PPSFL: Privacy-preserving Split Federated Learning via Functional Encryption

Jing Ma ¹, Lv Xixiang ², Yong Yu ², and Stephan Sigg ²

¹Xidian University
²Affiliation not available

November 1, 2023

Abstract

In this paper, we propose a novel and efficient privacy-preserving split federated learning (PPSFL) framework, that achieves both privacy protection and model accuracy with reasonable computational and communication cost. We describe the implementations of PPSFL on Multi-layer Perceptron (MLP) and Convolutional Neural Network (CNN) models with distributed clients to evaluate the performance of PPSFL.
PPSFL: Privacy-preserving Split Federated Learning via Functional Encryption

Jing Ma, Xixiang Lyu, Yong Yu, and Stephan Sigg

Abstract—Federated learning (FL) and split learning (SL) are two emerging distributed collaborative learning mechanisms that enable model optimization without sharing the local data. The integration of FL and SL, called split federated learning (SFL), reduces the training burden on the client side and parallelizes the model training to achieve efficient model optimization. Currently, concerns about the privacy leakage risks of split federated learning have garnered much attention, but effective solutions have not yet been fully explored. To this end, we propose a novel privacy-preserving split federated learning framework, named PPSFL, that guarantees the confidentiality of the intermediate parameters to protect data privacy. Specifically, in PPSFL, we design a secure model training based on single-input functional encryption and a secure model aggregation based on multi-input functional encryption to achieve secure model optimization. In addition, we describe the implementations of PPSFL and compare the performance with centralized learning (CL), FL, SL, SFL, state-of-the-art privacy-preserving SFL, and state-of-the-art privacy-preserving model training schemes. The experimental results show that PPSFL significantly reduces the computation and communication costs without compromising model accuracy.

Index Terms—federated learning, split learning, functional encryption, privacy protection.

I. INTRODUCTION

CONTEMPORARY learning applications may comprise the transmission of a large amount of data from distributed clients to open platforms for data analysis and model training. Such aggregation of data from devices with a central party may raise concerns regarding privacy and strategy.

To alleviate these concerns, distributed collaborative learning methods such as federated learning (FL) [1]–[3] and split learning (SL) [4], [5] have been proposed. They enable multiple parties to collaborate in the training of a deep learning model while maintaining the distribution of data. In FL, a global model is shared with all clients for local training, and only the model updates (e.g. model weights) are shared with a central party (server) for aggregation. However, FL places the burden of model training completely onto the distributed clients, which may be infeasible in the presence of resource-constrained clients. SL partially transfers computation to the central server by splitting the whole model into multiple network portions, which are then trained separately on distributed clients and one or more central server entities. By sharing the cut-layer representations (smashed data) [6], clients and server conduct forward and backward propagation on the global model. SL is, therefore, more suitable for resource-constrained devices than FL. Furthermore, since most of the network is located by the central entity, SL protects the model property. However, as a result of the lack of parallelization during training (caused by the physical distribution of the model), SL adapts a model slower than FL.

To simultaneously exploit the respective advantages and to make up for their deficiencies, [7] proposed a splitfed learning (SFL) framework to achieve efficient privacy-preserving collaborative learning applicable to resource-constrained distributed devices. Specifically, the global model is split into server and client models. The client-models are trained in parallel, exploiting local data and interacting with the central entity for global model training. Finally, federated model aggregation is applied by clients and server jointly.

Studies have shown that FL and SL suffer from inference or adversarial attacks and that they have insufficient privacy protection [8]–[12]. Solutions include secure multi-party computation, differential privacy, and homomorphic encryption to protect the intermediate parameters. Security and privacy concerns on SFL have also been noticed [13], [14]. Solutions include differential privacy [7], which leads to reduced accuracy of the trained model.

We propose a novel and efficient privacy-preserving split federated learning (PPSFL) framework that achieves both privacy protection and model accuracy with reasonable computational and communication costs while protecting the model property to a certain extent. PPSFL splits the global model into a client and server-model and applies functional encryption to protect data privacy. Client and server exchange intermediate encrypted parameters to achieve collaborative learning at reduced computation and communication load and improved model privacy. Specifically, we propose secure forward and backward propagation based on single-input functional encryption (SIFE) for the inner product. In the model splitting training phase, the clients encrypt intermediate parameters and send them to the server. The server securely calculates the function of the model training by functional decryption to complete model training under the premise of protecting data privacy. To securely aggregate the client-model, we design a secure federated averaging based on multi-input functional encryption (MIFE) for the inner product. In the federated aggregation phase of PPSFL, the local models of multiple clients, which may be infeasible in the presence of resource-constrained clients.
clients are sent to the server through functional encryption. The central entity (server) then achieves secure aggregation of models through function decryption and protects the privacy and security of model parameters. We describe the implementations of PPSFL on Multi-layer Perceptron (MLP) and Convolutional Neural Network (CNN) models with distributed clients to evaluate the performance of PPSFL. We tested the model accuracy and the computational and communication cost. The results show that PPSFL is fast converging and preserves model accuracy. Compared with existing privacy-preserving deep learning approaches, the communication and computation cost are significantly reduced. It also has a higher model accuracy than existing privacy-preserving split federation learning frameworks.

Our contributions are:

1. We propose PPSFL, an efficient privacy-preserving split federated learning framework that applies functional encryption to protect the data and model parameters.
2. We design a secure forward and backward propagation based on single-input functional encryption (SIFE) for the inner product. This algorithm achieves secure model training while protecting the intermediate data privacy.
3. We design a secure federated averaging based on multi-input functional encryption (MIFE) for the inner product. Functional encryption happens at the distributed clients and decryption at the central server to achieve secure model aggregation. This guarantees the confidentiality of model parameters.
4. We conduct extensive experiments on MLP and CNN models to evaluate the performance of PPSFL and compare it with a state-of-the-art privacy-preserving SFL scheme. The results show a significant reduction in the communication and computation load as well as reasonable energy consumption while preserving the model accuracy.

II. RELATED WORK

Federated learning was first introduced by Konečný et al. [1] in 2016. The concept addresses both data isolation and data privacy, training a global model on individual devices with local data and sharing the gradient updates of the model only. This design makes it more difficult for an adversary to obtain privacy-sensitive information from data, such as medical records, behavior patterns, or identity [15]–[17]. McMahan et al. [18] proposed Federated Averaging (FedAvg) to reduce the communication cost in federated learning by conducting multiple model training epochs before model aggregation.

However, concerns about privacy leakage of federated learning have been raised. For example, Nasr et al. [19] analyzed the membership inference attack and designed a gradient ascending attack. Melis et al. [9] used the intermediate model in the federated learning training stage to generate model updates with target attributes and model updates without target attributes, effectively obtaining sensitive information. Hitaj et al. [10] and Wang et al. [20] used adversarial generative networks to infer certain pieces of other clients’ local data. Moreover, Zhu et al. [8] indicated that the model gradient information reveals characteristics of the training data.

Therefore, alternative mechanisms to further improve privacy in federated learning have been proposed, such as homomorphic encryption [21], differential privacy [22]–[24], as well as multi-party computation [25]. Homomorphic-encryption approaches [26]–[28] apply homomorphic encryption on model gradients to achieve secure model aggregation while protecting data privacy. Differential privacy approaches [29], [30] add differential noise to gradients to protect data privacy. Multi-party computation approaches [31], [32] design secure protocols for multiple parties to aggregate model updates. However, these approaches cause high computation and communication costs and may lead to accuracy reduction.

In 2018, a split neural network was first proposed in [4], [5]. Specifically, the model is split and physically distributed into multiple network portions by layer. Training is conducted on a central entity as well as on distributed clients separately. The clients interact with the central entity sequentially to complete model training. SL realizes multi-data resource collaborative model training on the premise that the client and server do not share the original dataset. After the forward propagation ends at the central entity, the model output is passed to the clients to calculate the loss function and return the loss for backward propagation. With the increasing number of clients, communication efficiency in SL exceeds that of FL. It is further scalable with model size. FL, on the other hand, has better efficiency with increased data samples and small model size and client count [33]. The training order across clients may impact the model performance [34] and the model performs worse for data from those clients that performed the training first.

Although SL reduces the computation and communication cost and protects data and model privacy by offloading parts of the model to the server, SL still has privacy and security risks. Abuadbba et al. [35] pointed out that SL provides insufficient privacy protection for the 1D CNN model. Erdogan et al. [36] showed for plaintext SL that an honest-but-curious SplitNN server can infer the client’s input and also the labels. In addition, Pasquini et al. [11] proposed Feature-space hijacking (FSHA) against split learning to reconstruct or infer properties of training data. The authors proposed SplitGuard to detect training-hijacking attacks.

Other proposals for privacy-preserving split learning comprise homomorphic encryption and differential privacy. The authors of [35] increased the number of hidden layers of the client models and added differential privacy noise to the activation value of the cut layer to eliminate the risk of privacy leakage for the 1D CNN model. Khan et al. [38] designed a simplified version of the 1D CNN and applied homomorphic encryption to protect the activation map of the cut layer. The
server then conducts forward and backward propagation on the encrypted activations, inevitably resulting in significant computational overhead.

To eliminate the inherent drawbacks (better model privacy than FL and smaller training delay than SL), Thapa et al. [7] proposed splitfed learning (SFL). The approach also uses differential privacy to protect intermediate parameters, but the added noise also reduces the accuracy of the model. To minimize the latency and communication cost of SFL, Han et al. [39] proposed federated split learning, in which the client and server-models are updated in parallel via an SL-specific local-loss-based training. Moreover, Abedi et al. [40] proposed FedSL, a federated split Learning framework for Recurrent Neural Networks models. Specifically, local models are shared with the central entity for aggregation.

To summarize, no framework has yet been proposed that equally takes into account both model performance and privacy protection for split federated learning.

III. TECHNICAL BACKGROUND

In this section, we first present the construction of single-input and multi-input functional encryption for the inner product. We then introduce the technical details of FE and SL. We denote vectors in bold and use $(\mathbf{u}, \mathbf{v})$ to denote the inner product of two vectors $\mathbf{u}$ and $\mathbf{v}$. $\lambda$ denotes the security parameter: all known effective attacks against the cryptographic scheme require $\Omega(2^{\lambda})$ bit operations.

A. Functional Encryption

Functional encryption (FE) [41] is a public-key paradigm in which the decryption key enables a user to learn a specific function $f(x)$ over the encrypted data $Ct(x)$ but not the data $x$ itself. FE requires a trusted authority (TA) to generate a master secret key so as to generate the decryption key $sk_f$ associated with a specific function $f(x)$. Anyone holding $sk_f$ is able to compute $f(x)$ over the ciphertexts $Ct(x)$ encrypted by the data owner.

Single-input functional encryption (SIFE) considers functionalities where all inputs are provided and encrypted by the data owner. FE requires a trusted authority (TA) to generate a master secret key so as to generate the decryption key $sk_f$ associated with a specific function $f(x)$. Anyone holding $sk_f$ is able to compute $f(x)$ over the ciphertexts $Ct(x)$ encrypted by the data owner.

Single-input functional encryption (SIFE) considers functionalities where all inputs are provided and encrypted by a single party. Given the ciphertext $Ct(x)$ for plaintext data $x = (x_1, \ldots, x_n)$, the decryption key $sk_f$ of function $f(x)$ can retrieve $f(x_1, \ldots, x_n)$ while no further information about $x$ is revealed. SIFE for inner products [42] is a special case when $f$ is the inner product function.

$$f_{y_1,\ldots,y_n}(x_1,\ldots,x_n) = \langle x, y \rangle = \sum_{i=1}^{n} (x_i \cdot y_i)$$

Multi-input functional encryption (MIFE) [43] is a variant of FE that allows the computation of $f(x_1, \ldots, x_n)$ from $n$ ciphertexts $Ct_1, \ldots, Ct_n$ corresponding to messages $x_1, \ldots, x_n$ of $n$ parties, respectively. When $f$ is the inner product function, the following functionality is achieved:

$$f_{y_1,\ldots,y_n}(x_1,\ldots,x_n) = \sum_{i=1}^{n} (x_i \cdot y_i)$$

Definition 1. The Decisional Diffie-Hellman (DDH) assumption. Let GroupGen be a probabilistic polynomial-time (PPT) algorithm that takes as input a security parameter $1^\lambda$ and outputs a triple $(G, p, g)$, where $G$ is a group of order $p$ with $g$ being the generator of $G$ and $p$ is an $\lambda$-bit prime number. Then, the tuples $(g, g^r, g^{r+s})$ and the DDH tuples $(g, g^r, g^{r+s})$ are computationally indistinguishable, where $(G, p, g) \leftarrow \text{GroupGen}(1^\lambda)$, and $x, y, z \in Z_q$ are chosen independently and uniformly at random.

(i) SIFE for inner product from DDH

- Setup$(1^\lambda, 1^\lambda)$: Sample $(G, p, g) \leftarrow \text{GroupGen}(1^\lambda)$ and $s = (s_1, \ldots, s_r) \leftarrow Z_p^*$. Set $mpk = (h_i = g^{s_i})_{i \in [r]}$, $msk = s$, return key pair $(mpk, msk)$.
- Encrypt$(mpk, x)$: for vector $x = (x_1, \ldots, x_r) \in Z_p^r$, generate random number $r_i \leftarrow Z_p$, compute $ct_i = g^{r_i} \cdot m_{i0}$, return ciphertexts $Ct = (ct_i, (ct_{i0})_{i \in [r]})$.
- KeyGen$(msk, y)$: for secret key $msk$ and function vector $y = (y_1, \ldots, y_r) \in Z_p^r$, compute decryption key $sk_y = (y, s)$.
- Decrypt$(mpk, Ct, sk_y)$: For public key $mpk$, ciphertexts $Ct = (ct_0, (ct_{ij})_{i \in [r]})$ and decryption key $sk_y$, compute:

$$C := \prod_{i=1}^{r} \frac{ct_i^{y_i}/ct_{i0}^{s_i}}{g^2}$$

return $\log(C) = \sum_{i=1}^{r} \langle x_i, y_i \rangle$.

We apply SIFE and MIFE for inner products based on the Decisional Diffie-Hellman (DDH) assumption to achieve secure model training and model aggregation.

B. Federated learning

As shown in Figure 1, multiple clients first download the global model from the server, train it with their local data, and then summarize the results into a model update (model weights or gradients) which is returned to the central server. At the
server, model updates from clients are aggregated into a new global model to be shared again. This process is conducted iteratively until model convergence. McMahan et al. [18] proposed FederatedAveraging (FedAvg, Algorithm 1) to exploit iterative model averaging and to reduce the communication load.

**Algorithm 1: FederatedAveraging (FedAvg)**

1. Let \( \eta \) be the learning rate, and \( B \) be the minibatch for local model training, \( S_i \) be the set of \( N \) clients.
2. **Server executes:**
   1. initialize \( w_0 \) // Global model parameters
   2. for each round \( t = 1, 2, ... \) do
      1. for each client \( d_i \) in \( S_i \) in parallel do
         1. \( w^t_{i+1} \leftarrow \text{ClientUpdate}(d_i, w^t_i) \);
      2. \( w^{t+1} \leftarrow \frac{1}{N} \sum_{i=1}^{N} w^t_{i+1} \);
   3. \( \text{ClientUpdate}(d_i, w) \): //Run on client \( d_i \)
   4. for local epochs \( l = 1, \ldots, L \) do
      1. for batch \( b \in B \) do
         1. \( w \leftarrow w - \eta \nabla \ell(w; b) \)
   5. return \( w \) to server

C. Split learning

We adopt the classic SL framework with label sharing [4]. We topologically define a model using a sequence of layers \( L_0, ..., L_K \). In SL, a model is split into two parts (cf. Figure 1): the client model \( W_C = L_0, ..., L_c \), in which \( L_c \) is called the cut layer, and the server model \( W_S = L_{c+1}, ..., L_K \), which are jointly trained by clients and server. The client forward-propagates \( W_C \) on the local dataset up to the cut layer and then passes the output at the cut layer (smashed data) and data label to the server. The server takes the smashed data as input to \( W_S \), continues forward propagation and calculates the loss function. Then, the server conducts backward propagation on \( W_S \) and passes the gradients to the client. After clients receive the gradient, the backward propagation of \( W_C \) is conducted by clients. SL involves multiple rounds of forward and backward propagation between clients and the server until the model converges.

IV. PROBLEM FORMULATION

In this section, we first formalize the system model, and then define the threat model and design goals.

A. System model

We distinguish three types of entities in PPSFL: A Trusted Authority (TA); \( N \) remote clients \((C = C_1, ..., C_N)\), as well as the central (cloud) server (CS). The entities are defined as follows:

1. **Trusted Authority (TA)** : TA is a trusted authority that provides key management services, initiates encryption systems and provides functional encryption and decryption keys to all parties.
2. **Remote clients \( C_i \)** : client has a local dataset, limited storage space and computing power, responsible for part of the model training task, and encrypts the local training parameters and submits them to the cloud server.
3. **Cloud server (CS)** : CS has unlimited storage space and computing power, responsible for most of the model training tasks, as well as the security aggregation and update of the global model.

As depicted in Figure 2, PPSFL including system setup (key distribution), model training and model aggregation phases. Overall, PPSFL splits a neural network between distributed users and the server to collaboratively train a global model. After several epochs of local training, the client models of all clients are sent to the server for aggregation. The server model is also aggregated on the server.

B. Threat model and design goals

We assume that the server and all participating clients are honest-but-curious, which means that they follow the scheme but may try to infer private information of other devices during...
Algorithm 2: Privacy-preserving split federated learning (PPSFL)

**Input**: Model of round $t$: $W^C_t, W^S_t$, set of $N$ clients: $S_t$, the minibatch $B$, the learning rate $\eta$, data label of client $C_i$; $Y_i$

**Output**: Model of round $t+1$: $W^C_{t+1}, W^S_{t+1}$

1 **TA executes:**
2 Sample $(G, p, q) \leftarrow \text{GroupGen}(1^L)$, then generate the SIFE key pair $(\text{mpk}_i, \text{msk}_i)$ and MIFE key pair $(\text{mpk}^m_i, \text{msk}^m_i)$ for each client $C_i$.

3 **Server executes:**
4 for Aggregation round $t = 1, 2, \ldots$ do
5     for $C_i \in S_t$, in parallel do
6         for local epochs $l = 1, 2, \ldots, L$ do
7             for batches $b \in B$ do
8                 receive from clients the ciphertexts $Ct(A_{i,t}), C(A'_{i,t}), Y_i$
9                 Secure forward propagation $\ell(A_{i,t}, C(A'_{i,t}), Y_i) \leftarrow \text{Serverfp}(Ct(A_{i,t}, W^S_{i,t}))$
10                Secure backward propagation $\nabla \ell(A_{i,t}; W^S_{i,t}) \leftarrow \text{Serverbp}(Ct(A'_{i,t}), \ell)$,
11                Send $\nabla \ell(A_{i,t})$ to client $C_i$
12                Server model update $W^S_{i,t} \leftarrow W^S_{i,t} - \eta \nabla \ell(W^S_{i,t}; A_{i,t})$
13             receive from clients the encrypted client model, securely aggregate them $W^C_{t+1} \leftarrow \text{FedSecure}((Ct(W^C_{i,t})_{i \in N})$
14         for local epochs $l = 1, 2, \ldots, L$ do
15             for batches $b \in B$ do
16                 Forward propagation $A_{i,t} \leftarrow \text{Clientfp}(W^C_{i,t})$
17                 Encrypt the activation matrix and its transpose $(Ct(A_{i,t}), C(A'_{i,t})) \leftarrow \text{Enc}(\text{mpk}_i, A_{i,t}, A'_{i,t})$
18                 Send $Ct(A_{i,t}), C(A'_{i,t}), Y_i$ to the server
19                 Receive from server $\nabla \ell(A_{i,t})$, execute backward propagation $\nabla \ell(W^C_{i,t}) \leftarrow \text{Clientbp}(\nabla \ell(A_{i,t}))$
20                 Client model update $W^C_{i,t} \leftarrow W^C_{i,t} - \eta \nabla \ell(W^C_{i,t}; b)$
21                 Client model encryption $Ct(W^C_{i,t}) \leftarrow \text{Enc}(\text{mpk}^m_i, W^C_{i,t})$

the process. We further assume that the clients may collude with the server to infer private information of other clients.

The PPSFL scheme aims to implement a privacy-preserving federated learning mechanism with the following design goals:

1. **Privacy protection**: Ensure the confidentiality of local model parameters and intermediate calculation results, and protect data and model privacy.

2. **High accuracy and efficiency**: Collaboratively train a global model with high accuracy with reasonable computational and communication costs.

V. THE PROPOSED PPSFL FRAMEWORK

In this section, we introduce the specific construction of our PPSFL framework for privacy-preserving split federated learning (cf. Figure 2).

A. PPSFL Workflow

PPSFL is a privacy-preserving split federated learning framework in which multiple distributed clients train a global model without sharing of their local (training) data. The framework physically splits/distributes a neural network across distributed clients and a central server to collaboratively train a global model. Model parameters are aggregated at the server after parallel training at the distributed clients (cf. Algorithm 2).
The server sends a forward propagation functional key request \( L \). Then, SIFE is applied to encrypt the activation matrix and its inner product function \( X \). During backward propagation, the model computes the gradients of loss and computes the loss. MIFE generates public and secret keys, \((\text{mpk}^i, \text{msk}^i)\) for each client \( C_i \), and sends public keys to corresponding clients.

2) **SIFE-based secure model training**: During model training, a secure forward and backward propagation based on single-input functional encryption (SIFE) for inner product is designed including the following steps (cf. algorithm 3).

**Step1**: Client-side forward propagation and encryption. A client \( C_i \) randomly selects the data sample set with batch size \( b \) from the local dataset \( D_{\text{local}} \), performs forward propagation according to the input and model weight parameters, obtains the activations of the cut layer, and computes the matrix \( A_{i,t} \). Then, SIFE is applied to encrypt the activation matrix and its transpose: \((\text{Ct}(A_{i,t}), \text{Ct}(A'_{i,t})) \leftarrow \text{Enc}(\text{mpk}, A_{i,t}, A'_{i,t})\).

**Step2**: Server-side decryption and forward propagation. The server sends a forward propagation functional key request to the TA with each row of the weight matrix \( W_{c+1} \) of layer \( L_{c+1} \) as function coefficients. After receiving the functional key, it decrypts the ciphertext \( \text{Ct}(A'_{i,t}) \) by row and obtains the inner product function \( X_{c+1} = A_{i,t} \cdot W_{c+1} \), as the input of layer \( L_{c+1} \). After that, the server continues the forward propagation and computes the loss.

**Step3**: Server-side backward propagation. The server conducts backward propagation, computes the gradients of the server-model and the gradient of \( A_{i,t} \) (except for layer \( L_{c+1} \)’s parameter gradient, other layer parameter gradients can be computed normally). For layer \( l \) model parameters’ gradients, the calculation is according to \( dw = A_{i,t} \cdot dz \). As \( A'_{i,t} \) is encrypted by a functional public key, the server needs to send a backward propagation functional key request to the TA for each column of the matrix \( dA_{i,t} \). After receiving the functional key, the server conducts functional decryption to obtain the gradient \( dw \) of layer \( L_{c+1} \), computes the gradient \( dA'_{i,t} \) for \( A_{i,t} \) and sends it to the client to continue backward propagation.

**Step4**: Client-side backward propagation. After receiving gradient \( dA'_{i,t} \) of the activation value of the cut layer, the client performs backward propagation normally.

---

**Algorithm 3: Secure forward and backward propagation**

**Input**: Model of round \( t \) for client \( C_i \): \( W^C_{i,t}, W^S_{i,t} \), the weight and bias parameters of \( L_{w_{c+1}} \)’s first layer: \((L^c_{w_{c+1}}, L^c_{b_{i,t}})\), data samples and labels of batches \( b \)

**Output**: Updated model after one epoch of training: \( W^C_{i,t} \) and \( W^S_{i,t} \)

1. **TA executes:**
   - for batches \( b \) do
     1. Accepts from server the key requests of forward propagation \( f_{i}^{bp} \) for client \( C_i \), generates the decryption key \( sk_{i}^{bp} \leftarrow \text{KeyGen}(\text{msk}_i, L_{c+1}^c) \), return them to the server
     2. Accepts from server the key request of backward propagation \( b_{i}^{bp} \) for client \( C_i \), generates the decryption key \( sk_{i}^{bp} \leftarrow \text{KeyGen}(\text{msk}_i, dz_{c+1}^i) \), return them to the server

2. **Client executes:**
   - for batches \( b \) do
     1. Clientfp(\( W^C_{i,t} \), \( b \)):
       1. Operate forward propagation up to cut layer \( L_c \), get the output \( A_{i,t} \)
       2. Encrypt the output activation matrix (smoothed data) and its transpose:
          \((\text{Ct}(A_{i,t}), \text{Ct}(A'_{i,t})) \leftarrow \text{Encrypt}(\text{mpk}, A_{i,t}, A'_{i,t})\)
     3. Clientbp(\( dA_{i,t}, W^C_{i,t} \)):
       1. Accepts from server the gradients of smoothed data \( dA_{i,t} \), continues to operate backward propagation from \( L_c \) to \( L_0 \) to get the gradients \( \nabla L(W^C_{i,t}) \)
       2. Update client model: \( W^C_{i,t} \leftarrow W^C_{i,t} - \eta \nabla L(W^C_{i,t}) \)

3. **Server executes:**
   - for batches \( b \) do
     1. Serverfp(\( \text{Ct}(A_{i,t}), W^S_{i,t} \)):
       1. For forward propagation of client \( C_i \), for each row of matrix \( L_{c+1}^c \), send to TA key requests
       2. Accept the decryption key \( sk_{i}^{bp} \), decrypts the output of layer \( L_{c+1} \): \( z_{c+1} \) (without activating): \( z_{c+1} = \text{Decrypt}(\text{mpk}, \text{Ct}(A_{i,t})), sk_{i}^{bp} \)
       3. Continue to forward propagation, get the loss value \( L(W^S_{i,t}) \)
     2. Serverbp(\( \text{Ct}(A'_{i,t}), \text{loss} \)):
       1. Backward propagation, compute the gradients \( \nabla L(W^S_{i,t}) \) of layer \( L_K, \ldots, L_{c+2} \), the gradients \( dz^c_{c+1} \) of layer \( L_{c+1} \)’s output regarding to the loss
       2. For backward propagation of client \( C_i \), for each column of matrix \( dz^c_{c+1} \), send to TA the backward propagation key request
       3. Receive from server the decryption key \( sk_{i}^{bp} \), decrypts and get the gradients of parameters of layer \( L_{c+1} \):
          \( dw^c_{i} = \text{Decrypt}(\text{mpk}, \text{Ct}(A'_{i,t})), sk_{i}^{bp} \)
        4. Compute the gradients of bias of layer \( L_{c+1} \):
           \( db^c_{i} = dz_{c+1}^c \cdot 1_{b_{i+1}} \)
        5. Compute the gradients of activations of layer \( L_{c} \):
           \( dA_{i,t} = dz_{c+1}^c \cdot L_{w_{i+1}}^c \), send to clients
     3. Update server model: \( W^S_{i,t} \leftarrow W^S_{i,t} - \eta \nabla L(W^S_{i,t}) \)
3) MIFE-based secure model aggregation: We apply FedAvg with all devices converging to model training as a baseline. The complete model training consists of multiple aggregation rounds with $L$ local epochs each between the server and the clients. In each aggregation round, the server computes the weighted average of distributed model weights by computing the inner product function $f_y = \sum_{i=1}^{N} y_i x_i$, where $x_i$ are model weights and $y_i$ the weighted coefficients.

For PPSFL model aggregation, the server model is aggregated by the server and client-models need to be encrypted and sent to the server for secure aggregation. We use MIFE to protect client model parameters and to compute the weighted average for local model parameters of multiple clients, which can be regarded as a special case of MIFE (when encrypting, every $|x_i| = 1$, $|y_i| = 1$, rather than a vector of the MIFE).

Specifically, during aggregation round $t$, the server computes the weighted average of the client model by calculating the inner product function $f_y = \sum_{i=1}^{N} (y_i W^C_{i,t})$, where $W^C_{i,t}$ is the model parameter of client $C_i$, and $y_i$ are the coefficients for all client models. As shown in algorithm 4, each client encrypts its local model $W^C_{t}$ with a MIFE key and sends the model ciphertext to the server, which then requests the function key for the weighting coefficient $y = y_1, \cdots, y_N$ from the TA. The function key is used to decrypt the weighted function value of the client model and to use it in the next round of the client’s initial model $W^C_{t+1}$.

Algorithm 4: Client model secure aggregation

<table>
<thead>
<tr>
<th>Input</th>
<th>Client models of round $t$: $(W^C_{i,t})_{i\in N}$, aggregation coefficients $y = y_1, \cdots, y_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Client model of round $t + 1$: $W^C_{t+1}$</td>
</tr>
</tbody>
</table>

1. TA executes:
   1. Accepts from server the key request, check if the number of non-zero element is larger than 1, otherwise reject. Generate the aggregation decryption key: $sk_y \leftarrow \text{KeyGen}^m((\text{msk}_{i})_{i \in N}, y)$

2. Clients executes:
   1. Encrypts model $\text{Ct}(W^C_{i,t}) \leftarrow \text{Encrypt}^m(\text{mpk}^m, W^C_{i,t})$

3. Send the encrypted model to server.

4. Server executes:
   1. For aggregation function coefficients $y = y_1, \cdots, y_N$, send the key request to TA

5. After receive $sk_y$ and all encrypts model, decrypts to aggregation:

   $W^C_{t+1} \leftarrow \text{Decrypt}^m(\text{mpk}^m, (\text{Ct}(W^C_{i,t}))_{i \in N}, sk_y)$

VI. PRIVACY ANALYSIS

We implement the privacy of the model training stage by protecting the output value of the client model. Likewise, the privacy of the model aggregation stage is protected by protecting the parameter value of the client model. In split federated learning, the client only sends the cut-layer activation value and the client model parameters to the server, and in PPSFL, the single-input functional encryption algorithm is used to encrypt the cut-layer activation value, so that the server can only find the output value of the next layer according to the functional decryption key, and cannot know any information about the single activation value. When multiple client models are aggregated, PPSFL encrypts each client model parameters using multi-input functional encryption and sends them to the server for aggregation. The server, in turn, can only obtain the weighted average of all client models, but is not able to know any information about a single client model. The above two points are supported by single and multi-input functional encryption, and their detailed security proof can be found in [42], [43]. The TA generates its own public and secret key for each client and distributes it to the corresponding client through a secure channel, so that it does not affect the security of the encryption algorithm.

When $k < N - 1$ malicious clients collude with the server, PPSFL can ensure the data security of honest clients as follows.

In PPSFL, the client model uses multi-input functional encryption for secure aggregation, assuming that a certain round of aggregation function is $f_y = y_1 \cdot W_1 + \cdots + y_N \cdot W_N$, the server sends a key request to the TA about the function coefficient $y_1, \cdots, y_N$. Since the obtained decryption key can only solve $f_y = y_1 \cdot W_1 + \cdots + y_N \cdot W_N$, there is no way to obtain information about $(W_i)_{i \in N}$ (ensured by the security of the multi-input functional encryption). In the presence of a malicious client conspiring with the server, as long as there are at least two honest clients, the server can only obtain $y_{h_1} \cdot W_{h_1} + y_{h_2} \cdot W_{h_2}$, and cannot infer the model plaintext information of a single client. Therefore, PPSFL ensures the privacy of the honest client when $k < N - 1$ malicious clients conspire with the server.

PPSFL is resistant to speculative attacks from malicious servers. In its client model secure aggregation process, a malicious server can obtain $W_1$ plaintext by modifying the weighting coefficient value, such as $y_2, \ldots, y_N = 0$. In order to prevent this kind of inference attack, PPSFL sets a weight coefficient filtering mechanism at TA side to reject function key requests containing $N - 1 \ 0$ value coefficients, so as to ensure that the malicious server cannot infer the plaintext information of specific client model parameters by tampering with the weighting coefficient values.

VII. EVALUATION

A. Experimental setup

Evaluations are carried out on a machine equipped with an Intel(R) Core(TM) i9 24-Core CPU at 3.19 GHz, 64GB of RAM. All the implementations are written in Python (V3.7). Specifically, we use numpy and gmpy2 to implement the secure computational protocols and Pytorch to achieve model training. The model is trained with the mini-batch SGD optimization algorithm at a learning rate of 0.1 with $L$ local epochs in one aggregation round. The batch size is 60 and the total aggregation round is set to 50 (20) for $L = 1$ ($L = 5$).

We use the publicly available MNIST of 28*28 pixel grayscale images with 60,000 samples for training and 10,000 samples for testing. The number of clients is 5, and each client is randomly assigned 12000 samples as the local training dataset.
1) Model and implementation: We use a Multilayer Perceptron (MLP) model and a convolutional neural network (CNN) for evaluation (cf. Figure 3).

The MLP model we use has three hidden fully-connected layers with linear and relu functions each and a softmax activation function as the output layer. Dropout is used to mitigate overfitting. As input, each $28 \times 28$ pixel gray-scale image is converted to a vector with a length of 784. When splitting happens between two fully-connected layers, we conduct the secure forward and backward propagation algorithm of PPSFL to achieve dot product operation of the activation matrix and the weight matrix.

Our CNN model implements two layers of convolution, max pooling and linearization each with Relu as the hidden layer activation function and Softmax activation for the output layer. Dropout is used to mitigate overfitting. When splitting occurs between the convolutional layers of the CNN model, it is necessary to unfold and reorganize the activations of the cut layer to adapt to the size of the convolutional kernel on the next layer and to perform matrix multiplication. Therefore, we apply Img2Col to accelerate convolution and adapt it for functional encryption. The dot product operation of the corresponding elements of the input feature submatrix and the weight matrix in the convolution is converted into the multiplication and addition operation of row and column vectors in the matrix operation. Therefore, we can apply our secure forward and backward propagation. If the splitting happens between a convolution layer and a fully-connected layer, the two-dimensional feature map of the convolution layer is transformed into a one-dimensional vector so as to operate the dot product. In this case, secure forward and backward propagation is the same as in the MLP model.

2) Comparison schemes and metrics: For ease of description, we describe the models by the number of neurons per layer: 784 → 256 → 128 → 64 → 10 for the MLP and 784 → 2240 → 320 → 50 → 10 for the CNN. To evaluate the performance of each scheme at different split locations, we define two split locations: “$X$” – 1 refers to a split between hidden layers 1 and 2: 784 → 256|split|128 → 64 → 10 for the MLP and 784 → 2240|split|320 → 50 → 10 for the CNN; Likewise, “$X$” – 2 refers to a split between hidden layers 2 and 3: 784 → 256 → 128|split|64 → 10 for the MLP and 784 → 2240 → 320|split|50 → 10 for the CNN.

To evaluate the performance of PPSFL, we tested the accuracy, computational and communication cost and compared them to centralized training (CL), federated learning (FL), split learning (SL), splitfed learning (SFL), and splitfed learning with differential privacy (SFL-DP). For PPSFL we set $\lambda = 256$. For SFL-DP, the client adds Gaussian noise to the client-model gradient before model updates to achieve differential privacy. As the original code for SFL with DP is not publicly available, we reproduce the scheme and set the differential privacy-related parameters as described in [7]: $\delta = 1e^{-5}, \epsilon = 0.5, \sigma = 1.3$. In addition, we compare PPSFL to the multi-sourced neural network training scheme NN-EMD based on functional encryption [44], which trains a DNN over encrypted multiple datasets collected from multiple sources. For ease of comparison, we select horizontal partitioning-based training NN-EMD (encrypting each batch of data sample matrix of each data source). It is worth noting that in NN-EMD, functional encryption and decryption happen between the input layer and the first hidden layer, which can be regarded as a particular case of splitting the model between the input layer and the first hidden layer but without a federation process. Although the authors show that NN-EMD can be integrated into SplitNN, the theoretical and implementation details are not given and they only consider the MLP model. NN-EMD-SplitNN trains on multiple clients’ samples simultaneously, which raises the concern of overfitting due to differences in the training sequence and the local data size of the clients. Moreover, for PPSFL and NN-EMD, we encode the floating-point type model parameters or activations into integers before encryption with a precision of 3.

B. Discussion of results

1) Accuracy: Figure 4 shows the accuracy of each scheme. The split location of SFL, SFL-DP, and PPSFL defaults to 784 → 256 → 128|split|64 → 10 for MLP and 784 → 2240 → 320|split|50 → 10 for CNN. From Figure 4a, 4b, 4c and 4d, we can see that PPSFL does not lead to a decrease in model accuracy, and with more local epochs in one aggregation round, the model converges quicker. From Figure 4c and 4f, we can see that the model accuracy of SFL and PPSFL is not affected by the split location. In addition, the accuracy of the SFL-DP scheme is lowest because the Gaussian noise added to the gradient.

2) Computational cost: We discuss the computational cost of each scheme for the model training and the model aggregating phases and report the results in Table I. We use marker “×” in the table to denote that there is no corresponding operation.

As shown in Table I, SFL reduces the computational cost on the client by transferring part of the computational cost to the server. With fewer client-model parameters, the computational cost of model training and aggregation is reduced for the server and increased for the server. With the use of functional encryption, PPSFL increases the computational costs at the training and aggregation phases for encryption and decryption. Comparing PPSFL-2 to PPSFL-1, we see that increased client-model layers translate to less activations that need to be encrypted, thus reducing the computational load during training. During aggregation, because the client model has an additional layer, the model parameters that need to be encrypted increase.
For NN-EMD, the client needs to encrypt the data samples which is more than the activations in the cut layer, thus resulting in a higher computational cost for one mini-batch. Note that one local epoch includes many mini-batch training rounds (in our case, 200). Thus PPSFL is superior to NN-EMD. By conducting more local epochs of training before aggregation, PPSFL further reduces the computational cost for aggregation.

As shown in Table 1, with more layers on the client side, the training and aggregating time for the CNN also increases. In PPSFL-1, the activations shape of the cut layer is $[60,10,12,12]$, and the second convolutional layer has 10 convolutional kernels of shape $[5,5]$, so the activation matrix to be encrypted after the reorganization by Img2Col is $\mathbf{X}^{(2)} \in \mathbb{R}^{5 \times 5 \times 12 \times 12}$, which is more than the activations in the cut layer, thus resulting in a higher computational cost in the encryption and decryption in the aggregation phase. In PPSFL-2, the splitting occurs between the convolutional layer and the fully connected layer, and the activations of each sample are directly expanded into a one-dimensional vector, so for each batch, the activation matrix to be encrypted is $\mathbf{X}^{(2)} \in \mathbb{R}^{60,320}$. Therefore, PPSFL-2 has a lower computational cost than PPSFL-1 in the encryption and decryption in the training phase. In the aggregation phase, PPSFL-2 has one more layer parameter on the client model to be encrypted and aggregated, thus having more time cost than PPSFL-1. Compared to the MLP model, the CNN model has fewer parameters and therefore a smaller computational cost in the aggregation phase.

Furthermore, when we employ $L > 1$ local epochs in one aggregation round, we can deduce the computational cost for model aggregation.

![Fig. 4. Accuracy of schemes with different split locations or different numbers of local epochs](image)

### TABLE I

<table>
<thead>
<tr>
<th>Stage</th>
<th>Model training(one mini-batch($b=60$))</th>
<th>Model aggregating(one aggregation round)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train</td>
<td>Encrypt</td>
</tr>
<tr>
<td>MLP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FL</td>
<td>0.00714s</td>
<td>$\times$</td>
</tr>
<tr>
<td>SF1</td>
<td>0.0015/8s</td>
<td>$\times$</td>
</tr>
<tr>
<td>SF2</td>
<td>0.00214s</td>
<td>$\times$</td>
</tr>
<tr>
<td>PPSFL-1</td>
<td>0.00162s</td>
<td>0.16s</td>
</tr>
<tr>
<td>PPSFL-2</td>
<td>0.00231s</td>
<td>0.07s</td>
</tr>
<tr>
<td>NN-EMD</td>
<td>$\times$</td>
<td>1.56s</td>
</tr>
<tr>
<td>CNN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FL</td>
<td>0.00836s</td>
<td>$\times$</td>
</tr>
<tr>
<td>SF1</td>
<td>0.00301s</td>
<td>$\times$</td>
</tr>
<tr>
<td>SF2</td>
<td>0.00554s</td>
<td>$\times$</td>
</tr>
<tr>
<td>PPSFL-1</td>
<td>0.00355s</td>
<td>33.263s</td>
</tr>
<tr>
<td>PPSFL-2</td>
<td>0.00515s</td>
<td>0.29208s</td>
</tr>
</tbody>
</table>

### TABLE II

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Training</th>
<th>Aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL</td>
<td>$\times$</td>
<td>$2 \cdot M \cdot N_w$</td>
</tr>
<tr>
<td>SFL</td>
<td>$2 \cdot M \cdot \eta \cdot N_w$</td>
<td>$2 \cdot M \cdot \eta \cdot N_w$</td>
</tr>
<tr>
<td>PPSFL-1c</td>
<td>$(2 \cdot C + M) \cdot d \cdot n_{c}$</td>
<td>$(C + M \cdot \eta \cdot N_w$</td>
</tr>
<tr>
<td>PPSFL-con</td>
<td>$(2 \cdot C + M) \cdot d \cdot X_H \cdot X_W$</td>
<td>$(C + M) \cdot \eta \cdot N_w$</td>
</tr>
<tr>
<td>NN-EMD</td>
<td>$2 \cdot C \cdot d \cdot n_{c}$</td>
<td>$\times$</td>
</tr>
</tbody>
</table>
3) Communication cost: We discuss the communication cost of PPSFL and compare it to FL, SL, and NN-EMD. Suppose the dataset size of the client is \( d \), mini-batch size is \( b \). In order to unify the encryption and decryption length, if \( d/b \) is not even, the last batch of samples with less than \( b \) samples is discarded. For ease of description, here we assume that \( d \) is divisible by \( b \). \( n_c \) refers to the number of neurons in the cut layer. The weight and bias parameters of the global model are \( N_w \), and the fraction of the client model is \( \eta \). Assume that the plaintext size of each gradient parameter and model parameter is \( M \), and the size of the encrypted ciphertext value is \( C \). Because the label data has little communication compared to the training data and encrypted data, and it has nothing to do with whether encryption is used and which layer is split, the following analysis ignores the label data communication.

When splitting happens between two convolution layers, because of the use of the Img2Col algorithm to recognize the output activation of the cut layer, the size of the matrix that needs to be encrypted becomes larger than the activation matrix. Specifically, suppose the weight parameter of the next layer is of the shape \([C_{out}, C_{in}, H_k, W_k]\), in which \( C_{in} \) and \( C_{out} \) are the number of kernels’ channel of cut layer and its next layer, \( H_k \) and \( W_k \) are the height and width of the kernel. To operate the matrix dot product, it is reshaped to matrix \( W \) of the shape \( C_{out} \times (C_{in} \cdot H_k \cdot W_k) \). Suppose the output of the cut layer has the shape \([h, C_{in}, Img_H, Img_W]\), in which \( Img_H \) and \( Img_W \) are the height and width of the image output of the cut layer. The recognized activation matrix \( X \) is of the shape \((C_{in} \cdot H_k \cdot W_k) \times [(Img_H - H_k + 1) \cdot (Img_W - W_k + 1)] \cdot b\).

For ease of description, we denote \((C_{in} \cdot H_k \cdot W_k) \) and \([(Img_H - H_k + 1) \cdot (Img_W - W_k + 1)] \) as \( X_H \) and \( X_W \). When splitting happens between the convolution layer and the fully-connected layer or two fully-connected layers, the communication cost is mainly included in the first hidden layer, the aggregation communication cost only decreases a little with splitting the model. The CNN model parameter is mainly included in the fully-connected layers, therefore with splitting, the aggregation communication cost decreases a lot. For NN-EMD, the data samples that need to be encrypted are much more than the activations in the hidden layer, thus having a much higher communication cost than PPSFL.

In order to visualize the communication overhead brought by our encryption to model learning, Table III shows the specific values under different schemes and different split locations. Specifically, in our evaluation, each client has 12000 data samples. The plaintext size of each activation and model parameter is \( M = 32\text{bits} \), and ciphertext size \( C = 255\text{bits} \).

As shown in the table, with model splitting, the neural number of hidden layer 1 is larger than that of hidden layer 2, therefore the communication cost of SFL-1 and PPSFL-1 in the training phase is higher than SFL-2 and PPSFL-2, respectively. For the aggregation communication cost, as the MLP model parameter is mainly included in the first hidden layer, the aggregation communication cost only decreases a little with splitting the model. The CNN model parameter is mainly included in the fully-connected layers, therefore with splitting, the aggregation communication cost decreases a lot.

### Table III

<table>
<thead>
<tr>
<th>Scheme</th>
<th>( n_c ) or ( X_H \times X_W )</th>
<th>fraction of client model ( \eta )</th>
<th>Communication cost (Training)</th>
<th>Communication cost (Aggregation)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MLP</strong></td>
<td></td>
<td></td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>FL</td>
<td>( \times )</td>
<td>100%</td>
<td>23.45MB</td>
<td>1.85MB</td>
</tr>
<tr>
<td>SFL-1</td>
<td>256</td>
<td>82.8%</td>
<td>11.73MB</td>
<td>1.53MB</td>
</tr>
<tr>
<td>SFL-2</td>
<td>128</td>
<td>96.3%</td>
<td>1.2%</td>
<td>1.78MB</td>
</tr>
<tr>
<td>PPSFL-1</td>
<td>256</td>
<td>82.8%</td>
<td>577.12MB</td>
<td>6.88MB</td>
</tr>
<tr>
<td>PPSFL-2</td>
<td>128</td>
<td>96.3%</td>
<td>100.40MB</td>
<td>8.00MB</td>
</tr>
<tr>
<td>NN-EMD</td>
<td>( \gamma )</td>
<td>%</td>
<td>0.0%</td>
<td>0.17MB</td>
</tr>
</tbody>
</table>

### Table II

<table>
<thead>
<tr>
<th></th>
<th>Communication cost (Training)</th>
<th>Communication cost (Aggregation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv</td>
<td>100%</td>
<td>205.08MB</td>
</tr>
<tr>
<td>FL</td>
<td>( % )</td>
<td>( % )</td>
</tr>
<tr>
<td>SFL-1</td>
<td>( % )</td>
<td>( % )</td>
</tr>
<tr>
<td>SFL-2</td>
<td>( % )</td>
<td>( % )</td>
</tr>
<tr>
<td>PPSFL-1</td>
<td>( % )</td>
<td>( % )</td>
</tr>
<tr>
<td>PPSFL-2</td>
<td>( % )</td>
<td>( % )</td>
</tr>
</tbody>
</table>

4) Impact of encoding precision on the model accuracy:

In PPSFL, we use functional encryption which only works on integer inputs. Therefore, it is necessary to encode the floating-point type model parameters or activations into integers before encryption. We set the precision to 3 and 5, which means expanding the floating-point number by multiplying \( 10^3 \) and \( 10^5 \). Figure 5 shows the model accuracy of PPSFL-2 on the MLP model with different precision. We can see that the encoding precision setting has little impact on the model accuracy.
VIII. CONCLUSION

We have proposed PPSFL, a novel privacy-preserving split federated learning scheme based on functional encryption to protect data privacy and reduce the computational and communication burden of distributed clients. By distributing most of the model on the server side, PPSFL reduces the training overhead of clients and provides better model property protection. The secure forward and backward propagation algorithm achieves secure model training with encrypted smashed data sent from clients to the server. The secure model aggregation algorithm achieves model synchronization among distributed parties while protecting local model privacy. Our evaluation shows that PPSFL has significant advantages in training time and communication overhead comparing other privacy-preserving model training schemes. We also discussed that PPSFL can achieve efficient distributed model optimization by selecting a suitable split location while providing sufficient privacy protection. Furthermore, we give an implementation of PPSFL on MLP and CNN models, different from many existing cryptography-based deep learning schemes that only focusing on the MLP model. With this, PPSFL is able to implement privacy-preserving deep learning on models containing Le Net, AlexNet, VGG Net, etc., which are more practical and popular in real-world application scenarios.

ACKNOWLEDGMENT

This study is supported by the National Natural Science Foundation of China under grant number 62072356.

REFERENCES


**Jing Ma** received the BS degree in applied mathematics from Xidian University, China, in 2016. She is currently working towards the PhD degree in security of cyberspace at Xidian University, China. Her research interests include privacy and security of deep learning, federated learning and applied cryptography.

**Xiuxiang Lyu** studied in Xidian University from 1997 to 2007, and received her MS and PhD degrees in 2004 and 2007 respectively. She is currently a full professor of Xidian University.

Her research interests lie in anonymous networks, special cryptographic algorithms, cyber security, privacy and security of deep learning. In recent years, as the project leader, she has successively completed a project of National Key Basic Research Program of China, a key project of China National Natural Science Foundation, and several general projects of China National Natural Science Foundation, etc.

**Yong Yu** is currently a Professor with Shaanxi Normal University, China. He holds the prestigious one hundred talent Professorship of Shaanxi Province as well. He has authored over 120 referred journal and conference papers. His research interests are cryptography and its applications. He is an Associate Editor of IEEE Transactions on Computers.

**Stephan Sigg** (M’04, PhD’08) is an Associate Professor at Aalto University in the Department of Communications and Networking. With a background in the design, analysis and optimisation of algorithms for distributed and ubiquitous systems, he focuses on sensing systems for environmental perception and Usable (perception-based) Security. Especially, his work covers proactive computing, distributed adaptive beamforming, context-based secure key generation and device-free passive activity recognition. Stephan is an editor of the Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT), and an area editor for Elsevier Pervasive and Mobile Computing and for the Elsevier Journal on Computer Communications. He has served on the organizing and technical committees numerous prestigious conferences including IEEE PerCom, ACM Ubicomp, IEEE ICDCS.