Privacy Preserving Federated Learning from Multi-input Functional Proxy Re-encryption

Xinyu Feng, Qingni Shen✉, Cong Li, Yuejian Fang✉, Zhonghai Wu

School of Software and Microelectronics & School of Computer Science, Peking University

Introduction

In this paper, we proposed a novel multi-input functional proxy re-encryption (MI-FPRE) scheme which allows the semi-trusted server to evaluate the output of a function without knowing the original input and the result of the function. Based on MI-FPRE, we further presented a new FL framework. Compared with prior frameworks, in our FL framework, the central server cannot get any information during the parameter aggregation process. To the best of our knowledge, this is the first FE-based FL framework that can achieve such property, and solved the attack on FL through multiple rounds of ciphertext and key mixing. Eventually, we conducted a series of comparative experiments to prove that our framework is efficient and accurate while providing a better protection of privacy.

Our motivation is to propose a new framework to address the above security issues in FL. The main goals of our framework are: (1) Preventing the central server from accessing parameter information by combining ciphertexts and keys from different iteration rounds. (2) Preventing the central server from recovering original information through multiple ciphertexts corresponding to a same session key. (3) Ensuring that the central server cannot obtain any information of intermediate or global parameters. Our contributions can be summarized below:

1. We proposed a novel MI-FPRE scheme which enables a semi-trusted proxy to evaluate the output of a function without revealing any information. To the best of our knowledge, this is the first FE scheme to achieve such properties. Our MI-FPRE scheme enhances the aggregation process of FL to achieve better security and privacy.
2. We formalize a new FL framework based on MI-FPRE. In our framework, FL clients encrypt the intermediate parameters and upload the ciphertexts to the server. The server aggregates the parameters by running the re-encryption algorithm. In this aggregation process, the server cannot disclose any information about either the intermediate parameters or the global parameters.
3. We conduct a series of experiments on image classification tasks. The results demonstrate that while achieving better privacy, our framework achieves less communication overhead and higher computational efficiency without losing accuracy. Besides, the efficiency of parameters aggregation in our FL framework grows linearly with the increasing of parameters size, making it easy to predict the time cost of federated aggregation with known model parameter sizes, making it very practical.

System Model

There are 3 types of parties in our FL (Fig. 1).

- Key Generation Center ($\mathcal{K}$): It maintains the MI-FPRE system and generates the public parameters, encryption keys and decryptions for other participants. $\mathcal{K}$ is a trusted third party and will not collude with any other participant.
- Central Server ($\mathcal{S}$): The central server aggregate the intermediate parameters from the clients and delivers the global parameters back in each round of iteration.
- Client ($\mathcal{C}$):There are multiple clients in the framework, each client trains the local model and uploads the parameter vectors to the central server for aggregation.

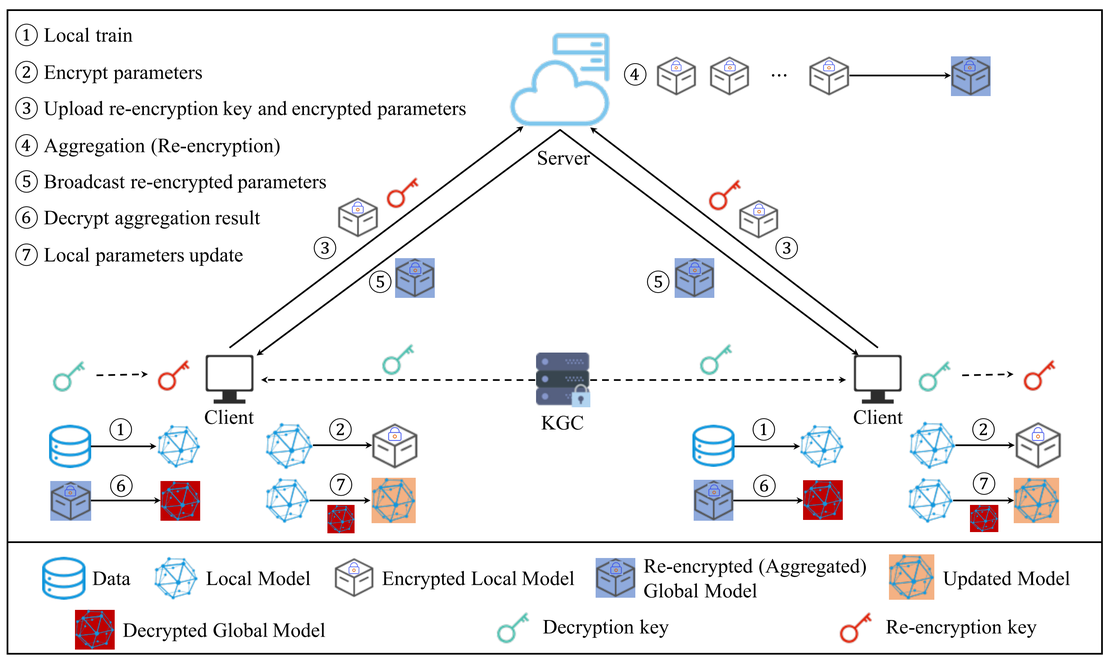


Figure 1. System model of MI-FPRE

Construction of MI-FPRE

- Setup(λ). The key generation center (KGC) takes the security parameter λ as input and calls the group generation algorithm $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$. Then it selects a hash function $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{G}^2$ that maps any string to a group element of $\mathbb{G}^2$. For $i \in [n]$, the KGC selects $\boldsymbol{s}_i \in \mathbb{Z}_p^2$, and set $\boldsymbol{ek}_i = \boldsymbol{s}_i$. The public parameters and master key are formed as $pp = (\mathbb{G}, p, g, \mathcal{H}), msk = ((\boldsymbol{ek}_i)_i)$.
- DKGen($msk, \boldsymbol{y}$). The KGC takes as input the $msk = (\boldsymbol{s}_i)_i$ and a vector $\boldsymbol{y}$, it returns the decryption key $dk_{\boldsymbol{y}}$ to each client. For $y_i \in \boldsymbol{y}$, the KGC calculates $\boldsymbol{d} = \sum_{i=1}^{m} \boldsymbol{s}_i \cdot y_i$ and set the decryption key $dk_{\boldsymbol{y}} = (\boldsymbol{y}, \boldsymbol{d})$.
- Encrypt($\boldsymbol{ek}_i, x_i, \ell$). With an encryption key $\boldsymbol{ek}_i = \boldsymbol{s}_i$, a client computes $[\boldsymbol{\mu}_\ell] = \mathcal{H}(\ell) \in \mathbb{G}^2$. It calculates $[c_i] = [\boldsymbol{\mu}_\ell^\top \boldsymbol{s}_i + x_i]$. The ciphertext is formed as $ct_i = [c_i]_{i \in [n_j]}$ (n_j is the vector size of client j).
- RKGen($dk_{\boldsymbol{y}}$). One of the clients selects $s' \in \mathbb{Z}_p^*$. For $i \in [m]$ and client $j \in [n]$, it calculates $rk = \sum_{i=1}^{m} \boldsymbol{\mu}_\ell^\top \boldsymbol{s}_j \cdot (y_i - s')$ and sends rk to the server $\mathcal{S}$.
- ReEnc(ct_i, rk). For $i \in [m]$ where m is the vector length. The server re-encrypts the original ciphertext ct_i with the re-encryption key rk from the clients. It calculates $ct' = \sum_{i=1}^{m}(c_i \cdot y_i) - rk$. Then it returns the re-encrypted ciphertext ct' to each of the clients.
- Decrypt(ct, dk). For the original ciphertext $ct = ct_i$, the user with $dk = dk_{\boldsymbol{y}}$ calculates $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^{m} c_i \cdot y_i - [\boldsymbol{\mu}_\ell^\top] \cdot \boldsymbol{d} = \sum_{i=1}^{m} x_i \cdot y_i$. For the re-encrypted ciphertext $ct = ct'$, the user with $dk = s'$ calculates $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = ct' - \sum_{i=1}^{m} \boldsymbol{\mu}_\ell^\top \boldsymbol{s}_i \cdot s' = \sum_i^m x_i \cdot y_i$ and returns $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$.

FL Framework

As is demonstrated in Framework 1, there are two phases in our FL framework.

- Initialization: This phase can be executed in advance, where the system parameters are generated by the KGC and necessary information will be synchronized.
- Federated learning: There are three stages in this phase, including LocalTrain, FedAverage and LocalUpdate. The clients train the local models in LocalTrain and the server aggregates the global parameters in FedAverage, finally the clients update the local parameters in LocalUpdate.

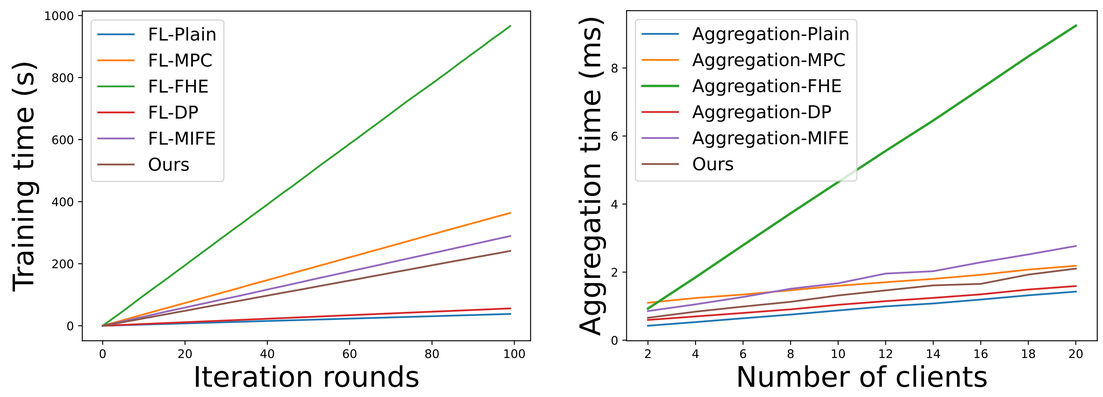
Framework 1 The proposed FL framework

Input: The security parameter λ, the clients $\mathcal{C}_i$ $(i \in [n])$, the central server $\mathcal{S}$, the key generation center $\mathcal{K}$.
Output: The global model $\boldsymbol{\theta}_g$.

procedure INITIALIZATION
  $\mathcal{K}$ runs $pp, msk \leftarrow \mathsf{Setup}(1^\lambda)$ and broadcasts pp.
  $\mathcal{K}$ sets $\boldsymbol{y} = (1, \cdots, 1)$, where $|\boldsymbol{y}| = n$.
  $\mathcal{K}$ runs $dk_{\boldsymbol{y}} \leftarrow \mathsf{DKGen}(msk, \boldsymbol{y})$.
  $\mathcal{K}$ sends $dk_{\boldsymbol{y}}$ to $\mathcal{C}_i$ $(i \in [n])$.
  $\mathcal{C}_i$ calls $rk \leftarrow \mathsf{RKGen}(dk_{\boldsymbol{y}})$
  $\mathcal{C}_i$ sends rk to $\mathcal{S}$.
procedure FEDERATED LEARNING
Stage 1: LocalTrain
For each client $\mathcal{C}_i$ $(i \in [n])$:
  1. Train locally and get the parameter $\boldsymbol{\theta}_i$.
  2. Run $ct_i \leftarrow \mathsf{Encrypt}(\boldsymbol{ek}_i, \boldsymbol{\theta}_i, \ell)$, ℓ is round index.
  3. Send ct_i to $\mathcal{S}$.
Stage 2: FedAverage
  $\mathcal{S}$ collects the encrypted parameters ct_i and rk.
  $\mathcal{S}$ calls $ct' \leftarrow \mathsf{ReEnc}(ct_i, rk)$ for aggregation.
  $\mathcal{S}$ sends ct' to $\mathcal{C}_i$ $(i \in [n])$.
Stage 3: LocalUpdate
For each client $\mathcal{C}_i$ $(i \in [n])$:
  $\mathcal{C}_i$ runs $\boldsymbol{\theta}_g \leftarrow \mathsf{Decrypt}(ct', s')$.
  $\mathcal{C}_i$ updates the local parameter with $\boldsymbol{\theta}_g$.

Experiments

The experiments of this work are running with image classification tasks using a convolutional neural network (CNN) on the MNIST, Fashion-MNIST and CIFAR-10 datasets. We compare our framework with the FL frameworks based on MPC (CrypTFlow2 [3]), HE (CKKS [2]), DP [4] and MIFE [1]. We refer to our paper for the results of testing accuracy and communication overhead, the running times of MI-FPRE and the FL protocol are shown in Fig. 2 and Fig. 3 respectively.



(a) Training time (b) Aggregating time

Figure 2. Comparison on running time of FL frameworks.

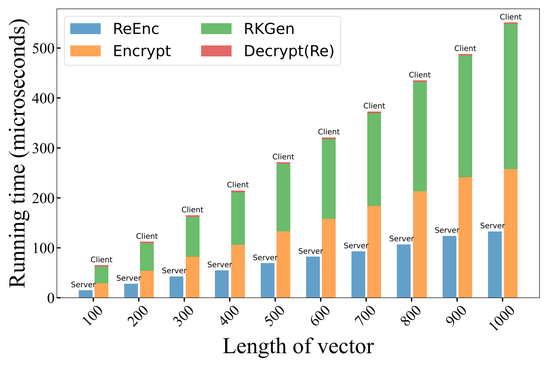


Figure 3. Running time of algorithms in MI-FPRE

References

[1] Yansong Chang, Kai Zhang, Junqing Gong, and Haifeng Qian.
Privacy-preserving federated learning via functional encryption, revisited.
IEEE TIFS, 18:1855–1869, 2023.

[2] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song.
Homomorphic encryption for arithmetic of approximate numbers.
In *ASIACRYPT 2017, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

[3] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma.
Cryptflow2: Practical 2-party secure inference.
In *CCS*, pages 325–342, 2020.

[4] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor.
Federated learning with differential privacy: Algorithms and performance analysis.
IEEE TIFS, 15:3454–3469, 2020.

[xyf, li.cong, wuzh]@pku.edu.cn, [qingnishen, fangyj]@ss.pku.edu.cn