Digital Communications and Networks 10 (2024) 355–368

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/dcan

# PEPFL: A framework for a practical and efficient privacy-preserving federated learning

Yange Chen [a,b,c], Baocang Wang [a,b,*], Hang Jiang [c], Pu Duan [d], Yuan Ping [b,**], Zhiyong Hong [e]

[a] State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an, 710071, China
[b] School of Information Engineering, Xuchang University, Xuchang, 461000, China
[c] School of Telecommunications Engineering, Xidian University, Xi'an, 710071, China
[d] Secure Collaborative Intelligence Laboratory, Ant Group, Hangzhou, 310000, China
[e] Facility of Intelligence Manufacture, Wuyi University, Jiangmen, 529020, China

ARTICLE INFO

*Keywords:*
Federated learning
Partially single instruction multiple data
Momentum gradient descent
ElGamal
Multi-key
Homomorphic encryption

ABSTRACT

As an emerging joint learning model, federated learning is a promising way to combine model parameters of different users for training and inference without collecting users' original data. However, a practical and efficient solution has not been established in previous work due to the absence of efficient matrix computation and cryptography schemes in the privacy-preserving federated learning model, especially in partially homomorphic cryptosystems. In this paper, we propose a Practical and Efficient Privacy-preserving Federated Learning (PEPFL) framework. First, we present a lifted distributed ElGamal cryptosystem for federated learning, which can solve the multi-key problem in federated learning. Secondly, we develop a Practical Partially Single Instruction Multiple Data (PSIMD) parallelism scheme that can encode a plaintext matrix into single plaintext for encryption, improving the encryption efficiency and reducing the communication cost in partially homomorphic cryptosystem. In addition, based on the Convolutional Neural Network (CNN) and the designed cryptosystem, a novel privacy-preserving federated learning framework is designed by using Momentum Gradient Descent (MGD). Finally, we evaluate the security and performance of PEPFL. The experiment results demonstrate that the scheme is practicable, effective, and secure with low communication and computation costs.

## 1. Introduction

As one of the most promising technologies, federated learning plays a significant role in joint learning model, and has been widely studied and applied in many fields such as image classification [1,2], medical imaging [3], computer vision [4], blockchain [5], and automatic systems [6,7]. For traditional centralized learning, it requires collecting a large amount of user data to train the model. However, user data may contain sensitive private information, which may lead to user information leakage. To avoid privacy leakage and break the data islands, federated learning has been proposed [8]. As an emerging distributed learning framework, federated learning can train model parameters from different users cooperatively without knowing their original data. The framework is a privacy protection approach that has gained the trust of more participants. Therefore, federated learning has aroused widespread concern in industry and academia [9,10].

Although federated learning preserves privacy to a certain extent, researchers [9,11] have shown that an aggregation server or attackers can still restore some sensitive information by the shared gradients or weights. To prevent the leakage of sensitive information, some investigators proposed new privacy-preserving federated learning frameworks based on differential privacy or different encryption schemes [12,13]. However, these schemes do not consider practicability owing to users frequently interacting with the server online except [14]. Besides, privacy-preserving federated learning schemes based on cryptosystems have low efficiency. Based on Fully Homomorphic Encryption (FHE), privacy-preserving deep learning schemes utilize various Single Instruction Multiple Data (SIMD) methods for parallel computing, such as vector-based [15] and lattice-based [16] methods. However, FHE data have high inflation rate and high ciphertext homomorphism computational cost, which cannot efficiently support federated multi-party calculation. Therefore, partially homomorphic encryption schemes

* Corresponding author. State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an, 710071, China.
** Corresponding author.
*E-mail addresses:* bcwang79@aliyun.com (B. Wang), pingyuan@bupt.cn (Y. Ping).

https://doi.org/10.1016/j.dcan.2022.05.019
Received 4 September 2021; Received in revised form 21 May 2022; Accepted 22 May 2022
Available online 1 June 2022

have been studied in many schemes [11,12,14], but parallel matrix computation similar to FHE schemes [15–17] is not used in these schemes, leading to low efficiency and high communication cost owing to the encryption of single plaintext or vector plaintext. Moreover, few schemes have considered the multi-key question in federated learning. Also, to improve the convergence rate of Stochastic Gradient Descent (SGD) in the training process, some schemes [18–20] introduced the momentum term. However, they do not consider privacy.

Based on the aforementioned issues, to address the questions of interactivity, parallel computing, multi-key and accelerating convergence, in this paper, we propose a Practical and Efficient Privacy-preserving Federated Learning (PEPFL) framework, which is the first privacy-preserving scheme supporting the Parallel Partially Single Instruction Multiple Data (PSIMD) method in partially homomorphic cryptosystems. The framework leverages the users' noninteraction method [14] and Momentum Gradient Descent (MGD) way to make the system model practicable. The proposed lifted distributed ElGamal cryptosystem can solve the multi-key problem in federated learning. In summary, the main contributions of PEPFL are as follows:

- **Privacy-preserving framework**. We propose a noninteractive and efficient privacy-preserving collaborative training and prediction framework with MGD in the federated learning model. By introducing trainers to interact with the aggregation server, the framework can realize user offline training and prediction in the federated learning process. The framework protects the privacy of all users and trainers and accelerates convergence with MGD.
- **Cryptography scheme**. We design a novel lifted distributed ElGamal cryptosystem, which can realize distributed encryption and decryption while maintaining homomorphism. The cryptosystem avoids collusion issues and solves the multi-user multi-key problem of different public and private key pairs.
- **PSIMD parallel technology**. We present an efficient PSIMD method, which can encode a plaintext matrix into a single plaintext for encryption and decode the plaintext prediction result into a plaintext matrix. The method realizes efficient encryption and reduces the communication cost compared with the prior methods in partially homomorphic cryptographies.
- **Security analysis and implementation**. We analyze and evaluate the security and performance of PEPFL. The analysis and experimental results demonstrate that PEPFL has high security, feasibility and efficiency with low communication and computation costs for all users and trainers.

The remainder of this paper is organized as follows: First, we discuss the related work in detail in Section 2. Then we introduce some preliminaries in Section 3. Section 4 establishes the system and threat models. Section 5 describes the details of PEPFL. Section 6 analyzes the security of the constructed framework. The performance analysis and experimental results are shown in Section 7. Finally, we draw the conclusions in Section 8.

## 2. Related work

In this section, we review the existing works on privacy-preserving deep learning. Existing privacy-preserving schemes are mostly studied according to three technologies: differential privacy [21,22], Homomorphic Encryption (HE) [11,23], and Secure Multiparty Computation (SMC) [24,25].

In differential privacy, Zhao et al. [21] proposed a privacy-preserving collaborative deep learning model with unreliable participants that exploited a function mechanism to protect the local data privacy of participants with differential privacy. Cui et al. [22] proposed a differentially private decentralized federated learning using a modified Generative Adversarial Nets (GAN) model in the Internet of Things infrastructures, which satisfied differential privacy requirements while approximating the raw data to the best degree. Xu et al. [26] proposed a privacy-preserving federated learning approach (HybridAlpha) based on SMC protocol of function encryption and differential privacy. However, differential privacy can reduce the accuracy of the training data owing to the approximate results.

In HE, Phong et al. [11] presented a privacy-preserving deep learning model that used additive HE to realize asynchronous federated learning. However, it is impractical to leverage the same private key without considering the multi-key question. Xu et al. [27] exploited Yao's garbled circuits and additive homomorphic cryptosystems to protect the data privacy of federated learning while reducing the negative impact of irregular users. In SMC, Truex et al. [24] developed a privacy-preserving federated learning hybrid approach with differential privacy, SMC, and threshold HE. Agrawal et al. [25] presented a discretized training and prediction of deep neural networks with a customized secure two-party protocol. However, SMC has high communication overhead owing to more interactions.

To represent our motivation, we conduct a detailed analysis of the related works about SIMD, multi-key, and SGD.

Focusing on SIMD, Liu et al. [16] proposed an oblivious neural network (MiniONN) combining additive HE with SIMD, secret sharing, and oblivious transfer to support privacy-preserving prediction. However, it has a high computation cost for clients. Subsequently, Juvekar et al. [17] presented a secure neural network inference (GAZELLE) to realize lower latency than [16] and designed a lattice-based SIMD HE scheme with optimized encryption switching protocols. However, this model is unfit for non-lattice cryptosystems. Xie et al. [15] proposed a secure deep neural network inference scheme (BAYHENN) combining Bayesian deep learning and HE. BAYHENN adopted a vectorizable HE scheme with the encoding function and FHE, but it does not consider the security of the communication process owing to the activation function returned to the user in plaintext. These schemes containing SIMD technologies, which cannot be utilized in existing privacy-preserving federated learning schemes with partially HE, are leveraged in FHE or the cryptosystems based on the lattice. In addition, FHE cannot be used in large-scale federated learning or neural networks due to its ciphertext characteristics and multiple interactions.

Based on multi-key privacy-preserving schemes, Liu et al. [28] presented a distributed two-trapdoor public-key cryptosystem to reduce the associated key management costs. However, a mistake in the parameter setting was indicated by Li et al. [29], and the strong private key employed by the Cloud Service Provider (CSP) reduces security. Li et al. [30] proposed a multi-key privacy-preserving deep learning framework that employed Multi-key FHE (MK-FHE) and a double decryption mechanism. However, MK-FHE has more interactions and low efficiency. Ma et al. [31] developed a multi-key Privacy-preserving Deep Learning Model (PDLM) with a Distributed Two-trapdoor Public-Key Cryptosystem (DT-PKC). However, it has low efficiency and low classification accuracy. Chen et al. [32] proposed multi-key variants of FHE with packet ciphertexts applied to oblivious neural network inference. These schemes utilize DT-PKC or FHE, which have some limitations as described above. Moreover, the multi-key question is not considered in federated learning.

Focusing on SGD, Zhang et al. [33] proposed an Elastic Averaging SGD (EASGD) algorithm with both synchronous and asynchronous models. However, the algorithm requires all users to own the entire dataset and participate in the entire model training, regardless of the privacy issue or communication costs. To improve the convergence rate, momentum term with MGD was introduced [18,19]. Liu et al. [20] proposed Momentum Federated Learning (MFL) to accelerate convergence using MGD. However, this method does not consider the privacy issue. Some schemes [14,34] considered privacy in the model training process, but they only leverage SGD without considering momentum. To

solve the above problems, we design a multi-key and noninteractive privacy-preserving federated learning scheme with PSIMD and MGD methods.

## 3. Preliminaries

### 3.1. SMC

SMC is a set of cryptographic protocols proposed to realize secure computing that solves the problem of jointly calculating a function between untrusted participants. The technique was first introduced in Yao's millionaires' problem in 1982 [35].

Specifically, $P$ participants maintain the privacy information $x_1$, $x_2$, …, $x_P$ respectively, and jointly calculate and share the information of the given function $Y = F(x_1, x_2, \ldots, x_P)$ while preserving the privacy of the private key. The protocol can securely ensure that honest participants obtain the correct results.

### 3.2. (Lifted) ElGamal cryptosystem

The lifted ElGamal or ElGamal cryptosystem [36] consists of the following three algorithms.

1) **Key generation**: Given the large primes $p$ and $p'$ satisfying $p = 2p' + 1$. Choose a generator $h \in \mathbb{Z}_p^*$ such that $g = h^2$, and pick a private key $x$ randomly from $\mathbb{Z}_{p'}^* = \{1, 2, \ldots, p' - 1\}$. Then user computes the public key $y = g^x$.
2) **Encryption**: To encrypt a message $m$, it first chooses a number $r$ randomly from $\mathbb{Z}_{p'}^*$. Then a ciphertext $C = E(m) = (A, B)$ can be calculated as

$$A = g^r, B = g^m y^r (B = m y^r) \tag{1}$$

3) **Decryption**: On input a ciphertext $(A, B)$, the message $g^m(m)$ can be obtained:

$$g^m = \frac{B}{A^x} \left( m = \frac{B}{A^x} \right) \tag{2}$$

In formula (1)(2), the formula outside the brackets is the lifted ElGamal cryptosystem, and the formula inside the brackets is the ElGamal cryptosystem. In addition, because the message $m$ is small in size, $m$ can be extracted from $g^m$ using the lookup table method instead of the Pollard's Rho method [37].

**Homomorphic property**: The above two cryptosystems can satisfy the additive and multiplicative homomorphic properties respectively.

Given the ciphertexts $E(m_1)$ and $E(m_2)$ of two messages $m_1$ and $m_2$ with the random numbers $r_1$ and $r_2$, respectively. When $E(m_1) = (g^{r_1}, g^{m_1} y^{r_1})$ and $E(m_2) = (g^{r_2}, g^{m_2} y^{r_2})$, the followed natures are satisfied:

1) Additive homomorphism. The ciphertext $E(m_1 + m_2)$ can be calculated as follows:

$$E(m_1 + m_2) = (g^{r_1 + r_2}, g^{m_1 + m_2} y^{r_1 + r_2}) \tag{3}$$

2) The multiplication nature is: $E(m_1)^{m_2} = (g^{r_1 m_2}, g^{m_1 m_2} y^{r_1 m_2})$. When $E(m_1) = (g^{r_1}, m_1 y^{r_1})$ and $E(m_2) = (g^{r_2}, m_2 y^{r_2})$, the multiplicative homomorphism is: $E(m_1 m_2) = (g^{r_1 + r_2}, m_1 m_2 y^{r_1 + r_2})$.

### 3.3. Lifted Distributed ElGamal Cryptosystem

In early research, the distributed ElGamal cryptosystem leveraged in Refs. [38,39] can only be utilized on $P$ servers without collusion. According to ElGamal cryptosystem [36] and distributed ElGamal cryptosystem [38], we propose a Lifted Distributed ElGamal Cryptosystem (LDEC) with $P (P \neq 1)$ users substituting $P$ servers. The scheme can be described as follows:

**Key Generation**: Given a multiplication cyclic group $\mathbb{G}$ of prime order $p$ with a generator $g$, user $U_i$ chooses a private key $x_i$ randomly from $\mathbb{Z}_p^*$, and calculates and sends the public key $y_i = g^{x_i}$ to the server. Then the server computes the associated public key $y = \prod_{i=1}^{P} y_i = g^{\sum_{i=1}^{P} x_i}$.

**Encryption**: After obtaining a plaintext vector $m \in \mathbb{G}$ and the associated public key $y$, $P$ users share a random number $r$ jointly from $\mathbb{Z}_p^*$ with SMC [40], then user $U_i$ outputs a ciphertext $C = E(m) = (A, B)$, where $A = g^r$, $B = g^m \cdot y^r$.

**Decryption**: On inputs a ciphertext $(A, B)$, user $U_i$ generates a random number $r_i$, then calculates and sends $A^{r_i}$ to the server, which avoids the collusion question caused by sending $A^{x_i}$ to servers in Ref. [39]. In Ref. [39], the user information can be leaked by $P$ servers collusion, namely, $g^m = B/(A^{x_1} \cdot A^{x_2} \cdots A^{x_P}) = B \Big/ A^{\sum_{i=1}^{P} x_i}$, since we improve $A^{x_i}$ for $A^{r_i}$. After receiving $A^{r_i}$, the server computes $x_i = \sum_{j=1, j \neq i}^{P} x_j$ with SMC [40], then calculates and sends $A^{r_i} \cdot A^{\sum_{j=1, j \neq i}^{P} x_j}$ to user $U_i$. The message $g^m$ can be obtained by

$$g^m = \frac{B}{A^{x_i - r_i} A^{r_i} A^{\sum_{j=1, j \neq i}^{P} x_j}} \tag{4}$$

Utilizing the lookup table method, the plaintext $m$ can be output. Moreover, the scheme still satisfies the homomorphic property similar to the lifted ElGamal cryptosystem.

### 3.4. Federated learning

In federated learning, to realize the optimization weight $W^* = \arg\min L(W)$, the model needs to be trained to acquire the minimize cost function $L(W)$. Specifically, the trainer $T_k$ has a training dataset $D_k = \{(\chi_k, \mathfrak{y}_k)\}_{k=1}^{M}$, where $\chi_k$ and $\mathfrak{y}_k$ are the input data and the true label respectively, $M$ represents the number of trainers. The optimization cost function [10,41] is obtained:

$$\min L(W) \triangleq \sum_{i=1}^{M} \frac{N_i}{N} L_i(W) \tag{5}$$

where $W$ is the model parameter, $N_i$ is the sample size of the $i$th trainer, $N$ is the total sample size satisfying $N = \sum_{i=1}^{M} N_i$, and $L_i(W)$ is the local cost function of the $i$th trainer.

MGD contains the momentum term, which can accelerate the convergence rate compared to SGD [42]. Therefore, our scheme introduces the MGD to accelerate the convergence. Its process is followed.

1) Local Update: The local update rule can be written as

$$\begin{aligned} V_{t+1} &= \mu^* V_t + \nabla L(W_{t+1}) \\ W_{t+1} &= W_t - \eta V_{t+1} \end{aligned} \tag{6}$$

where $V_{t+1}$ is the $(t+1)$th momentum term with the same dimension $W_{t+1}$, $t$ is the iteration index, $\mu$ is the momentum factor, $\nabla$ represents the gradient, $\eta$ is the learning rate, and $W_t$ is the weight of the $t$th iteration. Thus, each trainer updates the momentums $V$ and weights $W$ by using the above rule and returns the updated momentums and weights to the server for aggregation.

2) Global Aggregation: After obtaining all the momentums and weights $W$ from the trainers, the aggregation server calculates the aggregation average of momentums and weights as follows:

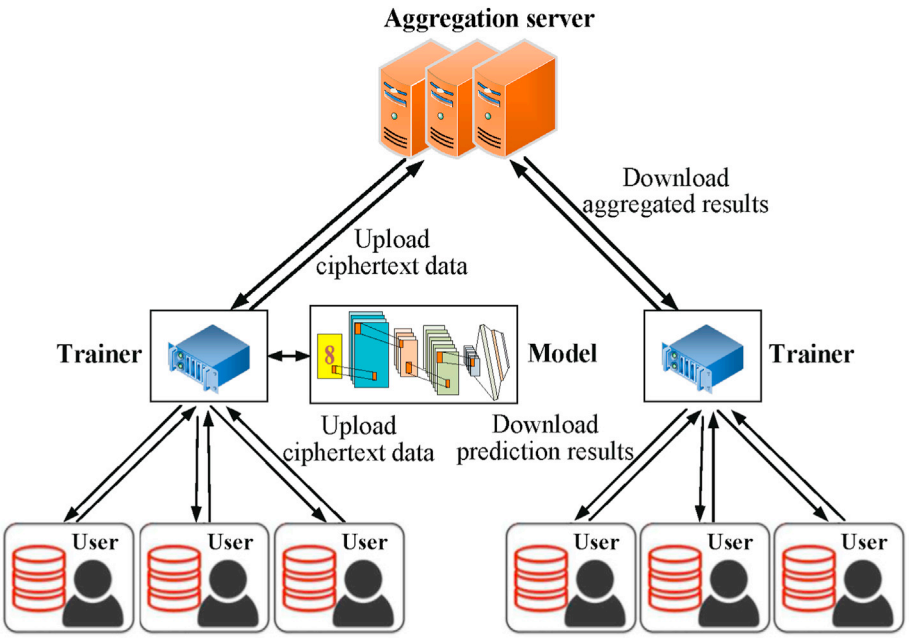



**Fig. 1.** Pepfl framework.

$$\overline{V_{t+1}} = \sum_{i=1}^{M} \frac{N_i}{N} \cdot V_{t+1}^i$$
$$\overline{W_{t+1}} = \sum_{i=1}^{M} \frac{N_i}{N} \cdot W_{t+1}^i \tag{7}$$

where $\overline{V_{t+1}}$ is the average of all momentums and $\overline{W_{t+1}}$ is the average of all weights from $M$ trainers.

## 4. System and threat models

In this section, we give an overview of PEPFL and describe the system and threat models considered in the framework. The architecture of PEPFL is illustrated in Fig. 1.

### 4.1. System model

In the system model, there are three entities: users, trainers, and the aggregation server. This is different from traditional two entities (users and the aggregation server) and supports users offline. The training model on the trainer leverages a Convolutional Neural Network (CNN) to realize training and prediction. Ultimately, the federated training framework can obtain the optimization training parameters and prediction results. All communication lines are guaranteed by the Secure Tunneling Protocol (STP).

**User**: In the PEPFL system, a user is an entity that can provide ciphertext data for training and prediction in the federated learning task. First, every user generates the public and private key pairs, and then sends the public key to the server for the associated public key. After obtaining the associated public key, the user encodes and encrypts its data with the associated public key, and then sends the ciphertext data to the trainer. Furthermore, the user can decrypt and decode the data to obtain the prediction results.

**Trainer**: As a machine learning entity, a trainer has a deep learning model to train the model parameters and can collect the encrypted data from the users in the local area. Specifically, the trainers can collaborate with the server to train the model using the federated learning method. Through multiple iterations, the trainers can acquire a well-trained model that can be leveraged for prediction with the users' demand.

**Server**: As a coordinator and an aggregator, the server can publish the associated public key and calculate the federated average with the aggregated weights from the trainers. Moreover, the server needs multiple iterations and cooperation to obtain the optimization model parameters in the training process. In our scheme, even if the server colludes with the other entities, it cannot obtain any data information from users or trainers.

### 4.2. Threat model

In our PEPFL, we consider that the aggregation server and trainers are honest-but-curious entities. Assume that a compromised internal entity from the server or trainer is an adversary $\mathcal{A}$, namely, it will honestly follow the protocols but also try to acquire the privacy information from the federated learning system. Besides, the user's private key and public key are generated by himself, which avoids the leakage of the private key. The threat model is followed:

1) An adversary $\mathcal{A}$ compromised by the server can try to retrieve sensitive information from the ciphertext weights and calculate the private keys from the public keys of the users. The adversary can collude with trainers or users.
2) An adversary $\mathcal{A}$ compromised by a trainer can acquire all encrypted data and parameters from the aggregation server or users. Besides, the adversary can collude with the server or users.
3) An adversary $\mathcal{A}$ compromised by a user tries to obtain sensitive information from encrypted training and prediction data. Moreover, multiple users can collude with each other, or users can collude with the trainers or the aggregation server.

## 5. Proposed scheme

In this section, we give a detailed description of PEPFL. First, users provide the encoded ciphertext data to the trainer in the local area. After collecting sufficient ciphertext data, the trainer trains the model with the

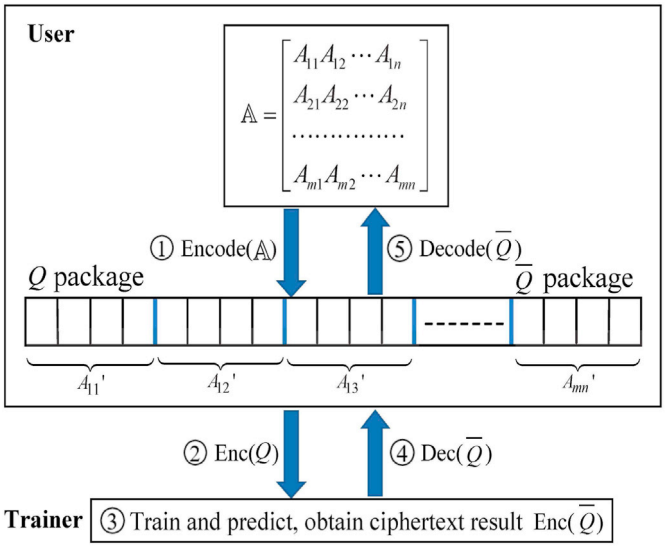



**Fig. 2.** PSIMD workflow.

initial weights from the server. Then the trainer uploads the updated ciphertext momentums and weights to the aggregation server. After the aggregation server finishes the aggregated average, the trainer downloads the aggregated results and iterates them to train the local update rule in the training model. Then the trainer uploads the updated ciphertext momentums and weights to the server again, and obtains the optimal model after some rounds of training. When obtaining a well-trained model, the user can predict the data he needs. In the following details, we introduce the PSIMD building block, Equation Test Block (ETB), and Homomorphic Selection Algorithm (HSA). Then, we describe the construction of PEPFL in detail.

### 5.1. PSIMD building block

In many prior work, FHE schemes [15,17,43] use SIMD to realize parallel computing, but they still incur high communication and computation costs due to the peculiarity of FHE. In partially homomorphic cryptosystems, existing schemes do not utilize SIMD. To improve the effectiveness of partially homomorphic encryption, we propose a PSIMD scheme to accelerate the model training and improve communication efficiency.

Owing to the high communication costs of image ciphertext in single data or vector data encryption method, to reduce the costs and accelerate the communication efficiency, we present a packed matrix homomorphic encryption scheme, namely, PSIMD, which involves encoding and decoding algorithms that are capable of packing the matrix into an extensional element in the real number $R$ with the floating-point storage method. The concreted process is followed:

For a matrix $\mathbb{A}$, first, we turn a number $A_{ij}\{1\leqslant i \leqslant m,\ 1\leqslant j \leqslant n\}$ of the matrix into the floating-point number $A_{ij}^{'}$ which represents the integer part $I$ and base exponent $e$, i.e., $(I, e)$ with the same $e$ by recalibration. The negative number becomes a positive number greater than the max number which is set as half of $n$ [44]. Due to the dataset $\{A_{ij}\}$ of the matrix $\mathbb{A}$ store sequentially in a device and a float number occupies four bytes space, to pack the matrix $\mathbb{A}$ into a plaintext packet $Q = A_{11}^{'} \parallel A_{12}^{'} \parallel \ldots \parallel A_{mn}^{'}$ for encrypting, it needs expending the $m*n$ matrix into $m*n*4$ bytes linear space as a package. Then user encrypts the package $Q$ as a whole and sends it the trainer, as shown in Fig. 2. The encoding algorithm is denoted as Encode($\mathbb{A}$), as shown in Algorithm 1.

**Algorithm 1:** PSIMD Encoding Algorithm

**Input**: $\mathbb{A}$
**Output**: $E_{pk}(Q)$
**Trainer:**
  Define matrix array $(\mathbb{A}_{ij})_{mn}$;
  Define float array $(\mathbb{A}'_{ij})_{mn}$;
  Encode($\mathbb{A}$):
  **for** $i = 1, 2, \ldots, m$ **do**
    **for** $j = 1, 2, \ldots, n$ **do**
      set $A_{ij}' \leftarrow A_{ij}$;
    **end for**
  **end for**
  Encrypt the packet $Q$ for $E_{pk}(Q)$.

After obtaining the ciphertext prediction result $E_{pk}(\overline{Q})$, the user decrypts the ciphertext $E_{pk}(\overline{Q})$ to obtain $\overline{Q}$. Then, the result package $\overline{Q}$ is divided into $m \times n$ numbers by every four bytes for a group, decoded, and then restored into a matrix $A'$. The decoding algorithm is shown in

**Algorithm 2:** PSIMD Decoding Algorithm

**Input**: $E_{pk}(\overline{Q})$
**Output**: $A'$
**Trainer:**
  Decrypt $E_{pk}(\overline{Q})$ for $\overline{Q}$;
  Define float array $(\mathbb{A}'_{ij})_{mn}$ in $\overline{Q}$ package;
  Define matrix array $(\mathbb{A}_{ij})_{mn}$;
  Decode($\overline{Q}$):
  **for** $i = 1, 2, \ldots, m$ **do**
    **for** $j = 1, 2, \ldots, n$ **do**
      set $A_{ij} \leftarrow A_{ij}'$;
    **end for**
  **end for**
  Restore the matrix $A'$ from $(A_{ij})_{mn}$.

Compared with the existing single plaintext encryption or vector encryption, this method greatly improves the encryption speed, reduces the ciphertext length and improves communication efficiency.

### 5.2. Equation Test Block

To test whether the two plaintexts $\overline{W}$ and $W^*$ of the ciphertext aggregation average $E_{pk}(\overline{W})$ and final global ciphertext model weights $E_{pk}(W^*)$ are equal, we design the following ETB building block.

Given two ciphertexts $E_{pk}(\overline{W}) = (g^{r_1}, g^{\overline{W}}y^{r_1})$ and $E_{pk}(W^*) = (g^{r_2}, g^{W^*}y^{r_2})$, to test if $\overline{W}$ and $W^*$ are equal, we compare $E_{pk}(\overline{W})$ and $E_{pk}(W^*)$ as follows:

$$\left(\frac{A_1}{A_2}, \frac{B_1}{B_2}\right) = (g^{r_1-r_2}, g^{\overline{W}-W^*}y^{r_1-r_2}) \tag{8}$$

After decrypting the above formula with the lifted ElGamal cryptosystem, if $g^{\overline{W}-W^*} = 1$, that is, $\overline{W} - W^* = 0$, then two plaintexts $\overline{W}$ and $W^*$ are equal plaintexts. Otherwise, two plaintexts are not equal.

In the decryption process, the associated private key $sk = \sum_{i=1}^{P} x_i = \sum_{i=1}^{P} sk_i$, which can be obtained from SMC [45], is needed.

### 5.3. Homomorphic Selection Algorithm

Since the lifted ElGamal or ElGamal cryptosystem supports additive or multiplicative homomorphism respectively, it is necessary to select an appropriate cryptosystem for homomorphism calculation in the ciphertext training process of the trainers. Therefore, we design the HSA algorithm that satisfies both additive homomorphism and multiplicative homomorphism as shown in Algorithm 3, which is applied in the CNN ciphertext training process. In the PEPFL, we adopt the proposed LDEC.

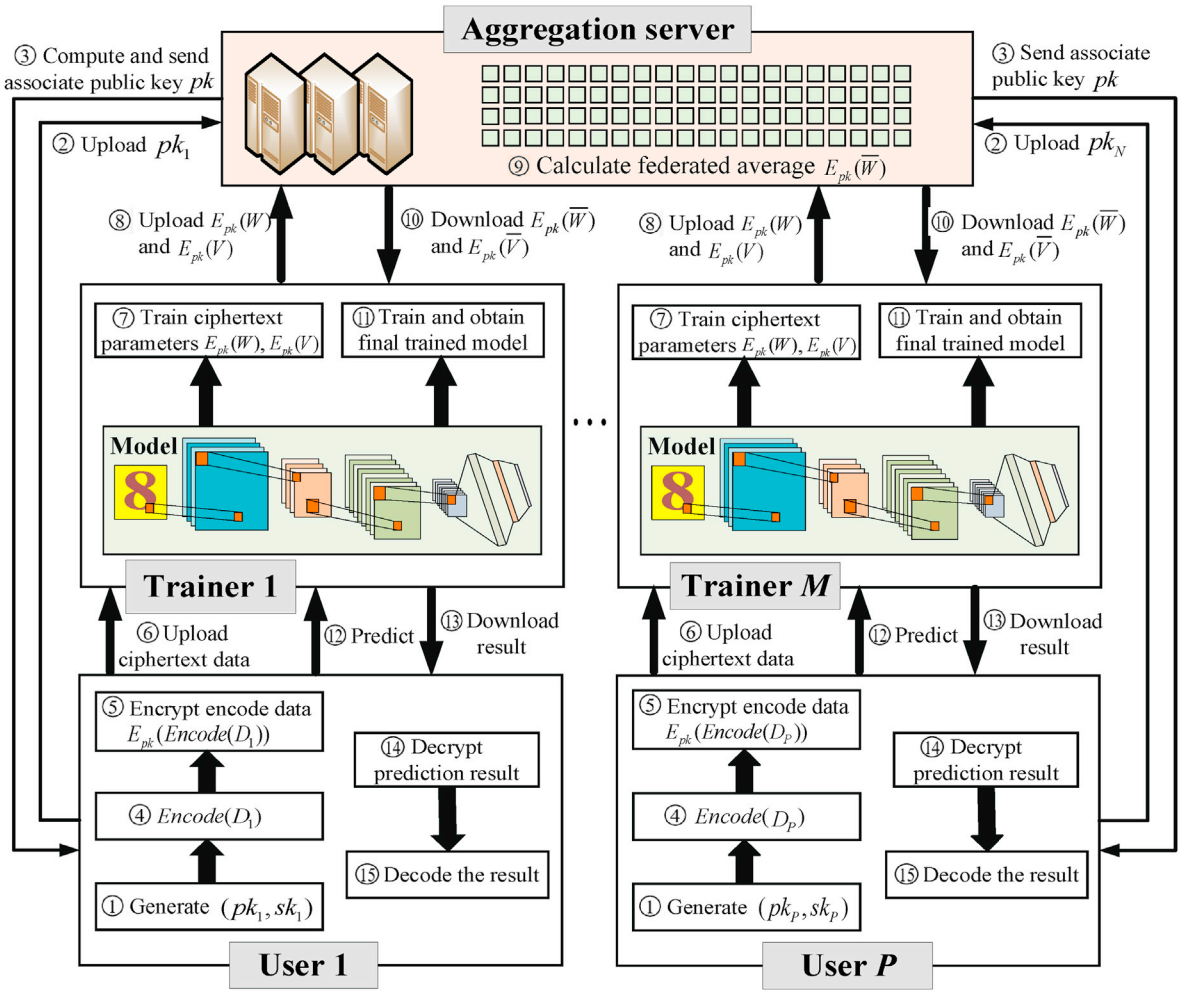



**Fig. 3.** PEPFL workflow.

### 5.4. Construction of PEPFL

In this subsection, we describe the workflow of the scheme in Fig. 3. According to the workflow, the steps of our scheme include the following phases: the initialization phase, the encoding and encryption phase, the model training phase, and the prediction phase.

**Initialization phase**: Firstly, user $U_i$ generates public and private key pairs ($pk_i$, $sk_i$). Then user $U_i$ uploads the public key $pk_i$ to the aggregation server. The server collects the public keys from $P$ users, and then calculates and publishes the associated public key $pk = \prod_{i=1}^{P} pk_i = g^{\sum_{i=1}^{P} x_i}$.

**Encoding and encryption phase**: After obtaining the associated public key $pk$, user $U_i$ encodes his data $D_i$ for $Encode(D_i)$ with the PSIMD encoding algorithm and encrypts the encoding data $Encode(D_i)$ for $E_{pk}(Encode(D_i))$ with lifted distributed ElGamal encryption. Then user $U_i$ sends the ciphertext $E_{pk}(Encode(D_i))$ to trainer $T_i$.

Specifically, firstly, user $U_i$ preprocesses and segments the figure data $D_i$ for the matrix data $\mathbb{A}_k\{k \in \{1, 2, \ldots, s\}\}$, where $s$ is the number of matrices, namely, the user divides the figure of an $a \times b$ matrix into multiple small matrices $\mathbb{A}_k = \{A_{mn}^{(k)}\}\{1 \leqslant m \leqslant a, 1 \leqslant n \leqslant b\}$. Secondly, user $U_i$ encodes every $\mathbb{A}_k$ for linear $Q_k$ with $A_{mn}^{(k)}$ which is a four byte floating number. Then, user $U_i$ encrypts the linear $Q_k$ as a whole for $E_{pk}(Q_k)$ with the associated public key $pk$. Owing to NPMML scheme [14] or other partially homomorphic cryptosystem only encrypting a single element $\{A_{mn}\}$ or a vector of a matrix in sequence, while our scheme encrypts the encoded package, the efficiency of our PSIMD package method is superior to that of [14] or other partially homomorphic cryptosystems. Besides, user $U_i$ encrypts the label $\mathfrak{y}_i$ for $E_{pk}(\mathfrak{y}_i)$.

When obtaining $\{E_{pk}(Q_1)\|E_{pk}(\mathfrak{y}_1), E_{pk}(Q_2)\|E_{pk}(\mathfrak{y}_2), \ldots, E_{pk}(Q_s)\|E_{pk}(\mathfrak{y}_s)\}$, user $U_i$ uploads these ciphertext data to trainer $T_i$ in local domain via STP secure channel.

**Model training phase**: After getting ciphertext data, trainer $T_i$ performs the ciphertext training on the CNN model, which can acquire the ciphertext momentums $E_{pk}(V)$ and the ciphertext weights $E_{pk}(W)$ from the CNN ciphertext training process. The ciphertext training process referred to other papers [46,47] with the multiplicative homomorphism of ElGamal cryptosystem, and it is not the focus of our research. In terms of the MGD algorithm, we can update the ciphertext momentums and weights by using a secure federated average algorithm.

More specifically, first, according to the local update rule, we can calculate the ciphertext momentums $E_{pk}(V_{t+1}^i)$ and weights $E_{pk}(W_{t+1}^i)$ for trainer $T_i$ according to formula (6) with the additive homomorphism and multiplication nature of the lifted ElGamal cryptosystem:

$$\begin{aligned} E_{pk}(V_{t+1}^i) &= E_{pk}(\mu \cdot \overline{V_t^i} + \nabla L(W_{t+1}^i)) \\ &= E_{pk}(\overline{V_t^i})^{\mu} \cdot E_{pk}(\nabla L(W_{t+1}^i)) \end{aligned} \tag{9}$$

From the above formula, we can gain the ciphertext weights $E_{pk}(W_{t+1}^i)$ with the same natures and cryptosystem on trainer $T_i$:

$$\begin{aligned} E_{pk}(W_{t+1}^i) &= E_{pk}(\overline{W_t^i} - \eta V_{t+1}^i) \\ &= E_{pk}(\overline{W_t^i}) \cdot E_{pk}(V_{t+1}^i)^{-\eta} \end{aligned} \tag{10}$$

After obtaining all $E_{pk}(V_{t+1}^i)$, $E_{pk}(W_{t+1}^i)$ from $M$ trainers, the aggregation server computes the ciphertext federated averages $E_{pk}(\overline{V_{t+1}})$, $E_{pk}(\overline{W_{t+1}})$ as follows.

$$E_{pk}(\overline{V_{t+1}}) = \sum_{i=1}^{M} \frac{N_i}{N} E_{pk}(V_{t+1}^i) \tag{11}$$

$$E_{pk}(\overline{W_{t+1}}) = \sum_{i=1}^{M} \frac{N_i}{N} E_{pk}(W_{t+1}^i) \tag{12}$$

After continuous iterations of local update and global aggregation, the server can obtain the optimal ciphertext weights $E_{pk}(W^*)$. The secure federated average algorithm is called the Privacy-preserving Federated Learning Protocol (PFLP), as shown in Algorithm 4, where $f$ is the maximum number of rounds. Compared to the traditional federated averaging algorithm [10], our PFLP improves the convergence rate of universal privacy-preserving federated learning.

In the protocol, the optimal ciphertext weights are estimated by comparing $E_{pk}(\overline{W_{t+1}})$ and $E_{pk}(W^*)$ through judging if $\overline{W_{t+1}}$ and $W^*$ are equal according to ETB, or if the number of rounds reaches the maximum iterations value. The optimal ciphertext weights are trained in the Ciphertext Aggregation Algorithm (CAA), as shown in Algorithm 5.

**Prediction phase**: After the trainers obtain the well-trained model, the users can realize the prediction on the trainers. First, user $U_i$ uploads the encrypted prediction data $E_{pk}(\mathbb{Q}_i)$ to a trainer, where $\mathbb{Q}_i$ is the encoded data. The trainer predicts the ciphertext data $E_{pk}(\mathbb{Q}_i)$ to attain the ciphertext prediction result $E_{pk}(\Re_i)$. Then, user $U_i$ downloads the ciphertext result $E_{pk}(\Re_i)$ and decrypts it for $\Re_i$ with the lifted distributed ElGamal decryption algorithm. Finally, user $U_i$ restores $\Re_i$ for the corresponding matrix by using the PSIMD decoding algorithm.

### 5.5. Decryption-Prevention Protocol

In the distributed ElGamal cryptosystem [38,39], each user $U_i$ sends $A^{x_i}$ to the honest-but-curious server. When the server or an adversary obtains all $A^{x_i}$ from every user or server, it can perform the calculation $G = A^{x_1} \cdot A^{x_2} \cdot \ldots \cdot A^{x_p} = g^{r \cdot \sum_{i=1}^{p} x_i}$ with the ciphertext $C = (g^r, g^m y^r)$, the server or the adversary can compute $g^m = g^m y^r / G$, and then it can obtain the plaintext $m$. To prevent decryption or attack, we propose a Decryption-Prevention Protocol (DPP), as shown in Algorithm 6. The algorithm avoids the collusion question of multiple users or servers.

**Algorithm 3:** Homomorphic Selection Algorithm(HSA)

**Input**: $m_1, m_2$
**Output**: $E(m_1 + m_2), E(m_1 m_2)$
Judging training formula:
  **if** $m_1 + m_2$ is needed in training process
  **then** lifted ElGamal cryptosystem;
    given the ciphertext $E(m_1)$, $E(m_2)$,
    calculate $E(m_1 + m_2)$;
  **end if**
  **if** $m_1 \cdot m_2$ is needed in training process
  **then** ElGamal cryptosystem;
    given the ciphertext $E(m_1)$, $E(m_2)$,
    calculate $E(m_1 m_2)$.
  **end if**

**Algorithm 4:** Privacy-preserving Federated Learning Protocol (PFL)

**Input**: ciphertext model parameters $E_{pk}(W_i)$, $E_{pk}(V_i)$
**Output**: the final global ciphertext model weight $E_{pk}(W^*)$
**Server:**
  Initialize the final global model weight $W^*$, the initial weight $\overline{W_i}(0)$, the initial momentum $\overline{V_i}(0)$;
  Broadcast the initial parameters $\overline{W_i}(0)$, $\overline{V_i}(0)$ to every trainer;
  **for** each round $t$=1, 2, …, $f$ **do**
    **for** $i$=1, 2, …, $M$ in parallel **do**
      Each trainer $i$ performs local update according to the following trainer's algorithm in PFL:
      Update $E_{pk}(V_{t+1}^i) \leftarrow E_{pk}(i, \overline{V_t})$;
        $E_{pk}(W_{t+1}^i) \leftarrow E_{pk}(i, \overline{W_t})$;
      Send $E_{pk}(V_{t+1}^i)$, $E_{pk}(W_{t+1}^i)$ to Server.
    **end for**
    Calculate federated aggregation average:
      $E_{pk}(\overline{V_{t+1}}) = \sum_{i=1}^{M} \frac{N_i}{N} E_{pk}(V_{t+1}^i)$;
      $E_{pk}(\overline{W_{t+1}}) = \sum_{i=1}^{M} \frac{N_i}{N} E_{pk}(W_{t+1}^i)$;
    Send $E_{pk}(\overline{V_{t+1}})$, $E_{pk}(\overline{W_{t+1}})$ to Trainer.
  **end for**

**Trainer** $T_i$**:**
  Trainer update $(i, E_{pk}(\overline{W_{t+1}}))$:
  Obtain the newest ciphertext model parameters $(i, E_{pk}(\overline{V_t}))$, $(i, E_{pk}(\overline{W_t}))$;
  **for** the number of iterations $i$=1, 2, …, $\mathcal{S}$ **do**
    Divide the dataset $D_i$ into the batch size $K$ randomly;
    **for** the batch number $j$=1, 2, …, $\frac{D_i}{K}$ **do**
      Calculate the ciphertext gradient $E_{pk}(\nabla L(W_{t+1}^i))$;
      Update the ciphertext momentum:
      $E_{pk}(V_{t+1}^i) = E_{pk}(\overline{V_t})^{\mu} \cdot E_{pk}(\nabla L(W_{t+1}^i))$;
      Update the ciphertext weight:
      $E_{pk}(W_{t+1}^i) = E_{pk}(\overline{W_t}) \cdot E_{pk}({V_{t+1}}^i)^{-\eta}$;
    **end for**
  **end for**
  Obtain local ciphertext model parameters $E_{pk}(V_{t+1}^i)$, $E_{pk}(W_{t+1}^i)$;
  **if** $E_{pk}(W_{t+1})$ is not an aggregation value judged by CAA in Algorithm 5;
  **then** send $E_{pk}(V_{t+1})$, $E_{pk}(W_{t+1})$ to Server;
  **else**
  Quit the update and the training is over.
  **end if**

**Table 1**
Communication cost.

| Phase | Participant | Communication cost |
|---|---|---|
| Initialization | user | $O(\log p)$ |
| | server | $O(\log p)$ |
| Encoding and encryption | user | $O(2s\log p)$ |
| Local update | trainer | $O(2u\log p)$ |
| Global aggregation | server | $O(2u\log p)$ |
| Prediction | user | $O(2s\log p)$ |
| | trainer | $O(\log p)$ |

$s$ is the number of matrices; $p$ is the big prime; $u$ is the number of the weight encoding packets.

**Algorithm 5:** Ciphertext Aggregation Algorithm (CAA)

**Input**: a global ciphertext model parameter $E_{pk}(\overline{W_{t+1}})$, the final global ciphertext model parameter $E_{pk}(W^*)$, the maximum round $f$
**Output**: The final global ciphertext model parameter $E_{pk}(W^*)$

**Server:**
  **if** the round $t \neq f$
  **then** continue training the PFL.
  **else** obtain the ciphertext average parameter $E_{pk}(\overline{W_{t+1}})$ and retreat from the training.
  **end if**

**Trainer:**
  After obtaining the ciphertext average weight $E_{pk}(\overline{W_{t+1}})$, in terms of ETB, judging:
  **if** $\overline{W_{t+1}} \neq W^*$
  **then** continue training the model on the CNN;
  **else** obtain the optimal ciphertext weight $E_{pk}(\overline{W_{t+1}})$.
  **end if**
  Predict the data from users according to the optimal ciphertext model.

**Algorithm 6:** Decryption-prevention Protocol

**Input**: $A^{x_i}$
**Output**: $g^m$

**Server or Adversary:**
  User $U_i$ sends $A^{x_i}$;
  **if** the number of users $P > 1$
  **then** it cannot decrypt utilizing lifted ElGamal cryptosystem.
  **else** it only can decrypt $D(C)$ exploiting LDEC.
  **end if**
  Output $g^m$.

## 6. Security analysis

The security of PEPFL is guaranteed by the ElGamal cryptosystem and LDEC under the Decisional Diffie Hellman (DDH) [48] assumption. The ElGamal cryptosystem [36] and threshold cryptosystems [49] are semantically secure provided that the DDH problem is hard.

In this section, we first prove the semantic security and the other security of LDEC. Then, we expound on the security of input privay, model privacy and inference result privacy.

**Theorem 1**. (*Semantic Security*). *If the ElGamal cryptosystem is semantically secure, then the LDEC is semantically secure.*

Proof. Knowing $y = g^{\sum_{i=1}^{P} x_i}$ and $A = g^r$, an adversary cannot acquire $y^r = g^{r\sum_{i=1}^{P} x_i}$; that is, an adversary cannot determine $g^c = g^{r\sum_{i=1}^{P} x_i}$ and cannot compute $g^m$ from $B = g^m y^r$ by multiplying the inversion $y^{-r}$, where $c = r\sum_{i=1}^{P} x_i$. Then, it cannot obtain $m$ from $g^m$. Due to the DDH and discrete logarithm hard problem, in terms of the semantic security of ElGamal, the LDEC is semantically secure.

**Theorem 2**. (*Against Collusion and Adversary Security*). *If the encryption schemes ElGamal and LDEC are semantically secure, then the security can be guaranteed under collusion or hacking attacks.*

Proof. In the distributed ElGamal cryptosystem [39], each server outputs $A_i = A^{x_i}$, and the plaintext can be obtained by $g^m = B \Big/ \prod_{i=1}^{P} A^{x_i}$. When a server colludes with an external adversary or an adversary eavesdrops the ciphertext data $B$ and all $A_i$, the adversary can obtain the plaintext $m$ from these ciphertext. To avoid collusion or hacking attacks, our proposed LDEC improves $A^{x_i}$ for $A^{r_i}$, and each user replacing each server computes $A^{x_i - r_i}$ with the private key $x_i$ to conduct decryption. Moreover, fewer than $P-1$ users collusion cannot decrypt the ciphertext data, and thus, security can still be guaranteed. When an adversary eavesdrops on these ciphertext informations $B, A^{r_i}, A^{r_i} A^{\sum_{j=1, j\neq i}^{P} x_j}$, he cannot gain the value $g^m$ and then cannot acquire the plaintext $m$.

**Theorem 3**. (*Input Privacy*). *If the LDEC scheme is semantically secure, then the input privacy of PEPFL can be guaranteed.*

Proof. After encoding and encrypting the plaintext data to obtain $\{E_{pk}(Q_1)\|E_{pk}(\mathfrak{v}_1),\ E_{pk}(Q_2)\|E_{pk}(\mathfrak{v}_2),\ \ldots,\ E_{pk}(Q_s)\|E_{pk}(\mathfrak{v}_s)\}$, user $U_i$ sends these ciphertexts to a trainer. Because LDEC is semantically secure, the input security for trainers is guaranteed: (1) Fewer than $P-1$ users collusion cannot obtain trusted users' information because there is no private key to decrypt. (2) An adversary or participant cannot obtain any information from the ciphertext except for the user himself. Therefore, the input privacy can be guaranteed.

**Theorem 4**. (*Model Privacy*). *If the LDEC scheme is semantically secure, then the model privacy of PEPFL can be guaranteed.*

Proof. In the PFLP protocol, all data are trained under ciphertext, which protects confidentiality in terms of the semantic security of LDEC. To formalize the security of model privacy, we prove the security of the local update and global aggregation according to the ideal world and the real-world model [50].

**Lemma 1**. *Local update (@Trainer) is secure against a semi-honest adversary $\mathcal{A}_{trainer}$ in the PFLP protocol.*

*Proof.* A challenge trainer can securely communicate with a semi-honest adversary $\mathcal{A}_{trainer}$. When constructing a simulator $S$, the simulator $S$ replacing the trainer is built to interact with the adversary $\mathcal{A}_{trainer}$. In the real world, the view of $\mathcal{A}_{trainer}$ is

$$V_{Real} = \{[\overline{V_t}]_{pk}, [\overline{W_t}]_{pk}, [\nabla L(W_{t+1})]_{pk}, [V_{t+1}^i]_{pk}, [W_{t+1}^i]_{pk}\}$$

In the ideal world, the view of $\mathcal{A}_{trainer}$ is

$$V_{Ideal} = \{[\overline{r_{11}}]_{pk}, [\overline{r_1}]_{pk}, [r_{12}]_{pk}, [r_{21}]_{pk}, [r_{11}]_{pk}\}$$

where the random numbers $\overline{r_{11}}, \overline{r_1}, r_{12}, r_{21}, r_{11} \in R$.

According to the semantic security of LDEC, the adversary $\mathcal{A}_{trainer}$ cannot distinguish the ideal world from the real world:

$$\{IDEAL^{PFLP}_{\mathcal{A}_{trainer},S}(V_{Ideal})\} \approx^c \{REAL^{PFLP}_{\mathcal{A}_{trainer},trainer}(V_{Real})\}$$

**Lemma 2**. *Global aggregation (@Server) is secure against a semi-honest adversary $\mathcal{A}_{Server}$ in the PFLP protocol.*

Proof. In the global aggregation process, a semi-honest adversary $\mathcal{A}_{Server}$ interacts with the server, and its view in the real world is

$$V_{Real}{}' = \{[\overline{V_t}]_{pk}, [\overline{W_t}]_{pk}, [V_{t+1}^i]_{pk}, [W_{t+1}^i]_{pk}, [\overline{V_{t+1}}]_{pk}, [\overline{W_{t+1}}]_{pk}\}$$

In the ideal world, a simulator $S$ replacing the server generates the same number of random ciphertext. The view of $\mathcal{A}_{Server}$ is

**Table 2**
Computation cost.

| Phase | Computation cost |
|---|---|
| Initialization | $(P-1)$Mul |
| Encoding and encryption | $s$(Ecod + 2Enc) |
| Local update | $l(\mu$Exp + Gra + 2Enc) |
| Global aggregation | $Ml$(Div + Mul + Enc) |
| Prediction | $s$(Ecod + 2Enc) |
| Prediction result | $s$(Dcod + Dec) |

$P$ is the number of users; $s$ is the number of matrices; $l$ is the number of weights and momentums; $\mu$ is the momentum factor; $M$ is the number of trainers.

**Table 3**
Functionality comparison.

| Scheme | Cryptographic techniques | Servers' number | Parameter privacy | Aggregation result privacy | Multi-key | MGD | PSIMD/ SIMD |
|---|---|---|---|---|---|---|---|
| PPDL [11] | Paillier, LWE | 1 | ✓ | ✓ | × | × | ✓ |
| PPFDL [12] | Paillier | 2 | ✓ | ✓ | ✓ | × | × |
| VerifyNet [13] | Homomorphic hash, Pseudorandom function, Secret sharing | 2 | ✓ | ✓ | ✓ | × | × |
| SecProbe [21] | Differential privacy, Function mechanism | 1 | ✓ | × | × | × | × |
| Hybridalpha [26] | MPC, Differential privacy | 2 | ✓ | ✓ | ✓ | × | × |
| HAPPFL [24] | Threshold Paillier, MPC, Differential privacy | 1 | ✓ | ✓ | ✓ | × | × |
| MFL [20] | ~ | 1 | × | × | × | ✓ | × |
| PEPFL | Lifted distributed ElGamal | 1 | ✓ | ✓ | ✓ | ✓ | ✓ |

$V'_{Ideal} = \{[\overline{r_a}]_{pk}, [\overline{r_b}]_{pk}, [r_{a1}]_{pk}, [r_{b1}]_{pk}, [\overline{r_{a1}}]_{pk}, [\overline{r_{b1}}]_{pk}\}$

where the random numbers $\overline{r_a}, \overline{r_b}, r_{a1}, r_{b1}, \overline{r_{a1}}, \overline{r_{b1}} \in R$.

Due to the semantic security of LDEC, the adversary $\mathcal{A}_{Server}$ cannot distinguish between the ideal world and the real world:

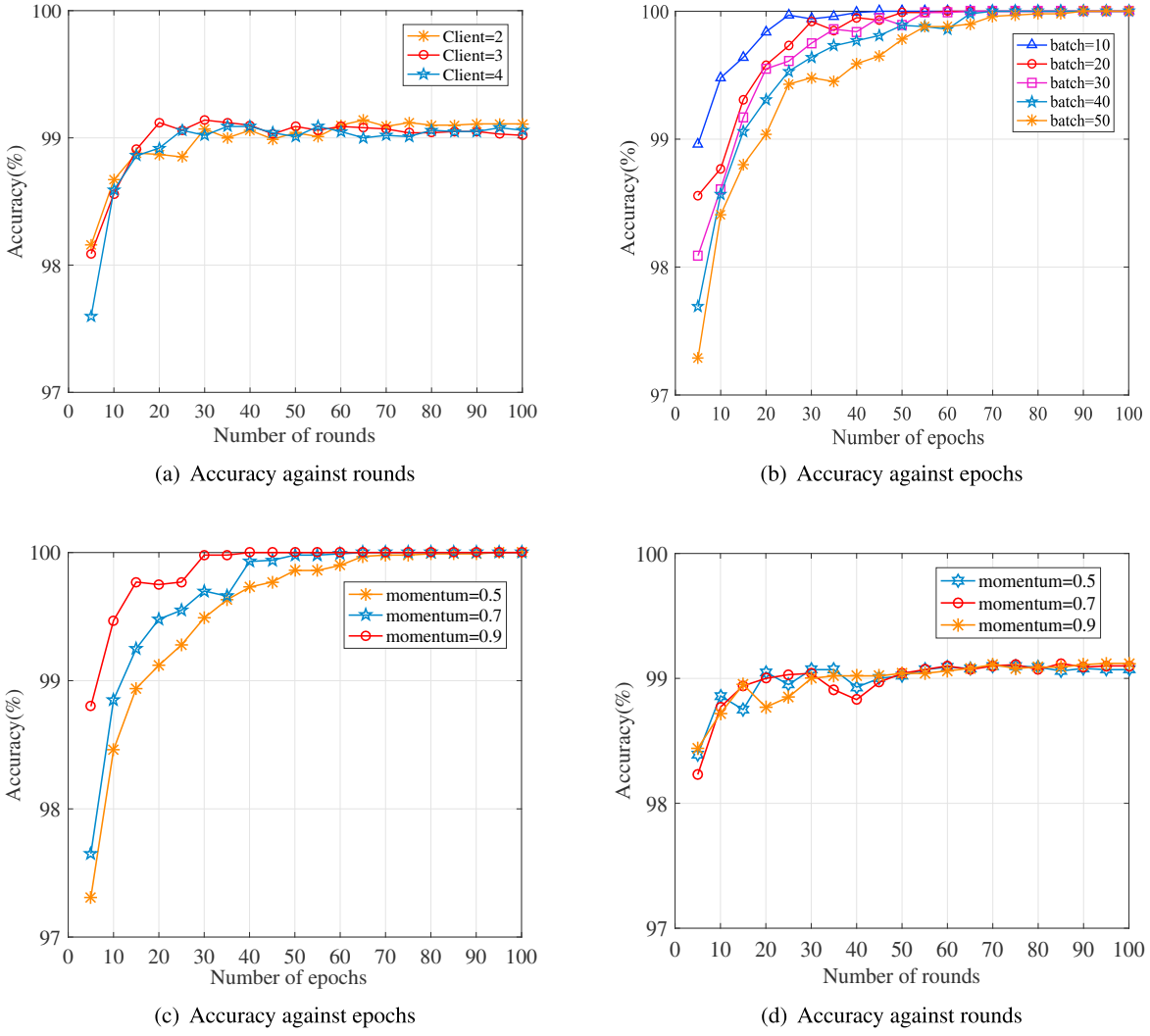
$$\{IDEAL^{PFLP}_{\mathcal{A}_{Server},S}(V'_{Ideal})\} \approx^{c} \{REAL^{PFLP}_{\mathcal{A}_{Server},Server}(V'_{Real})\}$$

**Theorem 5**. (*Inference Result Privacy*). *If the LDEC scheme is semantically secure, then the inference result privacy of PEPFL can be guaranteed.*

Proof. In the prediction phase, user $U_i$ sends the prediction ciphertext $E_{pk}(\mathbb{Q}_i)$ to a trainer, and the trainer uses the well-trained model to train and obtain the ciphertext prediction result $E_{pk}(\Re_i)$. All the data on the trainer are ciphertext, so the ciphertext result is returned to user $U_i$. Due to the semantic security of LDEC, inference result privacy can be guaranteed.

## 7. Performance evaluations

In this section, we analyze and compare the performance of PEPFL. First, we discuss the communication and computation costs of PEPFL. Additionally, we evaluate the efficiency and accuracy of our scheme.



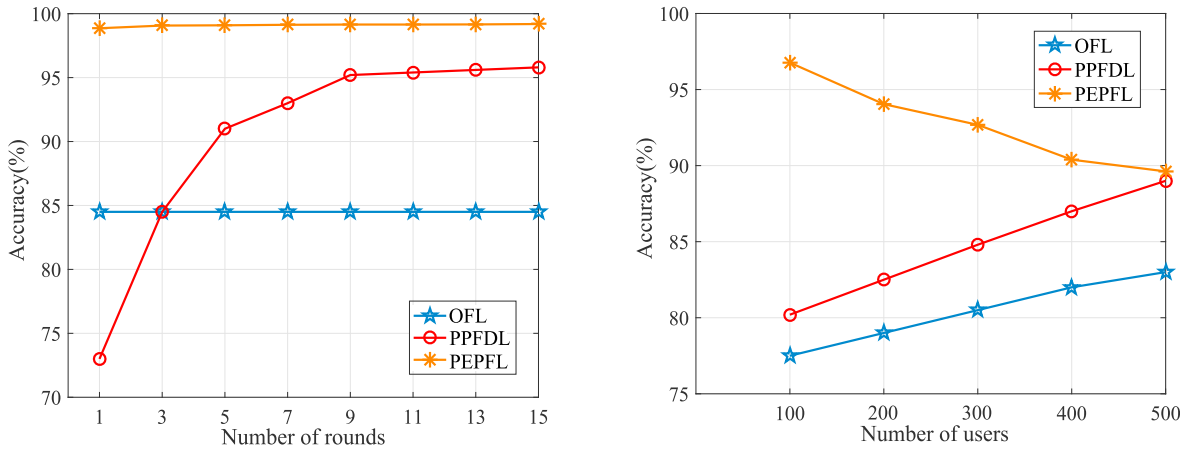

(a) Accuracy against rounds

(b) Accuracy against epochs

(c) Accuracy against epochs

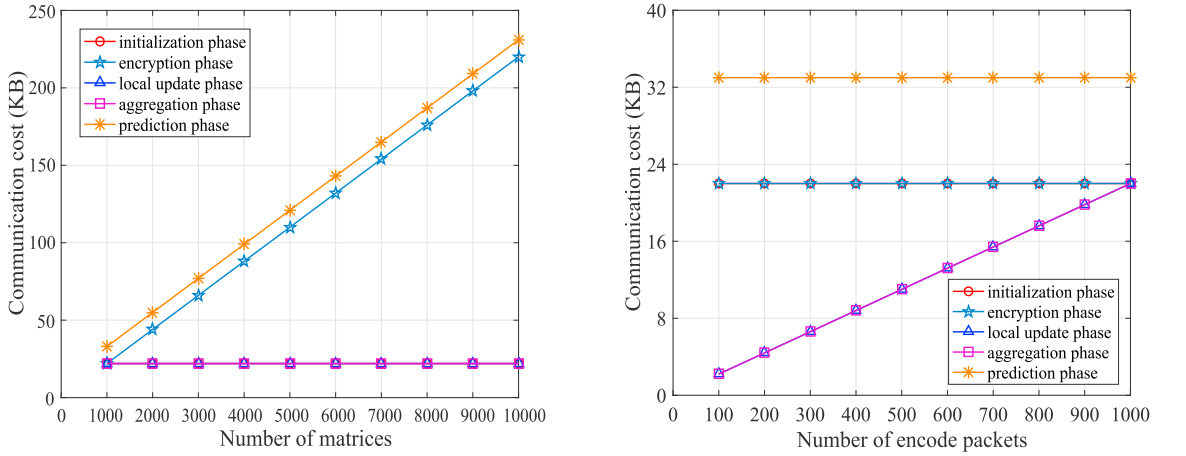
(d) Accuracy against rounds

**Fig. 4.** Classification Accuracy with rounds and epochs.




(a) Accuracy comparison against the number of rounds (b) Accuracy comparison against the number of users

**Fig. 5.** Accuracy comparison.

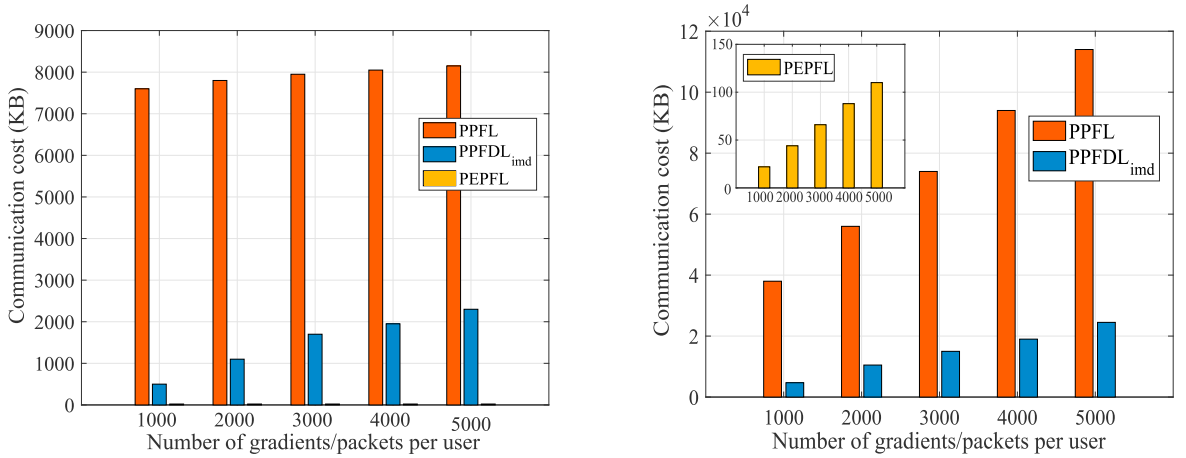



(a) Communication cost against the number of matrices (b) Communication cost against the number of encode packets

**Fig. 6.** Communication cost in different phases.

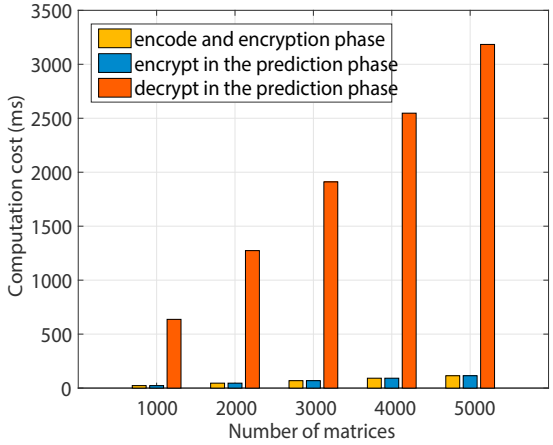
### 7.1. Complexity analysis

**Communication cost.** In the initialization phase, user $U_i$ sends $O(\log p)$ communication costs to the server. After computing the associated public key $pk$, the server takes $O(\log p)$ costs for every user. In the encoding and encryption phase, user $U_i$ sends $O(2s \log p)$ costs to trainer $T_i$. In the local update phase, trainer $T_i$ generates $O(2u \log p)$ costs for the server, where $u$ is the number of the weight encoding packets. After the

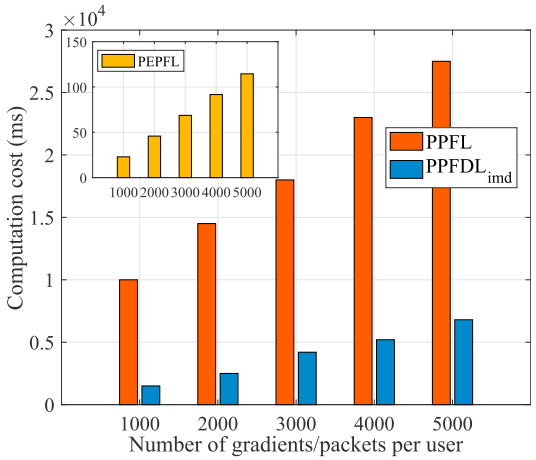



(a) Comparison of communication cost in user side (b) Comparison of communication cost in server side

**Fig. 7.** Communication cost.




(a) Computation cost in different phases




(b) Comparison of computation cost in user side

**Fig. 8.** Computation cost.

global aggregation average, the server imposes $O(2u \log p)$ costs on every trainer. When the trainer obtains the well-trained model, users can predict the result they require. In the prediction phase, user $U_i$ uploads $O(2s \log p)$ costs to a trainer who conducts the well-trained model. After predicting the ciphertext result, the trainer sends $O(\log p)$ costs to the predicted user. The summary of communication costs is shown in Table 1.

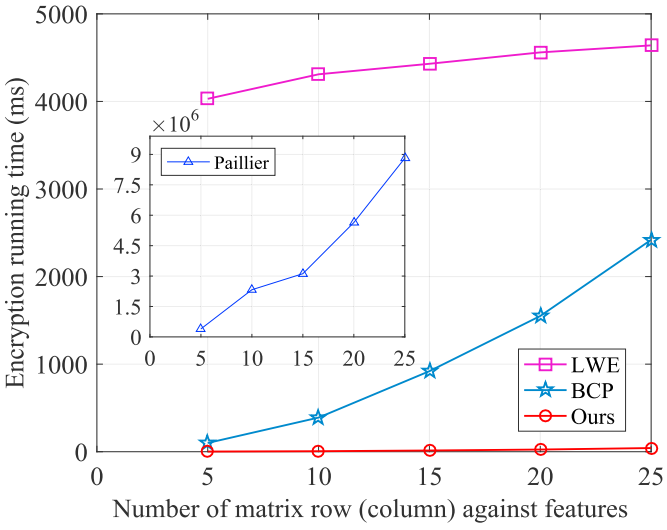
**Computation cost.** To simplify the notation, we denote point multiplication/division as Mul/Div, encoding/decoding as Ecod/Dcod, encryption/decryption as Enc/Dec, an exponent as Exp, and a gradient as Gra. In the initialization phase, the server takes $(P-1)$Mul costs to compute the associated public key. In the encoding and encryption phase, user $U_i$ costs $s$(Ecod + 2Enc) to compute the ciphertext data. After training and obtaining the ciphertext weights $E(W)$ in the local update process, trainer $T_i$ calculates the ciphertext momentums at $l(\mu\text{Exp} + \text{Gra} + 2\text{Enc})$ costs in each round, where $l$ is the number of weights and momentums. When the server receives all the weights and momentums from the trainers, it costs $Ml$(Div + Mul + Enc) to compute the ciphertext federated aggregation average. In the prediction phase, user $U_i$ takes $s$(Ecod + 2Enc) costs to compute the ciphertext data $E_{pk}(\mathbb{Q}_i)$ for the prediction. After obtaining the ciphertext predicted result, user $U_i$ costs $s$(Dcod + Dec) to obtain the plaintext prediction result. Table 2 shows the summary of computation costs.

### 7.2. Functionality

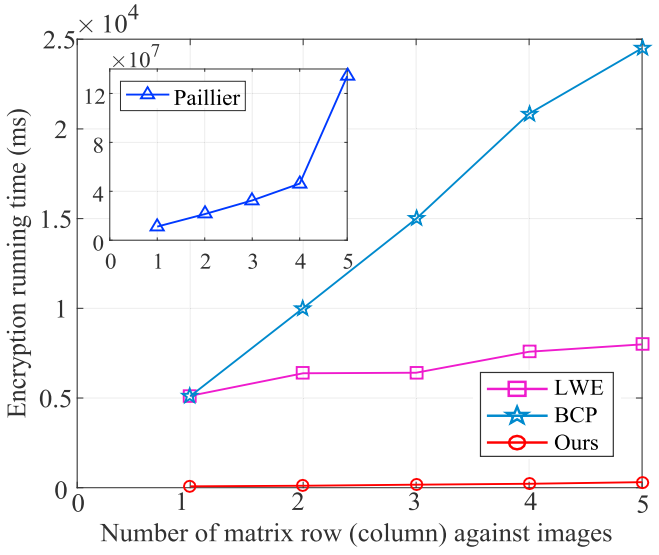
In this subsection, we compare the functional advantages of the proposed method with those of the previous schemes, as shown in Table 3. In Table 3, PPDL [11] proposed privacy-preserving deep learning models based on Paillier and Learning with Errors (LWE). However, they do not consider multi-key and MGD questions. PPFDL [12] and VerifyNet [13] achieved advantages with irregular users or verifiability, respectively. However, they do not consider MGD and SIMD. SecProbe [21] also considered irregular users in the federated training process. However, it does not guarantee the privacy of the aggregated result or support multi-key, MGD, and SIMD. Hybridalpha [26] and HAPPFL [24] protected the privacy of parameters and results, but they are not concerned with MGD and SIMD. MFL [20] improved MGD, but does not consider other functionalities. Our PEPFL scheme considers all the functionalities, as shown in Table 3.

### 7.3. Experimental analysis

To illustrate the performance and accuracy, PEPFL is simulated using a computer with an Intel(R) Core(TM) i7-7700HQ CPU @2.80 GHz

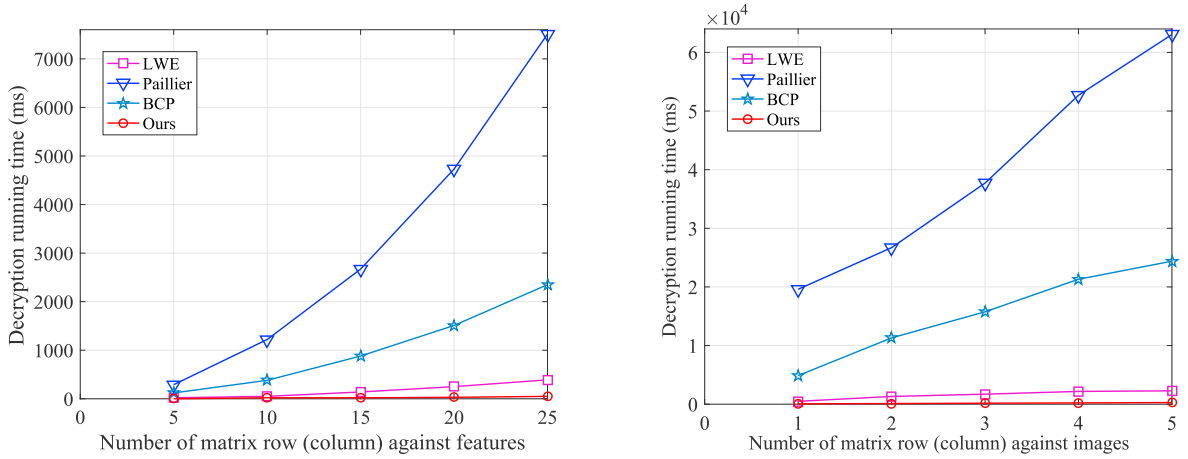



(a) Encryption time against features




(b) Encryption time against images

**Fig. 9.** Encryption time with the number of features and images.




(a) Decryption time against features (b) Decryption time against images

**Fig. 10.** Decryption time with the number of features and images.

(8CPUs), 8 GB RAM, and the 64-bit Windows operating system with Anaconda 3, PyCharm 2020.3.2 Professional Edition, Python 3.8.5, and PyTorch 1.7.0. The experimented dataset is MNIST dataset, which is composed of a training dataset with 50,000 handwritten digit images and validation and test datasets each with 10,000 images. The cryptosystem employs the LDEC with PSIMD, which is compared with the other cryptosystems, such as LWE [11], Paillier [51], Bresson-Catalano-Pointcheval (BCP) [52].

### 7.3.1. Accuracy

In this subsection, we tested the accuracy of PEPFL. Due to the complexity of ciphertext CNN, the CNN on the trainer conducts the training process in plaintext, which does not affect the accuracy. Therefore, the accuracy is tested without considering the privacy-preserving mechanism, and the experiment results are shown in Fig. 4 and Fig. 5.

Fig. 4 compares the classification accuracy according to the changes in the number of rounds and epochs. Fig. 4(a) plots the training accuracy of three kinds of different users {2, 3, 4} as the number of rounds changes. The figure indicates that the accuracy is slightly variable when the number of rounds is more than 40. The results show that the accuracy has less influence as the number of rounds and users increases. Fig. 4(b) draws the accuracy under different batches as the number of local epochs increases. When the number of the local epochs is less than 70, it can be seen that the smaller the batchs, the higher the accuracy. When the number of local epochs is greater than 70, the accuracy is nearly 100%. The results demonstrate that the higher the epochs, the better the accuracy. When the number of epochs reaches a certain amount, the accuracy is always 100%.

As the momentums change, Fig. 4(c) and (d) simulate the impact on accuracy with the variation of the number of epochs and rounds. Fig. 4(c) compares the accuracy of three different momentums {0.5, 0.7, 0.9} as the number of epochs grows. Obviously, as the number of local epochs increases, the accuracy improves constantly. When the number of local epochs approaches 80, the accuracy reaches 100%. However, when the momentum is greater than 1, the accuracy is below 10%, which is not plotted in Fig. 4(c). In Fig. 4(d), as the number of rounds increases, the accuracy remains almost constant. The results show that the accuracy remains almost unchanged with the increase in the number of rounds.

In Fig. 5, we compare the accuracy against the number of rounds and users with prior works [27,53]. As shown in Fig. 5(a), as the number of rounds {1, 3, 5, …, 15} increases, OFL [53] stays at 84.5% accuracy, PPFDL [27] grows constantly, and our accuracy maintains at 99% accuracy. The results demonstrate that the accuracy of the proposed scheme is superior to that of the two schemes with the number of rounds

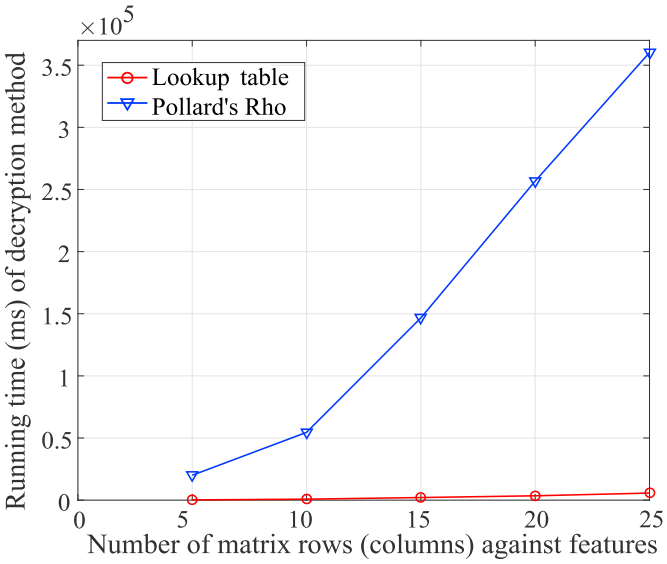



**Fig. 11.** The comparison of decryption time.

changing. In Fig. 5(b), as the number of users {100, 200, …, 500} grows, OFL and PPFDL increase continually, and our accuracy decreases with the number of users growing. In our scheme, owing to exploiting the method of split data in MNIST dataset, it influences the accuracy with the decrease of training data, but the accuracy is still higher than that of the comparison schemes in the comparison range.

### 7.3.2. Performance

In this subsection, we analyze communication costs, computation costs, and the efficiency of encryption and decryption time. Fig. 6 compares communication costs in different phases, where $u$ is the number of weight encoding packets. In Fig. 6(a), we set $p = 2048$ and $u = 1000$. The figure manifests that the communication costs of the encryption and prediction phases constantly grow as the number of matrices increases, while the other phases remain almost unchanged. The results show that the number of matrices influences the encryption and prediction phases, but does not affect the other phases. In Fig. 6(b), we set $s$ to 1000. As the number of the weight encoding packets $u$ increases, the figure indicates that the communication costs of the local update and aggregation phases continually increase, and the other phases are slightly influenced.

In Fig. 7, we compare the communication costs of our scheme with

those of two schemes in Ref. [27]. The schemes of [27] adopt the encryption of gradients, and our scheme utilizes the encryption of weight packets. As shown in Fig. 7(a), as the number of gradients {1000, 2000, …, 5000} increases, PPFL and $PPFDL_{imd}$ grow continually in user side. Our costs in user side are small and almost constant. Therefore, our cost is significantly lower than that of the comparison schemes. From Fig. 7(b), PPFL and $PPFDL_{imd}$ grow constantly, and our costs also increase, but it is obvious that our communication cost in server side is lower than the comparison schemes.

In Fig. 8, since the computation costs of the initialization and aggregation phases are close to zero and the local update phase contains the gradients, the computation costs of these phases cannot be compared. Therefore, we only compare the computation costs in the encoding and encryption phase (in user side) and the prediction phase. As shown in Fig. 8(a), as the number of matrices grows, the decryption time in the prediction phase increases distinctly. The results show that the impact on decryption time is obvious in the prediction phase, but the impact on encryption time is small. In Fig. 8(b), we compare the computation costs of our scheme with those of the two schemes (PPFL and $PPFDL_{imd}$) of [27] in user side. As shown in Fig. 8(b), as the number of gradients {1000, 2000, …, 5000} in user side increases, computation costs of PPFL and $PPFDL_{imd}$ augment apparently, and our computation costs also increase, but not more than 150. The results indicate that the proposed scheme has lower computation costs than the comparison schemes.

Then, we experimented with the running time of encryption and decryption for different cryptosystems (LWE, Paillier, BCP, and our cryptosystem). In the process of encryption and decryption, the experimented data are {5*5, 10*10, 15*15, 20*20, 25*25} features and {1, 2, …, 5} images.

In Fig. 9, as the number of matrix rows (columns) versus features or images increases, the running time of encryption constantly increases. Fig. 9(a) and (b) show that the encryption time of Paillier is the largest, and that of our scheme is the smallest. The results indicate that our scheme has shorter encryption running time than the comparison schemes.

Fig. 10 illustrates the decryption running time as the number of matrix rows (columns) versus an increasing number of features or images. From Fig. 10(a) and (b), it is apparent that Paillier has the longest decryption time and our scheme has the shortest decryption time. The results show that our scheme has shorter decryption time and higher decryption efficiency than the comparison schemes.

Finally, we tested the ElGamal decryption method with Pollard's Rho and lookup table methods. As shown in Fig. 11, as the number of matrix rows (columns) increases, the running time of decryption constantly increases with Pollard's Rho and lookup table methods, but that of Pollard's Rho method increases distinctly. The results show that the lookup table method is superior to the Pollard's Rho method. Therefore, we select the lookup table method to decrypt our scheme.

## 8. Conclusion

Federated learning is a fashionable collaborative learning method that can collaboratively update the weights or gradients to obtain the global model. Because the weights or gradients can leak private information, it has been researched under different privacy-preserving methods. However, noninteractive and high-efficiency schemes in partially HEs have not been proposed. Therefore, we propose a PEPFL, which improves the LDEC scheme and presents a PSIMD parallel computing method, solving the distributed multi-key question for federated learning and improving efficiency. In addition, users do not participate in the training by introducing trainers, which realizes users' noninteraction. The scheme can widely be applied in the scenarios of non-interaction, multi-key, or partially HEs. To the best of our knowledge, this is the first work on a partially HE scheme that supports parallel computing with PSIMD, which can also be applied in other partially HEs. In future work, we will focus on preventing participants' from cheating and providing incentive mechanisms in federated learning.

## Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

This work is supported by the National Natural Science Foundation of China under Grant No. U19B2021, the Key Research and Development Program of Shaanxi under Grant No. 2020ZDLGY08-04, the Key Technologies R & D Program of He'nan Province under Grant No. 212102210084, and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province.

## References

[1] S. Sharma, K. Chen, Image disguising for privacy-preserving deep learning, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS 2018), ACM, 2018, pp. 2291–2293.
[2] Y. Chen, Y. Ping, Z. Zhang, B. Wang, S. He, Privacy-preserving image multi-classification deep learning model in robot system of industrial iot, Neural Comput, Appl 33 (10) (2021) 4677–4694.
[3] G. Kaissis, M.R. Makowski, D. Rueckert, R. Braren, Secure, privacy-preserving and federated machine learning in medical imaging, Nat. Mach. Intell. 2 (6) (2020) 305–311.
[4] Y. Wang, H. Tan, Y. Wu, J. Peng, Hybrid electric vehicle energy management with computer vision and deep reinforcement learning, IEEE Trans. Ind. Inf. 17 (6) (2021) 3857–3868.
[5] Y. Qu, L. Gao, T.H. Luan, Y. Xiang, S. Yu, B. Li, G. Zheng, Decentralized privacy using blockchain-enabled federated learning in fog computing, IEEE Internet Things J. 7 (6) (2020) 5171–5183.
[6] S. Li, P. Zheng, L. Zheng, An ar-assisted deep learning-based approach for automatic inspection of aviation connectors, IEEE Trans. Ind. Inf. 17 (3) (2021) 1721–1731.
[7] W. Tang, B. Li, M. Barni, J. Li, J. Huang, An automatic cost learning framework for image steganography using deep reinforcement learning, IEEE Trans. Inf. Forensics Secur. 16 (2021) 952–967.
[8] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, D. Bacon, Federated Learning: Strategies for Improving Communication Efficiency, arXiv preprint arXiv: 1610.05492.
[9] B. Hitaj, G. Ateniese, F. Pérez-Cruz, Deep models under the GAN: information leakage from collaborative deep learning, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017), ACM, 2017, pp. 603–618.
[10] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2017), vol. 54, PMLR, 2017, pp. 1273–1282.
[11] L.T. Phong, Y. Aono, T. Hayashi, L. Wang, S. Moriai, Privacy-preserving deep learning via additively homomorphic encryption, IEEE Trans. Inf. Forensics Secur. 13 (5) (2018) 1333–1345.
[12] G. Xu, H. Li, Y. Zhang, S. Xu, J. Ning, R.H. Deng, Privacy-preserving federated deep learning with irregular users, IEEE Trans. Dependable Secure Comput. 19 (2) (2022) 1364–1381.
[13] G. Xu, H. Li, S. Liu, K. Yang, X. Lin, Verifynet: secure and verifiable federated learning, IEEE Trans. Inf. Forensics Secur. 15 (2020) 911–926.
[14] T. Li, J. Li, X. Chen, Z. Liu, W. Lou, Y.T. Hou, Npmml: a framework for non-interactive privacy-preserving multi-party machine learning, IEEE Trans. Dependable Secure Comput. (99) (2020) 1–14.
[15] P. Xie, B. Wu, G. Sun, BAYHENN: combining bayesian deep learning and homomorphic encryption for secure DNN inference, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, ijcai.org, 2019, pp. 4831–4837.
[16] J. Liu, M. Juuti, Y. Lu, N. Asokan, Oblivious neural network predictions via minionn transformations, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017), ACM, 2017, pp. 619–631.
[17] C. Juvekar, V. Vaikuntanathan, A. Chandrakasan, GAZELLE: a low latency framework for secure neural network inference, in: Proceedings of 27th USENIX Security Symposium (USENIX Security 2018), USENIX Association, 2018, pp. 1651–1669.
[18] E. Ghadimi, H.R. Feyzmahdavian, M. Johansson, Global convergence of the heavy-ball method for convex optimization, in: Proceedings of the 14th European Control Conference (ECC 2015), IEEE, 2015, pp. 310–315.
[19] J. Wang, V. Tantia, N. Ballas, M.G. Rabbat, Slowmo: improving communication-efficient distributed SGD with slow momentum, in: Proceedings of the 8th International Conference on Learning Representations (ICLR 2020), OpenReview.net, 2020, pp. 1–25.

[20] W. Liu, L. Chen, Y. Chen, W. Zhang, Accelerating federated learning via momentum gradient descent, IEEE Trans. Parallel Distr. Syst. 31 (8) (2020) 1754–1766.
[21] L. Zhao, Q. Wang, Q. Zou, Y. Zhang, Y. Chen, Privacy-preserving collaborative deep learning with unreliable participants, IEEE Trans. Inf. Forensics Secur. 15 (2020) 1486–1500.
[22] L. Cui, Y. Qu, G. Xie, D. Zeng, R. Li, S. Shen, S. Yu, Security and privacy-enhanced federated learning for anomaly detection in iot infrastructures, IEEE Trans. Ind. Inf. 18 (5) (2022) 3492–3500.
[23] Y. Chen, B. Wang, Z. Zhang, Pdlhr: privacy-preserving deep learning model with homomorphic re-encryption in robot system, IEEE Syst. J. (2021) 1–12.
[24] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, Y. Zhou, A hybrid approach to privacy-preserving federated learning, in: Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security (AISec@CCS 2019), ACM, 2019, pp. 1–11.
[25] N. Agrawal, A.S. Shamsabadi, M.J. Kusner, A. Gascón, QUOTIENT: two-party secure neural network training and prediction, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS 2019), 2019, pp. 1231–1247.
[26] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, H. Ludwig, Hybridalpha: an efficient approach for privacy-preserving federated learning, in: Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security (AISec@CCS 2019), ACM, 2019, pp. 13–23.
[27] G. Xu, H. Li, Y. Zhang, S. Xu, J. Ning, R.H. Deng, Privacy-preserving federated deep learning with irregular users, IEEE Trans. Dependable Secure Comput. 19 (2) (2022) 1364–1381.
[28] X. Liu, R.H. Deng, K.R. Choo, J. Weng, An efficient privacy-preserving outsourced calculation toolkit with multiple keys, IEEE Trans. Inf. Forensics Secur. 11 (11) (2016) 2401–2414.
[29] C. Li, W. Ma, Comments on "an efficient privacy-preserving outsourced calculation toolkit with multiple keys", IEEE Trans. Inf. Forensics Secur. 13 (10) (2018) 2668–2669.
[30] P. Li, J. Li, Z. Huang, T. Li, C. Gao, S. Yiu, K. Chen, Multi-key privacy-preserving deep learning in cloud computing, Future Generat. Comput. Syst. 74 (2017) 76–85.
[31] X. Ma, J. Ma, H. Li, Q. Jiang, S. Gao, PDLM: privacy-preserving deep learning model on cloud with multiple keys, IEEE Trans. Serv. Comput. 14 (4) (2021) 1251–1263.
[32] H. Chen, W. Dai, M. Kim, Y. Song, Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS 2019), ACM, 2019, pp. 395–412.
[33] S. Zhang, A. Choromanska, Y. LeCun, Deep learning with elastic averaging SGD, in: Proceedings of Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, 2015, pp. 685–693.
[34] A. Fu, X. Zhang, N. Xiong, Y. Gao, H. Wang, J. Zhang, Vfl, A verifiable federated learning with privacy-preserving for big data in industrial iot, IEEE Trans. Ind. Inf. 18 (5) (2022) 3316–3326.
[35] A.C. Yao, Protocols for secure computations (extended abstract), in: Proceedings of 23rd Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1982, pp. 160–164.
[36] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, in: Proceedings of CRYPTO '84, vol. 196, Springer, 1984, pp. 10–18.
[37] J. Pollard, Monte Carlo method for index computation (mod p), Math. Comput. 32 (143) (1978) 918–924.
[38] R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, Secure distributed key generation for discrete-log based cryptosystems, J. Cryptol. 20 (1) (2007) 51–83.
[39] X. Yi, F. Rao, E. Bertino, A. Bouguettaya, Privacy-preserving association rule mining in cloud computing, in: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '15), ACM, 2015, pp. 439–450.
[40] D. Chaum, C. Crépeau, I. Damgård, Multiparty unconditionally secure protocols (abstract), in: Proceedings of the Conference on the Theory and Applications of Cryptographic Techniques (CRYPTO '87), vol. 293, Springer, 1987, p. 462.
[41] X. Yang, Y. Feng, W. Fang, J. Shao, X. Tang, S.-T. Xia, R. Lu, An Accuracy-Lossless Perturbation Method for Defending Privacy Attacks in Federated Learning, 2021 09843 arXiv:2002.
[42] N. Qian, On the momentum term in gradient descent learning algorithms, Neural Network. 12 (1) (1999) 145–151.
[43] T. Ogilvie, R. Player, J. Rowell, Improved privacy-preserving training using fixed-hessian minimisation, IACR Cryptol. ePrint Arch. 2020 (2020) 1514.
[44] D. Chai, L. Wang, K. Chen, Q. Yang, Secure federated matrix factorization, IEEE Intell. Syst. 36 (5) (2021) 11–20.
[45] D. Chaum, C. Crépeau, I. Damgård, Multiparty unconditionally secure protocols (extended abstract), in: Proceedings of the 20th Annual ACM Symposium on Theory of Computing, ACM, 1988, pp. 11–19.
[46] E. Hesamifard, H. Takabi, M. Ghasemi, Cryptodl: Deep Neural Networks over Encrypted Data, CoRR abs/1711.05189.
[47] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, R. Sharma, Cryptflow2: practical 2-party secure inference, in: Proceedings of 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), ACM, 2020, pp. 325–342.
[48] R. Canetti, M. Varia, Decisional diffie-hellman problem, in: Encyclopedia of Cryptography and Security, second ed., Springer, 2011, pp. 316–319.
[49] Y. Desmedt, Y. Frankel, Threshold cryptosystems, in: Proceedings of the 9th Annual International Cryptology Conference (CRYPTO '89), vol. 435, Springer, 1989, pp. 307–315.
[50] R. Canetti, Universally composable security: a new paradigm for cryptographic protocols, in: Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001), IEEE Computer Society, 2001, pp. 136–145.
[51] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: Proceedings of International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '99), vol. 1592, Springer, 1999, pp. 223–238.
[52] E. Bresson, D. Catalano, D. Pointcheval, A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications, in: Proceedings of the 9th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2003), vol. 2894, Springer, 2003, pp. 37–54.
[53] V. Smith, C. Chiang, M. Sanjabi, A.S. Talwalkar, Federated multi-task learning, in: Advances in Neural Information Processing Systems, vol. 30, 2017, pp. 4424–4434. NeurIPS 2017.