataset in the field of machine learning, primarily used for training and testing image processing systems, particularly in the task of handwritten digit recognition. It consists of 60,000 training images and 10,000 test images, each a 28*28 pixel grayscale representation of digits ranging from 0 to 9. In our experiment, we distribute the training set evenly across 30 clients, ensuring that the data is independent and identically distributed (iid) among them, providing a consistent basis for our analysis. For MNIST, we set the number of global iterations $T = 100$, the number of local iterations $\psi = 50$, and the batch size $|\mathcal{B}| = 100$.

For untargeted attack, malicious clients upload random intermediate results that follow a (0,1) normal distribution. In the case of label flipping attack, each malicious client alters the label from l to $M - l - 1$, where l is an element of the set $\{0, 1, \ldots, N-1\}$ and N represents the total number of labels. Regarding scaling attacks, malicious clients replicate 20% of their local training samples, create a feature-pattern trigger by setting every 20-th feature to 0, modify their labels to 2, and then incorporate these altered samples into their local training dataset. Subsequently, during each global iteration, these clients calculate their intermediate results using the modified training data. These intermediate results are then scaled by a factor prior to being transmitted to the server.

C.2. Experimental Results

In this section, we present the testing accuracy of FedAvg, M-FLAME, and ShieldFL under different attacks. For each attack, we evaluate scenarios with 3, 9, and 12 attackers respectively. The result of Scaling attack are presented in the form of "testing accuracy / backdoor attack success rate". Table 3 and Table 4 show the results on HAR and MNIST, respectively.

AGR \ ATTACK		NO	UNTARGETED ATTACK			LABLE FLIPPING			SACLING ATTACK		
			3	9	12	3	9	12	3	9	12
FEDAVG	PLANTEXT	0.949	0.731	0.431	0.240	0.934	0.915	0.620	0.946/0.968	0.902/0.983	0.890/0.991
	CIPHERTEXT	0.946	0.732	0.429	0.240	0.936	0.919	0.620	0.945/0.970	0.905/0.981	0.895/0.993
M-FLAME	PLANTEXT	0.937	0.939	0.910	0.896	0.933	0.934	0.935	0.936/0.189	0.902/0.209	0.886/0.205
	CIPHERTEXT	0.938	0.941	0.910	0.898	0.933	0.933	0.934	0.939/0.188	0.902/0.205	0.888/0.206
SHIELFL	PLANTEXT	0.945	0.951	0.949	0.940	0.951	0.940	0.938	0.951/0.269	0.886/0.375	0.808/0.423
	CIPHERTEXT	0.945	0.951	0.948	0.943	0.949	0.937	0.938	0.952/0.269	0.888/0.377	0.810/0.424

Table 3. Testing Accuracy on HAR.

AGR \ ATTACK		NO	UNTARGETED ATTACK			LABLE FLIPPING			SACLING ATTACK		
			3	9	12	3	9	12	3	9	12
FEDAVG	PLANTEXT	0.953	0.236	0.089	0.072	0.950	0.929	0.885	0.953/0.941	0.955/0.986	0.955/0.993
	CIPHERTEXT	0.955	0.233	0.091	0.073	0.952	0.928	0.885	0.953/0.940	0.956/0.989	0.953/0.995
M-FLAME	PLANTEXT	0.950	0.953	0.949	0.946	0.952	0.948	0.945	0.953/0.120	0.950/0.170	0.950/0.167
	CIPHERTEXT	0.948	0.953	0.950	0.944	0.956	0.951	0.946	0.949/0.118	0.950/0.171	0.951/0.168
SHIELFL	PLANTEXT	0.934	0.941	0.933	0.928	0.926	0.922	0.918	0.934/0.139	0.891/0.373	0.947/0.521
	CIPHERTEXT	0.935	0.943	0.934	0.926	0.925	0.9922	0.920	0.935/0.139	0.891/0.375	0.944/0.518

Table 4. Testing Accuracy on MNIST.