Secure Partial Aggregation: Making Federated Learning More Robust for Industry 4.0 Applications

Jiqiang Gao, Baolei Zhang, Xiaojie Guo, Thar Baker, *Senior Member, IEEE*, Min Li, and Zheli Liu

***Abstract*—Big data, due to its promotion for industrial intelligence, has become the cornerstone of the Industry 4.0 era. *Federated learning*, proposed by Google, can effectively integrate data from different devices and different domains to train models under the premise of privacy preservation. Unfortunately, this new training paradigm faces security risks both on the client side and server side. This article proposes a new federated learning scheme to defend from client-side malicious uploads (e.g., backdoor attacks). In addition, we use cryptography techniques to prevent server-side privacy attacks (e.g., membership inference). The *secure partial aggregation* protocol we designed improves the privacy and robustness of federated learning. The experiments show that models can achieve high accuracy of over 90% with a proper upload proportion, while the accuracy of the backdoor attack decreased from 99.5% to 0% with the best result. Meanwhile, we prove that our protocol can disable privacy attacks.**

***Index Terms*—Federated learning, Industry 4.0, privacy preservation, secure aggregation.**

I. Introduction

THE fourth industrial revolution, also known as Industry 4.0, is one of the most trending academic and industrial circles topic. It focuses on cyber-physical systems (CPS) [1], cloud computing [2], Internet of Things (IoT) [3], and other modern technologies. Integration of multiple systems accelerates the age of big data, which increases the productivity of enterprises and provides a concrete foundation for Industry 4.0. Different organizations and different devices generate a large amount of data every day. Integrating and analyzing the big data help to increase the productivity, reduce production costs, and promote manufacturing upgrades.

However, due to privacy regulations (e.g., general data protection regulation (GDPR)) and commercial competition, more and more data silos emerge and prevent data collaboration severely. Federated learning [4], proposed by Google, aims to break data silos and integrate data over edge devices and organizations under the premise of privacy preservation. This new learning paradigm trains a model across multiple clients without exchanging local data. It avoids the risk of data leakage when sharing data for analysis, such as within the same CPS or between different CPS. The federated learning has been applied to many fields, such as intrusion detection system [5] and some protocol-level improvements [6], [7] have also been proposed.

Although the client only shares its model updates instead of the raw training data, federated learning remains vulnerable to some attacks. For example, the works [8]–[10] show that the server may exploit a model update vector to infer private information of one client. Also, the malicious clients can inject backdoor [11], [12] into model updates and result in the global model to making wrong predictions. Once the model is poisoned, the industrial infrastructure will face serious security risks. We show the overview of federated learning and potential attack surfaces in Fig. 1.

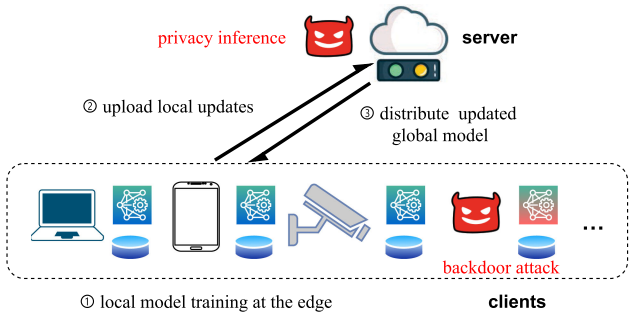


Fig. 1. Potential attacks from the server side and the client side in federated learning.

Manuscript received July 29, 2021; revised November 4, 2021 and December 28, 2021; accepted January 18, 2022. Date of publication January 25, 2022; date of current version June 13, 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 62032012 and in part by the National Key Research and Development Program of China under Grant 2020YFB1005700. Paper no. TII-21-3239. *(Jiqiang Gao, Baolei Zhang, and Xiaojie Guo contributed equally to this work.) (Corresponding authors: Min Li; Zheli Liu.)*

Jiqiang Gao, Baolei Zhang, Xiaojie Guo, Min Li, and Zheli Liu are with the Tianjin Key Laboratory of Network and Data Security Technology, College of Cyber Science, College of Computer Science, Nankai University, Nankai 300071, China (e-mail: jiqiang.gao@mail.nankai.edu.cn; zhangbaolei@mail.nankai.edu.cn; xiaojieg.veda@gmail.com; limintj@nankai.edu.cn; liuzheli@nankai.edu.cn).

Thar Baker is with the Department of Computer Science, College of Computing and Informatics, University of Sharjah, Sharjah 27272, UAE (e-mail: tshamsa@sharjah.ac.ae).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2022.3145837.

Digital Object Identifier 10.1109/TII.2022.3145837

Some works [13]–[16] have proposed solutions to defend against client-side or server-side attacks. One direction is to design robust aggregation rules. Another direction is to utilize cryptography to design secure aggregation protocols. However, no defense scheme that can defend against both side attacks at the same time.

A. Our Work

In basic aggregation scheme, the lack of restriction and verification enables malicious clients and server to attack. In this article, we propose an aggregation algorithm that can defend both the client-side and server-side malicious behaviors. We *design a new secure partial aggregation protocol to make the federated learning process more robust and secure* by the following.

1) Restricting the client to upload required proportion but not all model updates to the server.
2) Guaranteeing the server cannot infer private information through encrypted model updates and providing the ability to detect dishonest behaviors of clients in a secure way.

Existing secure aggregation protocols [14], [15] cannot be directly adapted to achieve secure partial aggregation. There are two challenges: 1) the server should aggregate all partial updates without knowing something about the individual update and 2) the server should detect an invalid partial update. To address the abovementioned two challenges, we utilize the distributed homomorphic encryption and provide optimization to reduce its overhead in Section V. Experimental results show that the model can converge to a high accuracy of over 90% under partial uploading. With a proper upload proportion, the accuracy of the global model is almost unchanged, while the attack success rate dropped from 99.5% to 0% with the best result. Meanwhile, the inference attack no longer works, even assuming that the server can obtain the model updates in a possible way.

The rest of this article is organized as follows. In Section II, we give some preliminaries about the *FedAvg* and the secure federated learning. In Section III, we describe the main part of our protocol except for cryptographic operations and the whole protocol is in Section V. In Section IV, the experimental results and analysis are provided. Finally, Section VI concludes this article.

II. Related Work

The federated learning technique has been widely applied in IoT, CPS, etc., for achieving privacy preservation. In academia, more and more studies [1], [5], [17] have applied traditional centralized machine learning algorithms to federated learning frameworks. We first define the parameters and descriptions used in the following sections in Table I.

A. Robust Federated Learning

Aggregation algorithms are the core of federated learning. The most popular is *FedAvg* [4], which Google proposes as a basic aggregation protocol. The *FedAvg* algorithm is executed as follows: At the tth round, the server selects a subset of clients C_t (the size of C_t is m) and sends the current global model G^t to them. Each selected client in C_t uses its local data to train its local model, and gets a new local model L^{t+1}. The server averages the received m updates $(L^{t+1} - G^t)$ to obtain a new global model for next round

$$G^{t+1} = G^t + \frac{\eta}{m}\sum_{i=1}^{m}(L_i^{t+1} - G_i^t). \tag{1}$$

The η is regarded as the learning rate of the global model. To solve the client-side Byzantine attack, several robust aggregation algorithms have also been proposed.

Krum: Krum [18] selects one of the m model updates that minimizes the sum of squared distances to its $m - f$ closest model updates, where f is the number of Byzantine participants. When $f < \frac{m}{2} - 1$, Krum guarantees the model convergence theoretically.

Median: In median [13] aggregation, for the jth model parameter of the m local models, the master device calculates the median and takes it as the jth parameter of the global model. The median aggregation rule can achieve order-optimal statistical rates when the loss function is strongly convex.

Trimmed mean: Like the median aggregation rule, the trimmed mean [13] aggregates each model parameter independently. However, for the jth model parameter of the m local models, the master device removes the largest and smallest β of them and computes the mean of the remaining $m - 2\beta$ parameters as the jth parameter of the global model in the trimmed mean aggregation rule. This rule can also achieve order-optimal statistical rates for strongly convex losses.

We give a comparison in Table II to show the differences between several robust aggregation algorithms and ours.

TABLE I
PARAMETERS AND DESCRIPTIONS IN THIS ARTICLE

Parameters	Descriptions
n	the number of all clients
l	the number of model parameters
d	the upload proportion
C_t	the subset of selected clients at the t-th round
m	the size of C_t
η	the learning rate of global model
L_i^t	client-i's local model at the t-th round
G^t	the global model at the t-th round
Δ_i^t	client-i's model update at the t-th round
∇	gradient operator
V_i	the flattened vector derived from Δ_i
$v_i[j]$	the j-th entry of the vector V_i

TABLE II
COMPARISON BETWEEN SEVERAL ROBUST AGGREGATION ALGORITHMS AND OURS

Method	Defend against malicious clients	Defend against malicious server	Model updates are invisible
Krum	✓	✗	✗
Median	✓	✗	✗
Mean	✓	✗	✗
Ours	✓	✓	✓

B. Secure Federated Learning

The parameter server can exploit updates to recover privacy information or tamper with global model parameters. To make federated learning more secure at the server-side, some secure protocols have introduced cryptography technology.

The most wide-used secure protocol is *SecAgg* [14], which designs a double-masking strategy to encrypt each update before uploading. Since the additional masks of these updates add up to zero, the server can only obtain the sum of these updates after performing add operation. In this way, it is almost impossible for the server to perform a privacy inference attack.

Some other works [15], [16] are concerned with whether the server honestly performs aggregation calculations. The design of a verification mechanism is necessary since the malicious aggregation can lead to global model poisoning or failure. Combined with cryptography, they propose secure verification protocols to prevent the server from tampering with model parameters. However, we point out that secure protocol and robust aggregation are mutually exclusive. The robust aggregation is hard to apply when adopting a secure protocol at the server side because it is based on plaintext updates. In short, federated learning needs a protocol that can meet privacy preservation and limit malicious client behaviors. Unfortunately, there are no such studies yet.

III. Secure Partial Aggregation Protocol

We design a secure partial aggregation protocol to simultaneously mitigate the risks caused by malicious clients and the privacy leakage caused by malicious server. By forcing the clients to upload randomly selected partial model updates, we can meet the robustness requirement of limiting the malicious behaviors of clients. Meanwhile, we can protect the privacy of clients' updates by encrypting them and performing aggregation over these encrypted updates. Our secure partial aggregation protocol achieves three goals.

1) It promises the convergence and accuracy of the global model, which is the essential requirement of federated learning.
2) It defends the client-side backdoor attack and the server-side inference attack simultaneously.
3) It is lightweight and can be easily deployed in existing federated learning frameworks for better privacy preservation.

A. Protocol Overview

Our secure partial aggregation protocol has two innovations for privacy preservation. First, we design parameter d to control the upload proportion of model updates of each client, which makes it hard for malicious clients to perform model substitution attacks, such as backdoor attacks. Because the existing backdoor methods are based on the full parameters poisoning, and the upload of partial update parameters will significantly reduce the toxicity of the bad model. Second, we introduce distributed homomorphic encryption and propose a secure scheme to make the model updates are invisible to the server. In this way, the server cannot obtain individual model updates for inferring information based on plaintext updates. Note that although the existing inference attacks have not been implemented on random partial model parameters, we consider the partial model updates in plaintext still risk privacy leakage for the sake of better security. To facilitate the understanding, we simplify the process of our full protocol in Algorithm 1 and omit the details of cryptography steps in this section. We describe the simplified protocol and term it *PartFedAvg*. There are several protocol details that we use function names to express, such as **Enc**$(\cdot)$, **RandomUpdateSelection**$(\cdot)$, and **ConstructUpdate**$(\cdot)$. The complete protocol is described in Section V. We describe the execution process of the protocol from both the client side and the server side.

Algorithm 1: The Framework of Secure Partial Aggregation.

Input: global model G^0 and local model L_i; n clients, with each one holding a private dataset D_i; update proportion d, which controls the number of parameter uploaded each round.
Server:
Initialize global model G^0 and send it to n clients
for $t = 0$ to T **do**
  Select a subset C_t from n clients to submit, $|C_t| = m$
  for $i = 1$ to m **do**
    $[\![V_i^{t'}]\!] \leftarrow \mathbf{ClientUpdate}(G_i^t, D_i)$
  end for
  $\Delta_{\text{agg}}^t \leftarrow \mathbf{ConstructUpdate}(\{[\![V_i^{t'}]\!] \mid i \in C_t\})$
  $G^{t+1} = G^t + \Delta_{\text{agg}}^t$
  Send G^{t+1} to n clients.
end for

ClientUpdate(L_i, D_i)**:**
$L_{\text{init}} \leftarrow L_i$
for each local epoch **do**
  $L_i \leftarrow L_i - \alpha \nabla L(L_i, D_i)$
end for
$\Delta_i \leftarrow L_i - L_{\text{init}}$
$V_i' \leftarrow \mathbf{RandomUpdateSelection}(\Delta_i, \text{d})$
$[\![V_i']\!] \leftarrow \text{Enc}(\Delta_i^p)$
return $[\![V_i']\!]$ to the server
Output: a convergent global model G^T

1) Client: Random Update Selection: We take the tth round of training process as an example. First, the ith client receives the current global model G^t as its current local model L_i^t. Then, the client trains it with its local dataset D^i, getting a new local model L_i^{t+1}. The client regards its local model update as Δ_i, where

$$\Delta_i^t = L_i^{t+1} - L_i^t.$$

In the original *FedAvg* protocol, the Δ_i^t will be directly sent to the server after this step. But in our *PartFedAvg* protocol, along with the initial global model G^0, each client receives an upload proportion d from the server. Before uploading model updates at each round, the clients need to execute the additional

RandomUpdateSelection$(\cdot)$ with Δ_i^t and d. The function selects $d \times 100\%$ parameters by random algorithm and outputs the results.

More specifically, the update Δ_i is a multilayer structure, the same as the global model structure. First, the update is expended into a flattened vector V_i with l entries, represented as $V_i = (v_i[1], v_i[2], \ldots, v_i[l])$. Then, each client randomly refills $l \times (1-d)$ entries with 0, where $0 \leq d \leq 1$, leaving the rest unchanged. Finally, we obtain the partial update V_i'. We give a formal expression as follows:

$$V_i = (v_i[j] \mid 0 < j \leq l) \leftarrow \mathbf{flatten}(\Delta_i)$$
$$V_i' = (v_i'[j] \mid 0 < j \leq l) \leftarrow \mathbf{random\_zero}(V_i)$$

where

$$v_i'[j] = \begin{cases} v_i[j] & \text{the index j is selected} \\ 0 & \text{the index j is unselected} \end{cases}.$$

After this operation, the client's model update parameters are randomized up by zero. Because the value of d is very small, it is almost impossible for an attacker to perform a backdoor attack with so few parameters.

2) Server: Update Construction: Once all selected clients submit their partial model updates, the server needs to construct a complete average update Δ^t to get the new global model G^{t+1}. Next, we discuss how the server aggregates all partial submissions from clients.

Concretely, there are m partial updates vector V_i' submitted at the tth round. The server needs to calculate the average of the nonzero values of each entry of these vectors V_i'. The function **ConstructUpdate**$(\cdot)$ counts the number $z^t[j]$ of nonzero values in each entry of these vectors V_i'. The averaged V_{agg}^t can be obtained as follows:

$$V_{\text{agg}}^t = (v_{\text{agg}}^t[j] \mid 0 < j \leq l)$$

where

$$z^t[j] = \begin{cases} z^t[j] & v_i'[j] = 0 \quad (0 < i \leq m, 0 < j \leq l) \\ z^t[j] + 1 & v_i'[j] \neq 0 \quad (0 < i \leq m, 0 < j \leq l) \end{cases}$$

$$v_{\text{agg}}^t[j] = \begin{cases} \frac{\eta}{z^t[j]} * \sum v_i'[j] & (z^t[j] \neq 0) \\ 0 & (z^t[j] = 0) \end{cases}.$$

$z^t[j]$ is initialized to 0. If the final $z^t[j] = 0$, it indicates that the jth entry is not contributed by anyone. Thus, the server sets $v_{\text{agg}}^t[j] = 0$ for avoiding division by zero, and does not update this entry. Once the V_{agg}^t is calculated, the server transforms the vector V_{agg}^t to multilayer structure Δ_{agg}^t and add it to G^t to get the new global model G^{t+1}.

B. Explanation of the Method

The purpose of our method is to resist both the client-side and server-side threats in federated learning while ensuring the convergence and usability of the global model. In the following we will explain why it works from three aspects.

1) Model Convergence: Our protocol *PartFedAvg* is based on *FedAvg*, and the *FedAvg* protocol has proved that the model can achieve convergence under both i.i.d. (independent identically distributed) [19] and non-i.i.d. data [20] settings when clients upload the complete model updates. Unlike *FedAvg*, *PartFedAvg* requires the client to submit only $d \times 100\%$ of update parameters. The work [21] has stated that the average of the subsampled updates is an unbiased estimator of the average of the original updates, where the subsampled update is a random subset of the original update parameters. Based on this point, our partial aggregate method ensures that the model can achieve convergence. The experiment in Section IV-B shows that the model can converge with only a little loss in accuracy.

2) Defend the Backdoor Attack: Our secure partial aggregation protocol invalids the backdoor attack in the following two aspects.

1) The key to a successful attack is to replace the global model parameters with backdoor model parameters completely [11]. With our protocol, each client can only upload a certain percentage of model updates per round according to the parameter d. Thus, if the malicious client wants to upload whole local backdoor model, it needs to be selected at least $\frac{1}{d}$ rounds. There are two points to note here. One is that being selected $\frac{1}{d}$ rounds in a short time is almost impossible. The second is that each round of aggregation operation in the protocol will bias the global model toward benign clients. Even if the malicious client uploads the backdoor model through multiple rounds, the attack effectiveness is almost negligible.
2) Scaling up the backdoor parameters is considered as another powerful backdoor attack in [11]. To substitute the global model, the malicious client scales up the backdoor model updates by $\frac{m}{\eta}$. However, with our *PartFedAvg*, the malicious client cannot scale each entry of updates appropriately since the benign clients randomly select local parameter entries, resulting in the value of each $z^t[j]$ is unknown to the malicious client. Without this adversarial knowledge, it is too hard for malicious clients to perform the backdoor attack in one shot.

3) Defend the Inference Attack: Our secure partial aggregation protocol has two advantages in terms of privacy protection. First, since each client uploads only a small part of the update in each round, the information disclosed by these updates is so limited that it is almost impossible for the server-side attacker to infer private information. Second, whether in the transmission process or the aggregation process, the parameters are encrypted to protect the client's privacy to the greatest extent. Thus, the method resist the server-side inference attacks.

IV. Evaluations

A. Experiment Settings

In order to verify the effectiveness of our method, we evaluate 1) the model convergence and 2) the success rate of attacks under both *FedAvg* and *PartFedAvg*. We implement all evaluations on two image datasets (MNIST and CIFAR-10) and one text dataset (LOAN). More details about the datasets are listed in Table III. Following previous works [11], [12], we set two-steps stochastic gradient descent (SGD) for local training. We assign each client

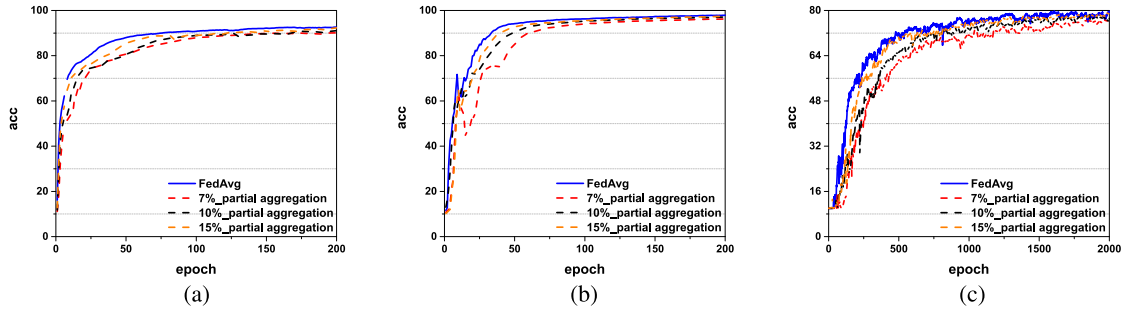


Fig. 2. Process of model convergence when using *FedAvg* and *PartFedAvg* on three datasets. (a) LOAN. (b) MNIST. (c) CIFAR.

TABLE III
DATASETS USED FOR EXPERIMENTS

Datasets	Classes	number/class	size/sample	Model
MNIST	10	6000	28*28	3 linear layers
CIFAR-10	10	5000	32*32	ResNet18
LOAN	9	2,260,668	91	3 linear layers

TABLE IV
NUMBER OF TRAINING ROUNDS REQUIRED FOR THE MODEL TO REACH CONVERGENCE UNDER DIFFERENT UPLOAD PROPORTIONS

Upload proportion	LOAN	MNIST	CIFAR-10
PartFedAvg: 7%	200	200	1853
PartFedAvg: 10%	168	137	1473
PartFedAvg: 15%	129	126	1121
FedAvg: 100%	79	99	915

a local dataset D_i according to the Dirichlet distribution to simulate non-i.i.d. condition. The learning rate η of global model is set to 0.1. Some other parameters will be specified in the following experiments. We use pytorch to train all models on NVIDIA 2080Ti GPU.

B. Evaluations on Model Convergence

In experiments, the server randomly selects m clients out of n per round to participate in training, where $m = 10$ and $n = 100$. To evaluate the impact of upload proportion d on the model accuracy and training efficiency, we set $d \times 100\% = 7\%$, 10%, 15%, and 100% (*FedAvg*), respectively, and train the global model for 200 epochs on the MNIST and the LOAN, 2000 epochs on the CIFAR-10. When the accuracy of the model reaches the highest point steadily, we consider that the model goes convergence.

As shown in Fig. 2, when the model ends its training, the model accuracy shows no significant difference under the different value of d. Even at minimal values of the upload proportion $d \times 100\% = 7\%$, the model accuracy drops only a little. Note that when $d \times 100\% = 100\%$, the training process is equal to *FedAvg*. We can also observe that higher the d, more the training efficiency. Table IV records the training rounds before the global model converges to the same accuracy. When d decreases to 0.15, the training rounds on each dataset only increase by 1.22–1.6×.

TABLE V
ATTACK SUCCESS RATE UNDER THE SINGLE-SHOT ATTACK

Dataset	Method	60%	70%	80%
MNIST	FedAvg	0.939	0.954	0.954
	PartFedAvg	**0.24**	**0.056**	**0.031**
CIFAR-10	FedAvg	0.930	0.954	0.954
	PartFedAvg	**0.038**	**0.040**	**0.066**
LOAN	FedAvg	0.995	0.995	0.992
	PartFedAvg	**0**	**0**	**0**

C. Evaluation on Backdoor Attacks

We assume the malicious client trains its local backdoor model by the label-flipping method [22]. Specifically, for each batch, we set 64 clean samples combined with 20 backdoor samples, ten backdoor samples, and five backdoor samples for LOAN, MNIST, and CIFAR-10, respectively. Each client submits $d \times 100\% = 10\%$ of its model per round.

1) Backdoor Attack Settings: We evaluate the performance of *PartFedAvg* against the backdoor attacks from three aspects, single-shot attack [11] (i.e., model replacement), continuous attack, and distributed backdoor attack (DBA) attack [12]. We test the effect of the single-shot attack when the global model accuracy stabilizes to 60%, 70%, and 80%, respectively. At the attack round, since the expected value of $\frac{\eta}{z^t[j]}$ is 0.1, we scale up the partial update of the malicious client by 10 times. For the continuous attack, we follow the *naive approach* [11]. The malicious client is selected for 80, 100, and 200 consecutive rounds. In every $\frac{1}{d}$ rounds of consecutive attacks, the malicious client uploads all the parameters of update in index order. For example, at the first round, the malicious client selects $v[1], v[2], \ldots, v[d * l]$. At the second round, the malicious client selects $v[d * l + 1], v[d * l + 2], \ldots, v[2 * d * l]$. The DBA attack means multiple malicious clients cooperatively perform the backdoor attack [12]. We set four distributed attackers and assume each attacker owns a dataset with a distributed trigger to train a backdoor model. Similar to a single-shot attack, we scale up the model update uploaded by each attacker 10 times. The accuracy of the attack against the global model is calculated on a dataset with the backdoor trigger.

Results. As given in Table V, the *PartFedAvg* can significantly reduce the impact of the single-shot attack on the global model

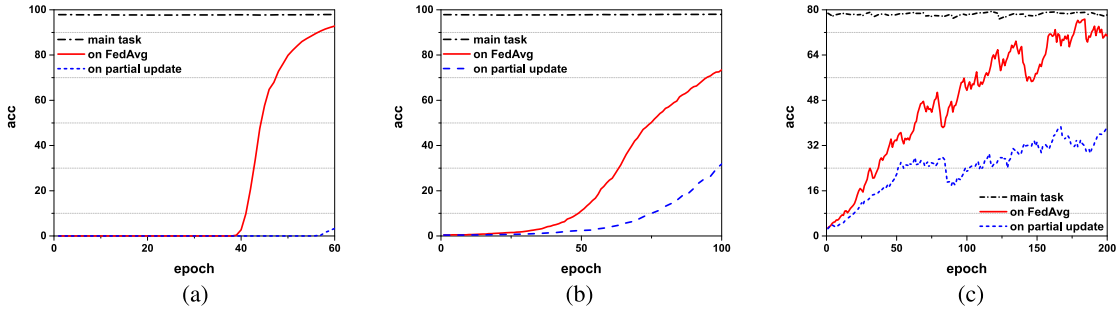


Fig. 3. Attack success rate of the continuous attack when using *FedAvg* and *PartFedAvg* on three datasets. (a) LOAN. (b) MNIST. (c) CIFAR.

TABLE VI
ATTACK SUCCESS RATE UNDER THE DBA ATTACK

Method	LOAN	MNIST	CIFAR-10
FedAvg	99.99	100	97.29
PartFedAvg	**0**	**0.88**	**8.67**

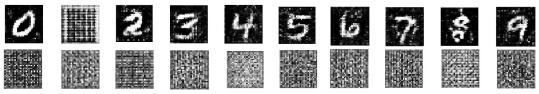


Fig. 4. Results of the inference attack on MNIST under two schemes, the first row represents *FedAvg* and the second row represents *PartFedAvg*.

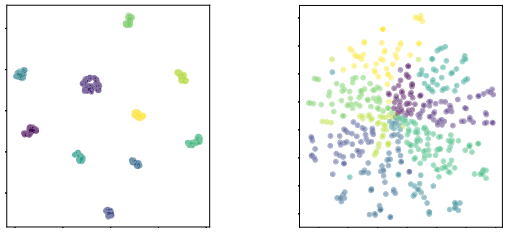


Fig. 5. Using t-distributed stochastic neighbor embedding (t-sne) to cluster the complete model updates and partial model updates under non-i.i.d. data setting. (a) Complete updates. (b) Partial updates.

without additional detections. Especially on the LOAN dataset, the attack success rate dropped from 99% to 0%. Similarly, a more powerful attack scenario which is almost impossible in reality, the continuous attack, is also significantly reduced and slowed down, as shown in Fig. 3. As for the DBA attack, the results in Table VI show that the *PartFedAvg* can mitigate the powerful distributed backdoor attack significantly. It is not surprising to get this conclusion, as the *PartFedAvg* can effectively reduce the impact of each malicious client.

D. Evaluations on Inference Attacks

The inference attack means that the malicious clients train a model with generative adversarial network (GAN) [23], which takes the update as input and outputs the original data. In our experiment, the inference attack is only applied to the MNIST dataset. Ten clients are involved into the training process, each of whom owns 5000 images, but all from one class. With the *PartFedAvg*, each client submits 10% of its parameters each round. The attacker takes each one's update into the GAN to inverse the training data. As shown in Fig. 4, the *PartFedAvg* can defeat the attacker with the strong attack assumption, while the *FedAvg* has leaked the data information to the attacker.

To explain the failure, we visualize the client's update distribution when each client is assigned only one class of data from the MNIST. As shown in Fig. 5(a), the complete model updates based on the *FedAvg* are clustered well. However, when we select 10% parameters of each model update randomly, and visualize these parts of updates. It's obvious that the incomplete updates calculated on different data distributions cannot be clustered clearly, as shown in Fig. 5(b). It proves that the partial model updates hardly reveal the distribution information of training data.

V. DETAILED CONSTRUCTION OF SECURE PARTIAL AGGREGATION PROTOCOL

In this section, we are concerned about how to design and implement the secure partial aggregation based on the *PartFedAvg* algorithm. As in the secure aggregation protocol [14], we assume that the semihonest server follows the protocol but is curious about clients' data. Our protocol promises that: 1) the individual model update of each client is invisible to others, including the semihonest server and 2) a dishonest upload of model update will be detected by the server with a reasonable probability. We note that the abovementioned detection probability serves as a tradeoff between efficiency and correctness. Such a probability is configurable according to the real-world applications.

A. Building Block

Our secure partial aggregation protocol relies on the distributed version of homomorphic encryption (Definition 1), which has concretely efficient implementations [24], [25] and is a widely used primitive in secure computation [26]–[29]. To adopt homomorphic encryption, we use fixed-point encoding to embed a decimal into a field element.

Definition 1 (Distributed homomorphic encryption): Given a number n of participating parties, an embedding field $\mathbb{F}_p$ and a dimension l of vectors, a distributed version of homomorphic encryption is defined as $\mathsf{HE} = (\mathsf{HE.DKeyGen}, \mathsf{HE.Enc}, \mathsf{HE.DDec}, \mathsf{HE.Add}, \mathsf{HE.Mult})$ where

1) $(\mathsf{pk}, \{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}) \leftarrow \mathsf{HE.DKeyGen}(1^\kappa)$: This is a subprotocol that is collaboratively executed by n parties.

It takes the public security parameter κ as input, generates a key pair (pk and sk), and divides sk into secret shares $\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}$. The ith party receives $(\mathsf{pk}, \mathsf{sk}_i)$ at the end of protocol execution.

2) $c \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, \mathbf{v})$: This is an offline algorithm that takes the public key pk and a plaintext vector $\mathbf{v} \in \mathbb{F}_p^l$ as inputs, and outputs its ciphertext c.
3) $\mathbf{v} \leftarrow \mathsf{HE.DDec}(\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}, c)$: This is a subprotocol that is collaboratively executed by n parties. It takes all secret shares of the secret key sk and a ciphertext c as input, and outputs its plaintext vector $\mathbf{v} \in \mathbb{F}_p^l$ to a designated party.
4) $c_{\mathsf{add}} \leftarrow \mathsf{HE.Add}(c_1, \ldots, c_k, \alpha_1, \ldots, \alpha_k)$: This is an offline algorithm that takes k well-formed ciphertexts such that $c_i \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, \mathbf{v}_i)$, and k combination coefficients such that $\alpha_i \in \mathbb{F}_p$ as inputs. It outputs a ciphertext c_{add} such that $\mathsf{HE.DDec}(\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}, c_{\mathsf{add}}) = \sum_{i=1}^{k} \alpha_i \mathbf{v}_i$.
5) $c_{\mathsf{mult}} \leftarrow \mathsf{HE.Mult}(c_1, \ldots, c_k)$: This is an offline algorithm that takes as input k well-formed ciphertexts such that $c_i \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, \mathbf{v}_i)$. It outputs a ciphertext c_{mult} such that $\mathsf{HE.DDec}(\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}, c_{\mathsf{mult}}) = (\prod_{i=1}^{k} \mathbf{v}_i[1], \ldots, \prod_{i=1}^{k} \mathbf{v}_i[l])$.

B. Protocol

Our secure partial aggregation protocol is presented in Algorithm 2. The high-level idea is that, to upload a partial update, a client is required to upload a homomorphic ciphertext of its *full update*, and k homomorphic ciphertexts of indicator vectors. Each indicator vector contains exactly $d \times l$ elements and other $(1-d) \times l$ entries are set to zero. To verify that the ith client does not attempt to upload fewer or more model parameters, the server instructs all selected clients to decrypt k_0-out-of-k encrypted indicator vectors submitted by this client. If there is at least one indicator vector that contains fewer or more ones, the server can immediately find that the ith client behaves dishonestly. This detection method guarantees that a dishonest client will be caught with a certain probability. Moreover, it does not reveal the clear model updates to others since these indicator vectors are independent of model updates.

Meanwhile, to facilitate partial aggregation, the sender first choose one encrypted indicator vector from the remaining $k - k_0$ ones, and multiplies it with the encrypted full update by $\mathsf{HE.Mult}$. This procedure aims to obliviously extract $d \times l$ parameters from the ith party's full update. Then, the server executes $\mathsf{HE.Add}$ to 1) add up all extracted updates and 2) add up all corresponding encrypted indicator vectors. Finally, the server instructs all selected clients to decrypt the two ciphertexts and computes the clear partial aggregation result. To avoid the division by zero, we introduce a small smoothing parameter $\lambda > 0$ to the denominator.

1) Optimization—Batch Verification: The k_0 calls of $\mathsf{HE.DDec}$ sub-protocol for each selected client is a performance bottleneck of the overall protocol. We observe that this cost can be reduced by verifying a random linear combination of the k_0 encrypted indicator vectors once, rather than verifying each encrypted indicator vector. Let $\|\mathbf{v}\| := \sum_{j=1}^{d} \mathbf{v}[j] \mod p$ for a vector $\mathbf{v} \in \mathbb{F}_p^d$, and $c_{i,j} \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, \mathbf{I}_{i,j})$ be the ciphertext of the jth indicator vector $\mathbf{I}_{i,j}$ for the ith client. To enable a batch verification of $\{c_{i,1}, \ldots, c_{i,k_0}\}$, the server uniformly draws $\alpha_1, \ldots, \alpha_{k_0} \leftarrow \mathbb{F}_p$ and computes

$$c_i = \mathsf{HE.Add}(c_{i,1}, \ldots, c_{i,k_0}, \alpha_1, \ldots, \alpha_{k_0}).$$

By the definition of homomorphic encryption, the server can instead verify that

$$\begin{aligned} &\|\mathsf{HE.DDec}(\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}, c_i)\| \\ &= \left\| \sum_{j=1}^{k_0} \alpha_j \mathbf{I}_{i,j} \right\| = \sum_{j=1}^{k_0} \|\alpha_j \mathbf{I}_{i,j}\| \mod p \\ &= d \cdot l \cdot \sum_{j=1}^{k_0} \alpha_j \mod p. \end{aligned}$$

This batch verification reduces the total number of $\mathsf{HE.DDec}$ calls to m, the number of selected clients in each epoch.

If we only care about whether there is a cheating client or not (instead of knowing the specific cheating client), the number of times the $\mathsf{HE.DDec}$ called can be further reduced to 1. This is achieved by having the server to sample another set of random coefficients: $\beta_1, \ldots, \beta_m \leftarrow \mathbb{F}_p$. More specifically, the server uses these coefficients to run a single batch verification with respect to $c_1, \ldots, c_m$ that are defined previously. Letting

$$c = \mathsf{HE.Add}(c_1, \ldots, c_m, \beta_1, \ldots, \beta_m)$$

it is supposed to hold that

$$\begin{aligned} &\|\mathsf{HE.DDec}(\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}, c)\| \\ &= \left\| \sum_{i=1}^{m} \beta_i \sum_{j=1}^{k_0} \alpha_j \mathbf{I}_{i,j} \right\| = \sum_{i=1}^{m} \sum_{j=1}^{k_0} \|\beta_i \alpha_j \mathbf{I}_{i,j}\| \mod p \\ &= d \cdot l \cdot \sum_{i=1}^{m} \sum_{j=1}^{k_0} \beta_i \alpha_j \mod p. \end{aligned}$$

C. Security Analysis

It follows immediately from the IND-CPA security of the homomorphic encryption that our secure partial aggregation protocol is secure against the semihonest server. It remains to be seen that a cheating client will be caught by the server with a reasonable probability. A dishonest client can successfully cheat in the following two cases.

1) Case_1: All malformed encrypted indicator vectors are not chosen by the server.
2) Case_2: The malformed encrypted indicator vectors happens to meet the verification equation.

For Case_1, it is clear that the number of malformed indicator vector should be not greater than $k - k_0$; otherwise, the dishonest behavior will be trivially detected. Given s malformed ciphertexts, the probability that a dishonest client bypasses the detection is $\binom{k-s}{k_0} / \binom{k}{k_0}$. Hence, the probability of successful detection

Algorithm 2: Secure Partial Aggregation.

Server:
Initialize the global model G^0
for $t = 1$ to T **do**
  Select a subset C_t from n clients where $|C_t| = m$.
  All clients in C_t collaboratively run $\mathsf{HE.DKeyGen}(1^\kappa)$ where the ith client obtains $(\mathsf{pk}, \mathsf{sk}_i)$.
  for $i = 1$ to m **do**
    $([\![\boldsymbol{\Delta}_i^t]\!], [\![\mathbf{I}_{i,1}]\!], \ldots, [\![\mathbf{I}_{i,k}]\!]) \leftarrow \mathbf{ClientUpdate}(G^{t-1}, D_i)$
    Sample a k_0-sized subset S_i of $\{[\![\mathbf{I}_{i,1}]\!], \ldots, [\![\mathbf{I}_{i,k}]\!]\}$
    Sample random coefficients: $\beta_i, \alpha_{i,1}, \ldots, \alpha_{i,k_0} \leftarrow \mathbb{F}_p$
  end for
  Run batch verification using $\{S_i, \beta_i, \alpha_{i,1}, \ldots, \alpha_{i,k_0}\}_{1 \le i \le m}$ and abort if verification fails
  for $i = 1$ to m **do**
    Sample an encrypted indicator vector $[\![z_i]\!]$ from $\{[\![\mathbf{I}_{i,1}]\!], \ldots, [\![\mathbf{I}_{i,k}]\!]\} \setminus S_i$
    $[\![\boldsymbol{\delta}_i^t]\!] \leftarrow \mathsf{HE.Mult}([\![\boldsymbol{\Delta}_i^t]\!], [\![z_i]\!])$
  end for
  $[\![\boldsymbol{\delta}^t]\!] \leftarrow \mathsf{HE.Add}([\![\boldsymbol{\delta}_1^t]\!], \ldots, [\![\boldsymbol{\delta}_m^t]\!], 1, \ldots, 1)$
  $[\![z]\!] \leftarrow \mathsf{HE.Add}([\![z_1]\!], \ldots, [\![z_m]\!], 1, \ldots, 1)$
  Receive $\boldsymbol{\delta}^t$ and z by instruct all clients in C_t to collaboratively run $\mathsf{HE.DDec}(\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}, [\![\boldsymbol{\delta}^t]\!])$ and $\mathsf{HE.DDec}(\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}, [\![z]\!])$
  Compute the *smoothed* weighted average $\mathbf{w} \leftarrow (\frac{\boldsymbol{\delta}^t[1]}{z[1]+\lambda}, \ldots, \frac{\boldsymbol{\delta}^t[l]}{z[l]+\lambda})$ for some smoothing parameter $\lambda > 0$
  $G^t = G^{t-1} + \mathbf{w}$
end for

$\mathbf{ClientUpdate}(G^{t-1}, D_i)$:
$L_i^t \leftarrow L_i^{t-1}$
for each local epoch **do**
  $L_i^t \leftarrow L_i^t - \alpha \nabla L(G^{t-1}, D_i)$
end for
$\Delta_i^t \leftarrow L_i^t - L_i^{t-1}$
$[\![\Delta_i^t]\!] \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, \Delta_i^t)$ // Implicit fixed-point encoding
for $j = 1$ to k **do**
  Sample a l-sized binary vector $\mathbf{I}_{i,j}$ with exactly $d \cdot l$ ones
  $[\![\mathbf{I}_{i,j}]\!] \leftarrow \mathsf{HE.Enc}(\mathsf{pk}, \mathbf{I}_{i,j})$
end for
Return $([\![\boldsymbol{\Delta}_i^t]\!], [\![\mathbf{I}_{i,1}]\!], \ldots, [\![\mathbf{I}_{i,k}]\!])$ to the server

with respect to Case_1 is at least $\min_{1 \le s \le k-k_0}(1 - \binom{k-s}{k_0}/\binom{k}{k_0})$. For Case_2, a cheating client should guess exactly one random coefficient $\beta_i \cdot \alpha_j$. This happens with a negligible probability $1/|\mathbb{F}_p| = 1/p$ if the field size p is exponential in the security parameter κ (e.g., $\kappa = 128$). That is, the probability of successful detection with respect to Case_2 is $1 - 1/p \approx 1$. To sum up, a cheating client will be detected by the server with a probability at least $\min_{1 \le s \le k-k_0}(1 - \binom{k-s}{k_0}/\binom{k}{k_0})$.

VI. Conclusion

In this article, we proposed the practical and efficient *PartFedAvg* algorithm to replace *FedAvg* as the basic training protocol, to improve the security and robustness of the federated learning. We proved that the *PartFedAvg* protocol can protect the model correctness from threats of backdoor attacks and protect the data privacy from threats of inference attacks simultaneously. Combined with cryptography and verification mechanisms, we also designed a practical secure *PartFedAvg* to achieve better security and robustness. We believed that our protocol can enable edge devices and servers to work together in federated learning in the industrial big data scenario. As part of the future work, the cost of encryption and decryption will be evaluated. In addition, we will use our protocol to defend against seemingly more powerful backdoor attack (e.g., the attacks that only rely on partial parameters). We believe that our work is helpful to break data silos under the premise of security and privacy preservation and will promote the development of Industry 4.0.

Acknowledgment

Jiqiang Gao, Baolei Zhang, and Xiaojie Guo are supervised by Zheli Liu.

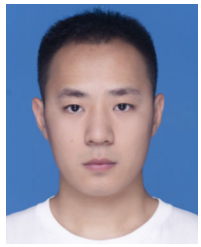
References

[1] C. Wang, G. Yang, G. Papanastasiou, H. Zhang, J. J. P. C. Rodrigues, and V. H. C. de Albuquerque, "Industrial cyber-physical systems-based cloud IoT edge for federated heterogeneous distillation," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5511–5521, Aug. 2021.
[2] M. Aazam, S. Zeadally, and K. A. Harras, "Deploying fog computing in industrial Internet of Things and industry 4.0," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4674–4682, Oct. 2018.
[3] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Informat.*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.
[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.
[5] S. Agrawal *et al.*, "Federated learning for intrusion detection system: Concepts, challenges and future directions," 2021, *arXiv:2106.*09527. [Online]. Available: https://arxiv.org/abs/2106.09527
[6] S. Agrawal, S. Sarkar, M. Alazab, P. K. R. Maddikunta, T. R. Gadekallu, and Q. Pham, "Genetic CFL: Optimization of hyper-parameters in clustered federated learning," *CoRR*, vol. abs/2107.07233, 2021. [Online]. Available: https://arxiv.org/abs/2107.07233
[7] Y. Zhou, Q. Ye, and J. Lv, "Communication-efficient federated learning with compensated overlap-fedavg," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 1, pp. 192–205, Jan. 2022.
[8] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 691–706.
[9] B. Zhao, K. R. Mopuri, and H. Bilen, "iDLG: Improved deep leakage from gradients," 2020, *arXiv:2001.02610.*
[10] L. Zhu and S. Han, "Deep leakage from gradients," in *Federated Learning*. Berlin, Germany: Springer, 2020, pp. 17–31.
[11] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," 2018, *arXiv:1807.00459.*
[12] C. Xie, K. Huang, P. Chen, and B. Li, "DBA: Distributed backdoor attacks against federated learning," in *Proc. Int. Conf. Learn. Representations.*, 2020, pp. 1–19.
[13] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5650–5659.

[14] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur., CCS*, 2017, pp. 1175–1191.
[15] X. Guo *et al.*, "VeriFL: Communication-efficient and fast verifiable aggregation for federated learning," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, no. 1, pp. 1736–1751, Jan. 2021.
[16] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–19.
[17] B. Zhao, K. Fan, K. Yang, Z. Wang, H. Li, and Y. Yang, "Anonymous and privacy-preserving federated learning with industrial big data," *IEEE Trans. Ind. Informat.*, vol. 17, no. 9, pp. 6314–6323, Sep. 2021.
[18] P. Blanchard *et al.*, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. Conf. Neural Inf. Process. Syst.*, 2017, pp. 119–129.
[19] S. U. Stich, "Local SGD converges fast and communicates little," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–17.
[20] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of FedAvg on non-IID data," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–26.
[21] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*.
[22] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47230–47244, 2019.
[23] B. G. H. Ateniese and F. Pérez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur., CCS*, 2017, pp. 603–618.
[24] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proc. Annu. Cryptol. Conf.*, 2012, pp. 643–662.
[25] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," IACR Cryptol., Lyon, France, Tech. Rep. 2012/144, 2012.
[26] M. Kraitsberg, Y. Lindell, V. Osheter, N. P. Smart, and Y. T. Alaoui, "Adding distributed decryption and key generation to a ring-LWE based CCA encryption scheme," in *Proc. Australas. Conf. Inf. Secur. Privacy*, 2019, pp. 192–210.
[27] C. Baum, D. Cozzo, and N. P. Smart, "Using topgear in overdrive: A more efficient ZKPoK for SPDZ," in *Proc. Int. Conf. Sel. Areas Cryptogr.*, 2019, pp. 274–302.
[28] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ great again," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2018, pp. 158–189.
[29] E. Orsini, N. P. Smart, and F. Vercauteren, "Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption," in *Proc. Cryptographers' Track RSA Conf.*, 2020, pp. 254–283.

Jiqiang Gao was born in Henan, China, in 1996. He received the B.S. degree in information security and law in 2019 from Nankai University, Tianjin, China, where he is currently working toward the master's degree in computer science and technology with the College of Cyber Science.

His research interests mainly include protocol security about machine learning algorithm, distributed machine learning, adversarial machine learning, and attack and defense on different machine learning algorithms.

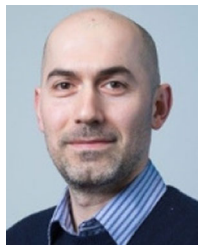


Baolei Zhang was born in Henan, China, in 1998. He received the B.S. degree in software engineering from Henan University, Kaifeng, China, in 2020. He is currently working toward the M.S. degree in computer technology with Nankai University, Tianjin, China.

He is currently focusing on the aggregation protocol in federated learning, which can simultaneously protect the data privacy of the client and guarantee the robustness of the global model. His research interests include privacy disclosure and protection in multiparty machine learning and centralized machine learning, attack and protection in machine learning, and multiparty secure computing.

Xiaojie Guo received the B.Eng. degree in information security and law in 2018 from Nankai University, Tianjin, China, in 2018, where he is currently working toward the Ph.D. degree in computer science and technology with the College of Cyber Science.

His research interests include cryptography, multiparty computation, and privacy-preserving machine learning.



Thar Baker (Senior Member, IEEE) received the Ph.D. degree in autonomic cloud applications from Liverpool John Moores University (LJMU), Liverpool, U.K., in 2010.

He became a Senior Fellow of Higher Education Academy in 2018. He is currently an Associate Professor with the Department of Computer Science, College of Computing and Informatics, University of Sharjah (UoS), Sharjah, UAE. Before joining UoS, he was a Reader of Cloud Engineering and the Head of the Applied Computing Research Group, Faculty of Engineering and Technology, LJMU. Prior to that, he was a Lecturer of Computer Science with the Department of Computing and Mathematics, Manchester Metropolitan University, Manchester, U.K. He has authored or coauthored numerous refereed research papers in multidisciplinary research areas, including parallel and distributed computing, algorithm design, green and sustainable computing, and energy routing protocols.

Min Li received the B.E. degree in automatic control, the M.E. degree in control theory and control engineering, and the Ph.D. degree in information security from Nankai University, Tianjin, China, in 1998, 2005, and 2012, respectively.

She is currently a Lecturer with the College of Cyber Science, Nankai University. Her current research interests include cryptography, differential privacy, and federated learning.

Zheli Liu received the B.Sc. and M.Sc. degrees in computer science and the Ph.D. degree in computer application from Jilin University, Changchun, China, in 2002, 2005, and 2009, respectively.

After a Postdoctoral Fellowship with Nankai University, Tianjin, China, he joined the College of Computer and Control Engineering, Nankai University, in 2011. He is currently a Professor with Nankai University. His current research interests include applied cryptography and data privacy protection.