This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

VerifyNet: Secure and Verifiable Federated Learning

Guowen Xu, *Student Member, IEEE*, Hongwei Li (Corresponding author), *Senior Member, IEEE*, Sen Liu, *Student Member, IEEE*, Kan Yang, *Member, IEEE*, Xiaodong Lin, *Fellow, IEEE*

***Abstract*—As an emerging training model with neural networks, federated learning has received widespread attention due to its ability of updating parameters without collecting users' raw data. However, since adversaries can track and derive participants' privacy from the shared gradients, federated learning is still exposed to various security and privacy threats. In this paper, we consider two major issues in the training process over Deep Neural Networks (DNNs): (1) How to protect user's privacy (i.e., local gradients) in the training process. (2) How to verify the integrity (or correctness) of the aggregated results returned from the server. To solve the above problems, several approaches focusing on secure or privacy-preserving federated learning have been proposed and applied in diverse scenarios. However, it is still an open problem enabling clients to verify whether the cloud server is operating correctly, while guaranteeing user's privacy in training process. In this paper, we propose VerifyNet, the first privacy-preserving and verifiable federated learning framework. In specific, we first propose a double-masking protocol to guarantee the confidentiality of users' local gradients during the federated learning. Then, the cloud server is required to provide the *Proof* about the correctness of its aggregated results to each user. We claim that it is impossible that an adversary can deceive users by forging *Proof*, unless it can solve the NP-hard problem adopted in our model. In addition, VerifyNet is also supportive for users dropping out during the training process. Extensive experiments conducted on real-world data also demonstrate the practical performance of our proposed scheme.**

***Index Terms*—Privacy-preserving, Deep Learning, Verifiable Federated Learning, Cloud Computing.**

I. Introduction

Deep learning has played a significant part in many applications, e.g., medical prediction [**?**], [**?**], autopilot [**?**], [**?**], etc. Such deep learning based applications have penetrated into every aspect of our society and gradually changed the habits of human beings in various areas like living, travel, and socializing [**?**], [**?**].

Deep learning requires a large number of data which are usually collected from users. However, user's data may be sensitive in nature or contain some private information. For example, in healthcare systems, the patients may not be willing to share their medical data with a third party service provider (e.g., cloud server) [**?**], [**?**], [**?**]. Recently, federated learning [**?**], [**?**] is gradually gaining attention from both academia and industry with its ability of training network without collecting users' original data, where all users and the cloud server work together only by sharing local gradients and global parameters. However, research shows that attackers can still indirectly obtain the sensitive information including tabs [**?**], [**?**] and memberships [**?**], [**?**] based on the shared gradients. On the other hand, data integrity breaches in federated learning have also been frequently reported in the media [**?**], [**?**]. Particularly, driven by certain illegal interests, a malicious cloud provider may return incorrect results to users. For example, a "lazy" cloud provider may compress the original model with a simpler but less accurate model to reduce its own computation cost, or worse, maliciously forge the aggregated results sent to users. Therefore, protecting user's privacy and data integrity (especially the correctness of results returned from the server) are two fundamental issues in the training process of federated learning. Hence, it is urgent and meaningful to design a secure federated training protocol, which can efficiently verify the correctness of results returned from the server while protecting user's data privacy.

In order to solve the above problems, several works focusing on privacy-preserving deep learning have been proposed. Shokri et al. [**?**] proposed a privacy-preserving deep learning protocol by selectively sharing updated parameters, which can achieve a balance between practicality and security. Trieu Phong et al. [**?**] proposed a secure deep learning system through the integration of additively homomorphic encryption and gradient descent technology. Recently, Keith Bonawitz et al. [**?**] put forward a practical and secure architecture for federated learning by exploiting the secret sharing and key agreement protocol, which allows users to be offline during the execution while still guaranteeing high accuracy. However, none of the above solutions support verifying the *correctness* of results returned from the server. It is closely related between the correctness of results returned from the server and the privacy of users' local gradients. The risk of users' privacy being compromised always tends to increase once the adversary has been able to manipulate the data returned to users. For instance, in the well-known white-box attacks [**?**], [**?**], adversaries can carefully return crafted results to users for analyzing statistical characteristics of user-uploaded data, and induce users to release more additional sensitive information.

Recently, several schemes [**?**], [**?**] have been successively proposed to alleviate data integrity problem under the well-trained neural network. However, these schemes either support a small variety of activation functions or require additional

Guowen Xu is with the school of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China, and also with the CETC Big Data Research Institute Co., Ltd., 550022 Guiyang, China (e-mail: guowen.xu@foxmail.com)

Hongwei Li is with the school of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China, and also with the Science and Technology on Communication Security Laboratory, Chengdu 610041, China (e-mail: hongweili@uestc.edu.cn)

Sen Liu is with the school of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: 893551724@qq.com)

Kan Yang is with the department of Computer Science, University of Memphis, TN 38152 USA (e-mail: Kan.Yang@memphis.edu)

Xiaodong Lin is with the School of Computer Science, University of Guelph, 50 Stone Rd E, Guelph, ON N1G 2W1, Canada (e-mail: xlin08@uoguelph.ca)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

hardware assistance. To the best of our knowledge, there is no existing solution supporting verifiability for a neural network during the training process. Compared with a well-trained neural network, it is obviously more complicated to verify the correctness of the results during the training process, since we have to update the parameters of the entire network in addition to predicting the results. Besides, it is also a challenge of how to support verifiability while tolerating users dropping out during the workflow (due to unreliable networks, device battery issues, etc.) and ensuring the confidentiality of all users'(including dropout) local gradients.

In this paper, we propose VerifyNet, the first privacy-preserving approach supporting verification in the process of training neural networks. We first design a verifiable approach based on the homomorphic hash function and pseudorandom technologies to support the verifiability for each user. Then, we use a variant of secret sharing technology along with key agreement protocol to protect the privacy of users' local gradients, and deal with the users dropping out problem during the training process. In summary, our contributions can be summarized as follows:

- We exploit the homomorphic hash function integrated with pseudorandom technology as the underlying structure of VerifyNet, which allows users to verify the correctness of results returned from the server with acceptable overhead.
- We propose a double-masking protocol to guarantee the confidentiality of users' local gradients during the federated learning. It can endure a certain amount of users exiting for some reasons during training process, and the privacy of these exiting users are still protected.
- We give a comprehensive security analysis for our VerifyNet. We claim that the attackers will not get any useful information of users' local gradients even if the cloud server colludes with multiple users. Besides, extensive experiments conducted on real-world data also demonstrate that our VerifyNet is practical.

The remainder of this paper is organized as follows. In Section II, we outline the problem statement. In Section III and Section IV, we describe the preliminaries and give a technical intuition to explain how we solve the challenges considered in this paper, respectively. In Section V, we describe the technical details of our VerifyNet. Then, we show the security analysis in Section VI. Next, performance evaluation and related works are discussed in VII and VIII, respectively. Finally, Section IX concludes the paper.

II. PROBLEM STATEMENT

In this section, we first review the main concepts of federated deep learning. Then we describe the system architecture, threat model and design goals.

A. Federated Deep Learning

1) Overview: Based on the training style, deep learning can be divided into the following two types.

- *Centralized Training*. As shown in Fig.1(a), traditional centralized training starts with the server asking users to upload their local data (i.e., training samples) to the cloud. Then, the server initializes deep neural networks on the cloud, and trains them with training samples until the optimal parameters are obtained. In the end, the cloud server will release the predictive service interface or return the optimal parameters to the user.
- *Federated Training*. As discussed before, users directly upload local data to the server with potential threats for privacy breaches. Hence, different from the centralized training, in federated training (shown in Fig.1(b)), each user and server collaborate to train a unified neural network model. To speed up the convergence of the model, each user shares local parameters (i.e., gradients) to the cloud server, which aggregates all gradients and returns the results to each user. Ultimately, the server and each user will get the optimal network parameters. Compared with the centralized training, federated training reduces the risk of user's privacy being compromised. However, research shows that attackers can still indirectly obtain the sensitive information based on the shared gradients. In addition, driven by certain illegal interests, a malicious cloud provider may return incorrect results to users. Therefore, in this paper, we focus on protecting the privacy of users' local gradients while verifying the correctness of results returned from the server in the federated training process.

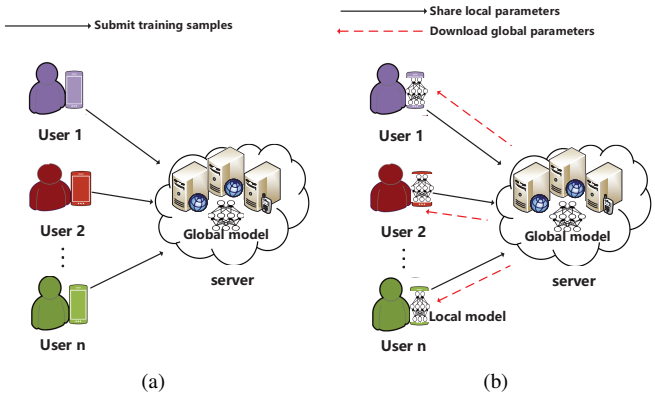


Fig. 1: General framework for centralized training and federated training. (a) Centralized Training. (b) Federated Training.

2) Neural Network: As the underlying structure of deep learning, neural network can be integrated with various technologies to achieve classification, prediction, and regression. As shown in Fig.2, there is a fully connected neural network with 3 inputs, a hidden layer and 2 outputs. The *fully connected* means that all neurons between two adjacent layers are connected by variables (called $\boldsymbol{\omega}$ in this section) to each other. In general, a neural network can be represented as a function $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\omega}) = \hat{\boldsymbol{y}}$, where $\boldsymbol{x}$ denotes the users' inputs, and $\hat{\boldsymbol{y}}$ is the corresponding outputs via function $\boldsymbol{f}$ with parameter $\boldsymbol{\omega}$.

3) Federated Learning Updates: Without loss of generality, assume that each data record is an observation pair $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$, and the entire training set is $D = \{\langle \boldsymbol{x_i}, \boldsymbol{y_i} \rangle, i = 1, 2, \cdots T\}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

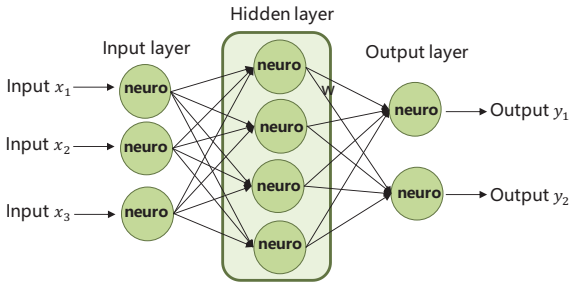


Fig. 2: Fully Connected Neural Network

A loss function can be defined on the training set as

$$\mathcal{L}_f(D, \boldsymbol{\omega}) = \frac{1}{|D|} \sum_{\langle \boldsymbol{x_i}, \boldsymbol{y_i} \rangle \in D} \mathcal{L}_f(\boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{\omega})$$

where $\mathcal{L}_f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\omega}) = l(\boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{\omega}))$ for a specific loss function l. In this paper, loss function is set as $l(\boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{\omega})) = l(\boldsymbol{y}, \hat{\boldsymbol{y}}) = ||\boldsymbol{y}, \hat{\boldsymbol{y}}||_2$, where $|| \cdot ||_2$ is the l_2 norm of a vector.

The goal of training neural network is to find the optimal parameters $\boldsymbol{\omega}$ consequently to minimize the loss function. In our VerifyNet, we adopt stochastic gradient descent [**?**], [**?**] to complete this task. Specifically, each parameter is iteratively calculated as follows.

$$\boldsymbol{\omega}^{j+1} \leftarrow \boldsymbol{\omega}^j - \lambda \nabla \mathcal{L}_f(D^j, \boldsymbol{\omega}^j)$$

where $\boldsymbol{\omega}^j$ indicates the parameters after the j-th iteration. D^j is a random subset of D, and λ is the parameter of learning rate. In our federated learning, each user $n \in \mathcal{N}$ holds a private local data set D_n, and trains local set with a certain neural network agreed with all other participants in advance, where $D = \sum_{n \in \mathcal{N}} D_n$. Concretely, the server selects a random subset $\mathcal{N}^j \subseteq \mathcal{N}$ at the j-th iteration, and then each user $n \in \mathcal{N}^j$ randomly chooses a subset $D_n^j \subseteq D_n$ to execute stochastic gradient descent. Therefore, parameter update can be rewritten as below.

$$\boldsymbol{\omega}^{j+1} \leftarrow \boldsymbol{\omega}^j - \lambda \frac{\sum_{n \in \mathcal{N}^j} \rho_n^j}{\sum_{n \in \mathcal{N}^j} |D_n^j|}$$

where $\rho_n^j = |D_n^j| \nabla \mathcal{L}_f(D_n^j, \boldsymbol{\omega}^j)$ is computed by each user and subsequently shared to the cloud server. Then, the cloud server returns the global parameters $\boldsymbol{\omega}^{j+1}$ to all users.

B. System Architecture

As shown in Fig. 3, our system model consists of three entities, *Trusted Authority (TA)*, *User* and *Cloud Server*.

- *Trusted Authority (TA)*: The main job of TA is to initialize the entire system, generate public parameters, and assign public and private keys to each participant. Afterwards, it will go offline unless a dispute arises.
- *User*: Each user needs to send his/her encrypted local gradients to the cloud server during each iteration. Besides, the cloud server will also receive some other encrypted information to prepare for generating *Proof* of its calculated results.

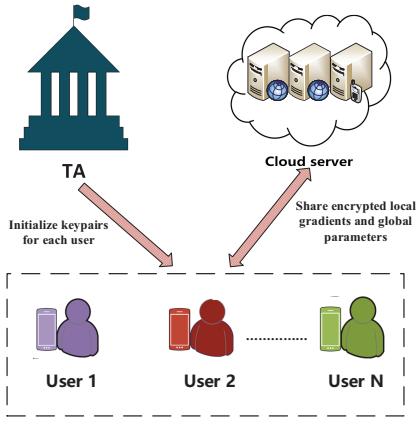


Fig. 3: System Architecture

- *Cloud server*: The cloud server aggregates the gradients uploaded by all online users and sends the results along with the *Proof* to each user, where we require that the cloud server knows nothing but the encrypted gradients and the final results.

C. Threat Model and Design Goal

There we define a threat model called *Honest but Curious Security* [**?**] in our VerifyNet. Specifically, in our VerifyNet, TA is trustworthy and will not collude with any entity. All other participants including the cloud server are considered to be honest-but-curious [**?**], which means that both the cloud server and users will execute the program according to the agreed agreement, but they may also try to infer other users' data privacy independently [**?**], [**?**], [**?**]. In particular, we allow that the cloud server colludes with multiple users to get the most offensive capabilities, and also allow the cloud server to forge *Proof* (There we do not allow collusion to forge *Proof*) and modify the calculated results for deceiving users.

Our VerifyNet aims to protect the confidentiality of users' local gradients while supporting strong verifiability to each user, and tolerate users dropping out during the workflow. We claim that it is impossible to succeed in deception unless the adversary can solve the NP-hard problem adopted in our model.

III. PRELIMINARIES

To facilitate the understanding of the article, we introduce some cryptographic primitives used in our VerifyNet, which will make it easier for readers to understand our approach.

A. Bilinear Pairing

A bilinear pairing can be represented as a map e: $G_1 \times G_2 \rightarrow G_I$, where both G_1 and G_2 are multiplicative cyclic groups with same prime order q. Without loss of generality, we assume that the generator of G_1 and G_2 are g and h, respectively. Informally, a bilinear pairing e has following properties.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

1) **Bilinearity**: Given the random numbers $a, b \in Z_q^*$, for any $g_1 \in G_1$ and $g_2 \in G_2$, we have $e(g_1^a, g_2 b) = e(g_1, g_2)^{ab}$.
2) **Computability**: $e(g_1, g_2)$ can be computed efficiently for any $g_1 \in G_1$ and $g_2 \in G_2$.
3) **Non-degeneracy**: $e(g, h) \neq 1$, where g and h are the generator of G_1 and G_2, respectively.

B. Homomorphic Hash Functions

Informally, given a message $x_i \in Z_q$, a collision-resistant homomorphic hash functions [**?**], [**?**] $HF : Z_q \rightarrow G_1 \times G_2$ can be indicated as follows.

$$HF(x_i) = (A_i, B_i) = (g^{HF_{\delta,\rho}(x_i)}, h^{HF_{\delta,\rho}(x_i)})$$

where both δ and ρ are the secret key randomly selected in finite field Z_q. $HF_{\delta,\rho}()$ is a one-way homomorphic hash function. More precisely, given $HF(x_1) = (g^{HF_{\delta,\rho}(x_1)}, h^{HF_{\delta,\rho}(x_1)})$, $HF(x_2) = (g^{HF_{\delta,\rho}(x_2)}, h^{HF_{\delta,\rho}(x_2)})$, homomorphic hash function has following properties.

1) Additivity (in the exponent) can be indicated as $HF(x_1 + x_2) \leftarrow (g^{HF_{\delta,\rho}(x_1)+HF_{\delta,\rho}(x_2)}, h^{HF_{\delta,\rho}(x_1)+HF_{\delta,\rho}(x_2)})$
2) Multiplying by a constant α can be expressed as $HF(\alpha x_1) \leftarrow (g^{\alpha HF_{\delta,\rho}(x_1)}, h^{\alpha HF_{\delta,\rho}(x_1)})$.

There are other interested properties of homomorphic hash function. Interested readers can refer to [**?**], [**?**] for more details.

C. Pseudorandom Functions

We adopt the pseudorandom functions designed by Dario Fiore et al. [**?**] in our VerifyNet. Informally, given the secret key $K = (K_1, K_2)$, a pseudorandom function $PF_K : \{0,1\}^* \times \{0,1\}^* \rightarrow G_1 \times G_2$ consists of two other pseudorandom functions, i.e., $PF_{K_1} : \{0,1\}^* \rightarrow Z_q^2$ and $PF_{K_2} : \{0,1\}^* \rightarrow Z_q^2$. Given an input (I_1, I_2), we have $PF_{K_1}(I_1) = (\gamma_{I_1}, \nu_{I_1})$ and $PF_{K_2}(I_2) = (\gamma_{I_2}, \nu_{I_2})$. Consequently, we have

$$PF_K(I_1, I_2) = (E, F) = (g^{\gamma_{I_1}\gamma_{I_2}+\nu_{I_1}\nu_{I_2}}, h^{\gamma_{I_1}\gamma_{I_2}+\nu_{I_1}\nu_{I_2}})$$

As part of the verification, the pseudorandom functions will be exploited to verify the correctness of the results from the cloud server.

D. Secret Sharing Protocol

In VerifyNet, we utilize the Shamir's t-out-of-$\mathcal{N}$ secret sharing protocol [**?**] to divide the secret s into $\mathcal{N}$ separate parts, where $\mathcal{N}$ denotes the number of users in our model, and t is the threshold. This means that any subset of shares greater than t can be used to recover the secret s. Specifically, implementing this secret sharing protocol involves following steps.

1) **S.share**$(s, t, \mathcal{U}) \rightarrow \{(n, s_n)\}_{n \in \mathcal{U}}$: Given the threshold $t \leq |\mathcal{U}|$ and the secret s, output the share s_n of s for each user n, where $\mathcal{U}$ represents the set of users' ID (presumed to be distinctive) specified in a finite field $\mathcal{F}$, and $|\mathcal{U}| = \mathcal{N}$.
2) **S.recon**$(\{(n, s_n)\}_{n \in \mathcal{M}}, t) \rightarrow s$: Input a subset $\mathcal{M}$ of shares, where $n \in \mathcal{M} \subseteq \mathcal{U}$ and $t \leq |\mathcal{M}|$, outputs the secret s.

E. Key Agreement

Diffie-Hellman key agreement [**?**], [**?**] is also adopted in our VerifyNet to create the shared key for any two users. Specifically, given a group G with prime order q, the secret/public key of each user n is created as **KA.gen**$(G, g, q) \rightarrow (SK_n, g^{SK_n})$, where g is the generator of group G. SK_n and g^{SK_n} are the secret and public key, respectively. Then, given the public key g^{SK_m} of user m, the shared key between user n and user m can be generated as **KA.agree**$(SK_n, g^{SK_m}) \rightarrow s_{n,m}$. In real-world applications, $s_{n,m}$ is often set to $H((g^{SK_m})^{SK_n})$ for convenience.

IV. Technical Intuition

As discussed above, in federated learning, each user needs to submit its local gradients to the cloud, and then receives the aggregated results (sum of all local gradients) from the server. However, there are three problems that need to be addressed. Firstly, we need to protect the privacy of the user's local gradients, because the adversary can indirectly breach user's sensitive information through these gradient information. Secondly, to prevent malicious spoofing by the server, each user should be able to effectively verify the correctness of the results returned by the server. Thirdly, in real-world scenarios, it is very common for users to be unable to upload data to the server on time due to unreliable networks or device battery issues. Therefore, our proposed protocol should support users offline for some reason in training process. In this section, we give a technical intuition to explain how we solve these three challenges.

A. Single Masking to Protect User's Gradients

Assume that each user n holds a local gradient $x_n, (n \in \mathcal{U}, |\mathcal{U}| = \mathcal{N})$, we originally intend to design a single masking protocol to protect the privacy of user's gradients. Specifically, suppose that all users' ID in our system are ordered, and any two users n and m agree on a random number $r_{n,m}$. Then, we can encrypt each user n's local gradient x_n as follows.

$$\hat{x_n} = x_n + \sum_{m \in \mathcal{U}: n < m} r_{n,m} - \sum_{m \in \mathcal{U}: n > m} r_{n,m} \tag{1}$$

Hence, after each user submitting its encrypted gradient $\hat{x_n}$ to the server, it can calculate the aggregated gradients $\sum_{n \in \mathcal{U}} x_n$ as below.

$$\mathbf{z} = \sum_{n \in \mathcal{U}} \hat{x_n} = \sum_{n \in \mathcal{U}} x_n \tag{2}$$

However, this approach has three drawbacks. Firstly, every user needs to negotiate a random number $r_{n,m}$ with all other users, which will result in quadratic communication overhead ($O(\mathcal{U}^2)$). Secondly, this protocol is failure to support users offline during the training process. We note that even if only one user does not upload data on time, the above aggregation operation cannot be successfully completed because the random number added in this user's gradient cannot be cancelled. Thirdly, verifiability is not supported by above protocol. It is closely related between the correctness of results returned from the server and the privacy of users local gradients. The risk of users'privacy being compromised always tends to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

increase once the adversary has been able to manipulate the data integrity. Therefore, a secure protocol should also support verifiability of results returned by the server.

B. Double-Masking Potocol Supporting Verifiability

We propose a double-masking protocol to address the problems existing in single masking protocol. We first exploit the pseudorandom generator [**?**] and Diffie-Hellman key agreement [**?**], [**?**] to generate the random number $r_{n,m}$ between two users n and m. Concretely, we first ask TA to randomly create key pairs (N_n^{PK}, N_n^{SK}) for each user n. Then, we require the cloud server to broadcast all public key $N_n^{PK}, n \in \mathcal{U}$ to all users. In the end, by exploiting pseudorandom generator and Diffie-Hellman key agreement, each two user n and m can generate the agreed random number denoted as $s_{n,m} \leftarrow$ **KA.agree**(N_n^{SK}, N_m^{PK}). Hence, each user's local gradient x_n can be encrypted as below.

$$\hat{x_n} = x_n + \sum_{m \in \mathcal{U}: n<m} \mathbf{PRG}(s_{n,m}) - \sum_{m \in \mathcal{U}: n>m} \mathbf{PRG}(s_{n,m}) \quad (3)$$

where $\mathbf{PRG}(s_{n,m})$ is a pseudorandom generator with seed $s_{n,m}$.

Next, we adopt the threshold secret sharing scheme [**?**] to support users offline during the training process. In briefly, to offset the random numbers added in the gradients of dropped out users, each user n shares its secret key N_n^{SK} to all other users in advance by utilizing the threshold secret sharing scheme. Hence, if a user n cannot submit its data $\hat{x_n}$ to the cloud on time, the server can decrypt the random numbers (i.e., $\mathbf{PRG}(s_{n,m})$) added in all other users' $\hat{x_m}(m \neq n \in \mathcal{U})$ by asking more than threshold users to submit user n's secret shares. In this way, the random numbers $\mathbf{PRG}(s_{n,m})$ can be recovered and be eventually removed from the $\hat{x_m}$. However, there is still a problem. At some point, some users may delay uploading data to the cloud, which may cause the server to incorrectly determine that these users are offline, and ask other online users to upload shares of those users for removing random numbers. However, just then, these users successfully uploaded their $\hat{x_n}$ to the cloud. As a result, since the server have sufficient secret shares of these users, it can get x_n by removing all the random numbers $s_{n,m}$. To address this problem, we add a new random noise call β_n in each $\hat{x_n}$. β_n will be also shared to all users by utilizing threshold secret sharing scheme. Hence, each user's local gradient x_n is encrypted as below.

$$\hat{x_n} = x_n + \mathbf{PRG}(\beta_n) + \sum_{m \in \mathcal{U}: n<m} \mathbf{PRG}(s_{n,m}) - \sum_{m \in \mathcal{U}: n>m} \mathbf{PRG}(s_{n,m}) \quad (4)$$

In this way, once decryption is needed to obtain the aggregated results $\sum x_n$, the cloud server can only receive the shares of β_n for all online users, and shares of N_n^{SK} for all dropped out users, since these information are enough for the decryption operation.

The double-masking protocol is mainly designed to protect user's data privacy during training, and support users offline for some reason in training process. However, it lacks consideration in terms of verifiability, i.e., there is no specific verifiable mechanism designed. Therefore, the double-masking protocol is not supportive for verifying the correctness of the aggregated results returned from the server. We want to design a verifiable solution that is highly compatible with our double-masking protocol, and allows each user to easily verify the correctness of results returned from the server without the involvement of trusted third parties. To address this challenge, we exploit the homomorphic hash function integrated with pseudorandom technology as the underlying structure of our verifiable approach, which allows users to verify the correctness of execution performed by the cloud server with acceptable overhead. For the specific verification process, please refer to Section V.

V. PROPOSED SCHEME

In this section, we present the technical details of our VerifyNet. At a high level view, the purpose of VerifyNet is to address three problems existing in federated training process. One is to protect the privacy of the user's local gradients in the workflow. Secondly, to prevent malicious spoofing by the server, our VerifyNet supports each user to effectively verify the correctness of the results returned by the server. Thirdly, VerifyNet is also supportive for users offline during the training process.

Fig.4 shows the detailed description of our VerifyNet, which consists of five rounds to complete above tasks. Specifically, TA first initializes the entire system and generates all the public and private keys needed in our VerifyNet. Then, each user n encrypts its local gradient x_n and submits it to the cloud server. After receiving enough message from all online users, the cloud server aggregates the gradients of all online users and returns the results along with $Proof$ to each user. In the end, every user decides to accept or reject the calculation results by verifying the $Proof$, and returns to the **round 0** to start a new iteration.

As shown in **Round 4**, the specific verification process is as follows.

Proof of correctness:

$$\begin{aligned}
(A,B) &= (\prod_{n=1}^{n=|\mathcal{U}_3|} A_n, \prod_{n=1}^{n=|\mathcal{U}_3|} B_n) \\
&= (g^{\sum_{n\in\mathcal{U}_3} HF_{\delta,\rho}(x_n)}, h^{\sum_{n\in\mathcal{U}_3} HF_{\delta,\rho}(x_n)}) \\
&= (g^{HF_{\delta,\rho}(\sum_{n\in\mathcal{U}_3} x_n)}, h^{HF_{\delta,\rho}(\sum_{n\in\mathcal{U}_3} x_n)}) \\
&= (A', B') \\
e(A,h) &= e(g^{HF_{\delta,\rho}(\sigma)}, h) = e(g, h^{HF_{\delta,\rho}(\sigma)}) \\
&= e(g,B) \\
e(L,h) &= e(g^{\sum_{n\in\mathcal{U}_3} \gamma_n\gamma+\nu_n\nu-HF_{\delta,\rho}(x_n)}, h)^{1/d} \\
&= e(g, h^{\sum_{n\in\mathcal{U}_3} \gamma_n\gamma+\nu_n\nu-HF_{\delta,\rho}(x_n)})^{1/d} \\
&= e(g,Q) \\
e(A,h) &\cdot e(L,h)^d \\
&= e(g^{HF_{\delta,\rho}(\sigma)}, h) \cdot e(g^{\sum_{n\in\mathcal{U}_3} \gamma_n\gamma+\nu_n\nu-HF_{\delta,\rho}(x_n)}, h) \\
&= e(g,h)^{\sum_{n\in\mathcal{U}_3} \gamma_n\gamma+\nu_n\nu} \\
&= \Phi
\end{aligned} \quad (5)$$

If any of the above equations are not valid, reject the aggregated result. Otherwise, accept the result and move to **Round 0**. The user and the cloud server iteratively run **Rounds 0-4** until the entire neural network configuration meets the constraints set in advance.

VI. SECURITY ANALYSIS

In this section, we first briefly describe the correctness of the verification. Then, we analyze how our VerifyNet guarantees the confidentiality of each user's local gradients. Other security indicators are beyond the scope of this paper.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

Implementation process of VerifyNet

- **Round 0** *(Initialization):*
  TA:
  - Generate the public/secret keys as $\{(\delta,\rho),(N_n^{PK},N_n^{SK}),(P_n^{PK},P_n^{SK}),K=(K_1,K_2)\}$ to each user n, $(n\in\mathcal{U},|\mathcal{U}|=\mathcal{N})$, where (δ,ρ) and $K=(K_1,K_2)$ are the secret key used in homomorphic hash functions and pseudorandom functions, respectively. $(N_n^{PK},N_n^{SK}),(P_n^{PK},P_n^{SK})$ will be exploited to encrypt user n's local gradient x_n.

  $User\ n$:
  - Send the public keys (N_n^{PK},P_n^{PK}) to the cloud server through a secure channel.

  Server Side:
  - Receive messages from at least t users (represented as $\mathcal{U}_1\subseteq\mathcal{U}$), where t is the threshold of the Shamir's t-out-of-$\mathcal{N}$ protocol used in our model. Otherwise, abort and start over.
  - Broadcast $\{m,N_m^{PK},P_m^{PK},\tau=sum\}_{m\in\mathcal{U}_1}$ to each user $\in\mathcal{U}_1$, where $\tau=sum$ represents the statistic label to be calculated.
- **Round 1** *(Key Sharing):*
  $User\ n$:
  - Receive the $\{m,N_m^{PK},P_m^{PK},\tau=sum\}_{m\in\mathcal{U}_1}$ from the cloud server. Check whether $|\mathcal{U}_1|\geq t$ and all of the key pairs (N_m^{PK},P_m^{PK}) are distinctive. If not, abort and start over.
  - Select a random number β_n. Generate the shares of β_n as $\{(m,\beta_{n,m})\}_{m\in\mathcal{U}_1}\leftarrow$ **S.share**$(\beta_n,t,\mathcal{U}_1)$, where $\beta_{n,m}$ is the share of user n to user m.
  - Generate the shares of N_n^{PK} as $\{(m,N_{n,m}^{SK})\}_{m\in\mathcal{U}_1}\leftarrow$ **S.share**$(N_n^{SK},t,\mathcal{U}_1)$, where $N_{n,m}^{SK}$ is the share of user n to user m.
  - Calculate $\mathcal{P}_{n,m}\leftarrow$ **AE.enc** $\big($**KA.agree**$(P_n^{SK},P_m^{PK}),n||m||N_{n,m}^{SK}||\beta_{n,m}\big)_{m\in\mathcal{U}_1}$, where **AE.enc**() denotes a symmetric encryption [**?**] with secret key **KA.agree**(P_n^{SK},P_m^{PK}).
  - Send $\{\mathcal{P}_{n,m}\}_{m\in\mathcal{U}_1}$ to the cloud server.

  Server Side:
  - Receive messages from at least t users (represented as $\mathcal{U}_2\subseteq\mathcal{U}_1$). Otherwise, abort and start over.
  - Broadcast $\{\mathcal{P}_{m,n}\}_{m\in\mathcal{U}_2}$ to each user $\in\mathcal{U}_2$.
- **Round 2** *(Masked Input):*
  $User\ n$:
  - Receive the $\{\mathcal{P}_{m,n}\}_{m\in\mathcal{U}_2}$ from the cloud server. Check whether $\mathcal{U}_2\subseteq\mathcal{U}_1$ and $|\mathcal{U}_2|\geq t$. If not, abort and start over.
  - Calculate the shared key with every user $m\in\mathcal{U}_2$ as $s_{n,m}\leftarrow$ **KA.agree**(N_n^{SK},N_m^{PK}).
  - Encrypt the local gradients as $\hat{x_n}=x_n+$ **PRG**$(\beta_n)+\sum_{m\in\mathcal{U}_2:n<m}$**PRG**$(s_{n,m})-\sum_{m\in\mathcal{U}_2:n>m}$**PRG**$(s_{m,n})$.
  - In order to verify the correctness of results returned from the server in the future, some additional information are computed as follows.
    Calculate $HF(x_n)=(A_n,B_n)=(g^{HF_{\delta,\rho}(x_n)},h^{HF_{\delta,\rho}(x_n)})$.
    Calculate $PF_{K_1}(n)=(\gamma_n,\nu_n);PF_{K_2}(\tau)=(\gamma,\nu)$.
    Calculate $PF_K(n,\tau)=(E_n,F_n)=(g^{\gamma_n\gamma+\nu_n\nu},h^{\gamma_n\gamma+\nu_n\nu})$
  - Calculate $L_n=(E_n\cdot A_n^{-1})^{1/d}=(g^{\gamma_n\gamma+\nu_n\nu-HF_{\delta,\rho}(x_n)})^{1/d}$, where d is a selected positive integer.
  - Calculate $Q_n=(F_n\cdot B_n^{-1})^{1/d}=(h^{\gamma_n\gamma+\nu_n\nu-HF_{\delta,\rho}(x_n)})^{1/d}$, where d is a selected positive integer.
  - Send $\sigma_n=(\hat{x_n},A_n,B_n,L_n,Q_n,\Omega_n=1)$ to the cloud server.

  Server Side:
  - Receive messages from at least t users (represented as $\mathcal{U}_3\subseteq\mathcal{U}_2$). Otherwise, abort and start over.
  - Broadcast the list of $\mathcal{U}_3$ to each user $\in\mathcal{U}_2$.
- **Round 3** *(Unmasking):*
  $User\ n$:
  - Check whether $\mathcal{U}_3\subseteq\mathcal{U}_2$ and $|\mathcal{U}_3|\geq t$. If not, abort and start over.
  - Decrypt each $\mathcal{P}_{n,m},m\in\mathcal{U}_2\backslash\{n\}$ as $n||m||N_{n,m}^{SK}||\beta_{n,m}\leftarrow$ **AE.dec** $\big($**KA.agree**$(P_n^{SK},P_m^{PK}),\mathcal{P}_{n,m}\big)$.
  - Send $\{(N_{n,m}^{SK})|m\in\mathcal{U}_2\backslash\mathcal{U}_3\}$ and $\{(\beta_{n,m})|m\in\mathcal{U}_3\}$ to the cloud, where $\mathcal{U}_2\setminus\mathcal{U}_3$ represents those users who have sent data to the server in **Round** 1, but drop out before uploading data to the server in **Round** 2.

  Server Side:
  - Receive messages from at least t users (represented as $\mathcal{U}_4\subseteq\mathcal{U}_3$). Otherwise, abort and start over.
  - Calculate $N_n^{SK}\leftarrow$ **S.recon** $\big(\{N_{n,m}^{SK}\}_{m\in\mathcal{U}_4},t\big)$.
  - Calculate $\beta_n\leftarrow$ **S.recon** $\big(\{\beta_{n,m}\}_{m\in\mathcal{U}_4},t\big)$.
  - Calculate **PRG**$(s_{n,m})\leftarrow$ **PRG** $\big($**KA.agree** $\big(\{N_n^{SK},N_m^{PK}\}_{n\in\mathcal{U}_2\backslash\mathcal{U}_3,m\in\mathcal{U}_3}\big)\big)$.
  - Calculate **PRG**$(\beta_n)_{n\in\mathcal{U}_3}$.
  - Calculate the aggregated gradients for all users $\in\mathcal{U}_3$ as
    $\sum_{n\in\mathcal{U}_3}x_n=\sum_{n\in\mathcal{U}_3}\hat{x}_n-\sum_{n\in\mathcal{U}_3}$**PRG**$(\beta_n)-\sum_{n\in\mathcal{U}_3,m\in\mathcal{U}_2\backslash\mathcal{U}_3:n<m}$**PRG**$(s_{n,m})+\sum_{n\in\mathcal{U}_3,m\in\mathcal{U}_2\backslash\mathcal{U}_3:n>m}$**PRG**$(s_{m,n})$.
  - Calculate the *Proof* $\{A,B,L,Q,\Omega\}$ of aggregated gradients as below.
  - $A=\prod_{n=1}^{n=|\mathcal{U}_3|}A_n;B=\prod_{n=1}^{n=|\mathcal{U}_3|}B_n;L=\prod_{n=1}^{n=|\mathcal{U}_3|}L_n;Q=\prod_{n=1}^{n=|\mathcal{U}_3|}Q_n;\Omega=\prod_{n=1}^{n=|\mathcal{U}_3|}\Omega_n$.
  - Broadcast $C_{result}=\{\sigma=\sum_{n\in\mathcal{U}_3}x_n,A,B,L,Q,\Omega\}$ to each user $\in\mathcal{U}_4$.
- **Round 4** *(Verification):*
  $User\ n$:
  - Known $PF_{K_1}(n)=(\gamma_n,\nu_n)$ and $PF_{K_2}(\tau)=(\gamma,\nu)$, calculates $\varphi=\sum_{n\in\mathcal{U}_3}(\gamma_n\gamma+\nu_n\nu)$ and $\Phi=e(g,h)^{\varphi}$.
  - Verify $(A,B)\stackrel{?}{=}(A',B'),e(A,h)\stackrel{?}{=}e(g,B);e(L,h)\stackrel{?}{=}e(g,Q),\Phi\stackrel{?}{=}e(A,h)\cdot e(L,h)^d$.
  - If any of the above equations are not valid, reject the aggregated result. Otherwise, accept the result and move to **Round 0**.

Fig. 4: Detailed description of the VerifyNet

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

A. Correctness of Verification

As shown in Section.V, after receiving the $\{\sigma, A, B, L, Q, \Omega\}$ from the cloud server, each user first checks whether $\Phi = e(A,h) \cdot e(L,h)^d$. Based on the l-$BDHI$ assumption [**?**], $\Phi = e(A,h) \cdot e(L,h)^d$ holds only when $\sum_{n\in\mathcal{U}_3} \gamma_n\gamma + \nu_n\nu$ contained in L (in the exponent of g). If so, each user can infer $L = \prod_{n=1}^{n=|\mathcal{U}_3|} L_n = g^{\sum_{n\in\mathcal{U}_3}\gamma_n\gamma+\nu_n\nu-HF_{\delta,\rho}(\sigma)}$, and knows that $A = g^{HF_{\delta,\rho}(\sigma)}$. Afterwards, based on the DDH assumption [**?**], each user further checks whether both $e(A,h) = e(g,B)$ and $e(L,h) = e(g,Q)$ hold. If this is true, every user will believe that the cloud server correctly calculated B and Q. Until here, each user has verified that (A,B) is calculated correctly. In the end, if $HF(\sigma) = (A',B') = (A,B)$ holds, every user is convinced that the cloud server did return the correct aggregate result $\sigma = \sum_{n\in\mathcal{U}_3} x_n$.

Here we omit the detailed proof since it can be easily proved by utilizing l- $BDHI$ [**?**] and DDH assumption [**?**].

B. Honest but Curious Security

In this section, we prove that our VerifyNet is secure under the honest but curious setting. In our threat model, the cloud server can collude with any $t-1$ users to get the most offensive capabilities, but they still know nothing about the local gradients of honest users except the aggregated results. As illustrated above, each user's local gradient x_n is encrypted as

$$\hat{x_n} = x_n + \mathbf{PRG}(\beta_n) + \sum_{m\in\mathcal{U}_2:n<m} \mathbf{PRG}(s_{n,m}) - \sum_{m\in\mathcal{U}_2:n>m} \mathbf{PRG}(s_{m,n})$$

Moreover, each x_n is also used in homomorphic hash functions [**?**], [**?**] to generate part of verification information. Since the homomorphic hash functions has been proven to be secure [**?**], here we mainly discuss the level of privacy protection that $\hat{x_n}$ can achieve. Before formally presenting the complete proof process, we introduce some useful symbols that will be used later.

We know that users may drop out at some point of the workfolw. We use $\mathcal{U}_i \subseteq \mathcal{U}$ to denote those users who upload data to the cloud server smoothly at **round** $i-1$. Therefore, we have $\mathcal{U} \supseteq \mathcal{U}_1 \supseteq \mathcal{U}_2 \supseteq \mathcal{U}_3 \supseteq \mathcal{U}_4$. The symbol $\mathcal{U}_i \setminus \mathcal{U}_{i+1}$ is exploited to represent those users who have sent data to the server in **Round** $i-1$, but drop out before uploading data in **Round** i. As illustrated before, assume that each user n holds a local gradient $x_n, (n \in \mathcal{U})$, we adopt $x_{\mathcal{U}'} = \{x_n\}_{n\in\mathcal{U}'}$ to indicate a subset $\mathcal{U}'$ of local gradients, where $\mathcal{U}' \subseteq \mathcal{U}$.

In our VerifyNet, the *view* of a party is defined as its internal state (containing its inputs and randomness) and all the messages received from other parties. It should be noted that a party will immediately stop receiving messages while this party exits the execution at some point.

For simplicity, we use S to represent the cloud server. Given a subset $\mathcal{W} \subseteq \mathcal{U} \cup \{S\}$ of parties, the joint *view* of all parties in $\mathcal{W}$ can be denoted as a random variable $\mathbf{REAL}_{\mathcal{W}}^{\mathcal{U},t,k}(x_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4)$, where t and k indicate the threshold and security parameter used in our protocol, respectively.

Next we will present two theorems. The first theorem shows that any collusion (excluding the cloud server) less than t users in failure to get other users private information except the result of the aggregation.

THEOREM 1 (Defense against Joint Attacks from Multiple Users). *For all t, k, $\mathcal{W} \subseteq \mathcal{U}$, $x_{\mathcal{U}}$, $\mathcal{U}$ with $|\mathcal{U}| \geq t$, and $\mathcal{U}_4 \subseteq \mathcal{U}_3 \subseteq \mathcal{U}_2 \subseteq \mathcal{U}_1 \subseteq \mathcal{U}$, there is a PPT simulator **SIM** whose output is indistinguishable from the output of $\mathbf{REAL}_{\mathcal{W}}^{\mathcal{U},t,k}$.*

$$\mathbf{REAL}_{\mathcal{W}}^{\mathcal{U},t,k}(x_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4) \equiv \mathbf{SIM}_{\mathcal{W}}^{\mathcal{U},t,k}(x_{\mathcal{W}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4)$$

Proof. Because we exclude the involvement of the cloud server, the joint view of parties in set $\mathcal{W}$ does not depend on the inputs of other users not in $\mathcal{W}$. Hence, the simulator can generate a perfect simulation by running the protocol with the true inputs of honest but curious users, but replacing the inputs of the honest users with fake data (such as randomly generated number). We stress that the simulated view of users in $\mathcal{W}$ is indistinguishable from the output of real view. More concretely, in **Round 2**, the simulator generates the masked input $\hat{x_n}$ for all honest users (not in $\mathcal{W}$) by utilizing random number (such as 0), instead of using true gradients. Besides, we note that the server just sends a list of all online users' ID in the round of *Unmasking*, not the actual value of the specific $\hat{x_n}$, which means that the honest but curious users cannot identify whether the calculation results returned by the cloud server are based on the true gradients of honest users. Therefore, the simulated view of users in $\mathcal{W}$ is indistinguishable from the output of real view $\mathbf{REAL}_{\mathcal{W}}^{\mathcal{U},t,k}$.

THEOREM 2 (Defense against Joint Attacks from The Cloud Server and Multiple Users). *For all t, k, $x_{\mathcal{U}}$, $\mathcal{W} \subseteq \mathcal{U} \cup \{S\}$, $|\mathcal{W}\backslash\{S\}| < t$, $\mathcal{U}$ with $|\mathcal{U}| \geq t$, and $\mathcal{U}_4 \subseteq \mathcal{U}_3 \subseteq \mathcal{U}_2 \subseteq \mathcal{U}_1 \subseteq \mathcal{U}$, there is a PPT simulator **SIM** whose output is indistinguishable from the output of $\mathbf{REAL}_{\mathcal{W}}^{\mathcal{U},t,k}$.*

$$\mathbf{REAL}_{\mathcal{W}}^{\mathcal{U},t,k}(x_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4) \approx \mathbf{SIM}_{\mathcal{W}}^{\mathcal{U},t,k}(x_{\mathcal{W}}, \xi, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4)$$

where

$$\xi = \begin{cases} \sum_{n\in\mathcal{U}_3\setminus\mathcal{W}} x_n & \text{if } |\mathcal{U}_3| \geq t \\ \perp & \text{otherwise} \end{cases}$$

Proof. We use a standard hybrid argument here to prove our THEOREM 2. The main idea is that the simulator **SIM** executes a series of modifications to our protocol, which ultimately makes the simulated view $\mathbf{SIM}_{\mathcal{W}}^{\mathcal{U},t,k}$ indistinguishable from the real view $\mathbf{REAL}_{\mathcal{W}}^{\mathcal{U},t,k}$. In our hybrid argument, $\mathbf{hyb_i}, \{i = 1, \cdots 9\}$ indicates a secure modification to the original protocol, which ensures that the modified operation is indistinguishable from the original operation.

$\mathbf{hyb_1}$ In this hybrid, the simulator changes the behavior of all honest users n, where $n \in \{\mathcal{U}_2\backslash\mathcal{W}\}$. Specifically, for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

each user n, a uniformly random number $v_{n,m}$ is selected to replace the shared key **KA.agree** (P_n^{SK}, P_m^{PK}) between user n and m in the same set, and to perform the function of encryption and decryption. For example, each honest user utilizes $v_{n,m}$ to generate ciphertext $\mathcal{P}_{n,m}$ mentioned in **Round 1**, instead of utilizing **KA.agree** (P_n^{SK}, P_m^{PK}). The DDH assumption [**?**] ensures that this hybrid possesses the indistinguishability from real protocol.

hyb$_2$ In this hybrid, the simulator replaces all encrypted data (i.e.,the encrypted shares of β_n and N_n^{SK}) sent by honest users (in the set $\{\mathcal{U}_2\backslash\mathcal{W}\}$) to other users with encrypted shares of random values (e.g., 0, with appropriate length). However, all honest users still return the correct shares to the cloud server in the round of *Unmasking*. Because we just change the content of ciphertext, the properties of symmetric authenticated encryption [**?**], [**?**] ensure the indistinguishability between this hybrid with real protocol.

hyb$_3$ In this hybrid, we first define a subset as below.

$$\mathcal{U}^* = \begin{cases} \mathcal{U}_2\backslash\mathcal{W} & \text{if } \xi = \perp \\ \mathcal{U}_2\backslash\mathcal{U}_3\backslash\mathcal{W} & \text{otherwise} \end{cases}$$

Then, in the round of *Key Sharing*, for all honest users n in the set $\mathcal{U}^*$, the simulator replaces all the shares of β_n with random values (e.g., 0, with appropriate length). It is obvious that the adversary will not get extra share of β_n, either because the honest users will not reveal their shares of β_n (resp. $|\mathcal{U}_3| \geq t, \mathcal{U}^* = \mathcal{U}_2\backslash\mathcal{U}_3\backslash\mathcal{W}$), or because all the honest users are offline (resp. $|\mathcal{U}_3| < t, \mathcal{U}^* = \mathcal{U}_2\backslash\mathcal{W}$, where $\xi = \perp$). The security of Shamir's secret sharing scheme guarantees that it is infeasible to recover the secret even possessing any $t-1$ shares of current secret, which means that even the honest but curious users have $|\mathcal{W}| < t$ shares of β_n, they still cannot tell whether the shares submitted from honest users comes from the real β_n.

hyb$_4$ In this hybrid, instead of generating $\mathbf{PRG}(\beta_n)$ for all users in the set $\mathcal{U}^*$, the simulator uses uniformly random number r_n with appropriate size to replace it. It is easy to understand that the simulator just substitutes the output of **PRG**, where **PRG** is the Pseudorandom Generator [**?**] mentioned before. Therefore, the security of Pseudorandom Generator [**?**] ensures that this hybrid is indistinguishable from the real protocol.

hyb$_5$ In this hybrid, for each user n in the set $\mathcal{U}^*$, the simulator generates the masked input as below:

$$\hat{x_n} = r_n + \sum_{m\in\mathcal{U}_2:n<m} \mathbf{PRG}(s_{n,m}) - \sum_{m\in\mathcal{U}_2:n>m} \mathbf{PRG}(s_{m,n})$$

instead of utilizing

$$\hat{x_n} = x_n + \mathbf{PRG}(\beta_n) + \sum_{m\in\mathcal{U}_2:n<m} \mathbf{PRG}(s_{n,m}) - \sum_{m\in\mathcal{U}_2:n>m} \mathbf{PRG}(s_{m,n})$$

Since $\mathbf{PRG}(\beta_n)$ has been changed in the previous hybrid with a uniformly random number, we know that $x_n + r_n$ is also a random value, and it is easy to deduce that the distribution of r_n and $x_n + \mathbf{PRG}(\beta_n)$ is indistinguishable. It should be noted that if $\xi = \perp$, the simulator has already completed the simulation (describe as **hyb$_5$**) since **SIM** successfully simulates **REAL** without knowing x_n for all parties $n \in \mathcal{W}$. Hence for all the simulations that follow, we assume $\xi \neq \perp$.

hyb$_6$ In this hybrid, for every user $n \in \mathcal{U}_3\backslash\mathcal{W}$, the simulator substitutes the shares of N_n^{SK} with shares of random values (e.g., 0, with appropriate length). Similar to **hyb$_3$**, the security of Shamir's secret sharing protocol guarantees that this hybrid is indistinguishable from the real protocol.

hyb$_7$ In this hybrid, given a user $m' \in \mathcal{U}_3\backslash\mathcal{W}$, for all other users $n \in \mathcal{U}_3\backslash\mathcal{W}$, the simulator uniformly selects a random number to replace the sharked key (i.e., $s_{n,m'} = \mathbf{KA.agree}\{N_n^{SK}, N_m^{PK}\}$) between user n and m', and this random number will be used as the seed of **PRG** for both user n and m.
Specifically, a random value $s'_{n,m'}$ is selected for each user $n \in \mathcal{U}_3\backslash\mathcal{W}\backslash\{m'\}$. Instead of sending

$$\hat{x_n} = x_n + \mathbf{PRG}(\beta_n) + \sum_{m\in\mathcal{U}_2:n<m} \mathbf{PRG}(s_{n,m}) - \sum_{m\in\mathcal{U}_2:n>m} \mathbf{PRG}(s_{m,n})$$

SIM submits

$$\hat{x_n} = x_n + r_n + \sum_{m\in\mathcal{U}_2\backslash\{m'\}:n<m} \mathbf{PRG}(s_{n,m}) - \sum_{m\in\mathcal{U}_2\backslash\{m'\}:n>m} \mathbf{PRG}(s_{m,n}) + \Delta_{n,m'} \cdot \mathbf{PRG}(s'_{n,m'})$$

where $\Delta_{n,m'} = 1$ if $n < m'$. Otherwise, $\Delta_{n,m'} = -1$. Correspondingly, we have

$$\hat{x_{m'}} = x_{m'} + r_{m'} + \sum_{n\in\mathcal{U}_2} \Delta_{n,m'} \cdot \mathbf{PRG}(s'_{n,m'})$$

Similarly, the DDH assumption [**?**] ensures that this hybrid possesses the indistinguishability from real protocol.

hyb$_8$ In this hybrid, for the same user m' selected in previous hybrid and all other user $n \in \mathcal{U}_3\backslash\mathcal{W}$, the simulator also uniformly selects a random number $r_{n,m'}$ to replace the computation of $\mathbf{PRG}(s'_{n,m'})$. Similar to **hyb$_4$**, it is easy to understand that the simulator just substitutes the output of **PRG**. Therefore, the security of Pseudorandom Generator [**?**] ensures that this hybrid is indistinguishable from the real protocol.

hyb$_9$ In this hybrid, for each user n in the set $\mathcal{U}_3\backslash\mathcal{W}$, the simulator submits

$$\hat{x_n} = R_n + r_n + \sum_{m\in\mathcal{U}_2\backslash\mathcal{U}_3\backslash\mathcal{W}} \Delta_{n,m} \cdot r_{n,m}$$

instead of sending

$$\hat{x_n} = x_n + \mathbf{PRG}(\beta_n) + \sum_{m\in\mathcal{U}_2:n<m} \mathbf{PRG}(s_{n,m}) - \sum_{m\in\mathcal{U}_2:n>m} \mathbf{PRG}(s_{m,n})$$

where $\{R_n\}_{n\in\mathcal{U}_3\backslash\mathcal{W}}$ is random value selected by the simulator, and subjected to $\sum_{n\in\mathcal{U}_3\backslash\mathcal{W}} R_n = \sum_{n\in\mathcal{U}_3\backslash\mathcal{W}} x_n = \xi$. Therefore,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

the simulator has already completed the simulation since **SIM** successfully simulates **REAL** without knowing x_n for all parties $n \in \mathcal{W}$. Based on the hybrid 1 to 9, we can infer that the distribution of this hybrid is identical to the real output. Completing the proof.

VII. PERFORMANCE EVALUATION

We recruit 600 mobile devices to evaluate the performance of our VerifyNet, where most smart devices come with 4GB of RAM and are equipped with Android 6.0 systems. Each mobile device runs the same convolutional neural network offline to obtain the local gradients of all parameters. All the raw data are selected from MNIST database (http://yann.lecun.com/exdb/mnist/) which has a training set of 60,000 examples, and a test set of 10,000 examples. Besides, the " Cloud " is simulated with a Lenovo server which has Intel(R) Xeon(R) E5-2620 2.10GHZ CPU, 16GB RAM, 256SSD, 1TB mechanical hard disk and runs on the Ubuntu 18.04 operating system. More specifically, we adopt the key agreement protocol based on Elliptic-Curve to achieve the key distribution between two users, and the standard Shamir's t-out-of-$\mathcal{N}$ secret sharing protocol [**?**] to generate the shares of secret. In addition, we use AES in counter mode and AES-GCM with 128-bit keys to achieve the authenticated encryption and pseudorandom generator, respectively.

A. Functionality

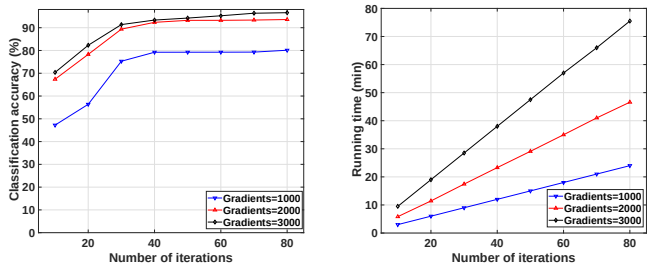
TABLE I: Comparison of Functionality

	PPML	PPDL	SafetyNets	VerifyNet
Data Privacy	✓	✓	×	✓
Robustness to Failures	✓	×	×	✓
Verifiability	×	×	✓	✓

As shown in TABLE.I, we compare the functionality with the latest work PPML [**?**], PPDL [**?**] and SafetyNets [**?**], since the main works of these schemes are similar to ours. Specifically, we know that both PPML and PPDL guarantee the confidentiality of data privacy during the execution, however, the property of verifiability is not supported by their model. In addition, PPDL is also failure to deal with the problem of users dropping out. On the other hand, SafetyNets is primarily designed from the verifiability perspective, hence the problems of data privacy leakage and users dropping out in the training process are not considered in its protocol. Compared with these schemes, our VerifyNet supports each user to verify the results returned by the cloud server while guaranteeing the confidentiality of user's local gradients. Besides, similar to PPML, VerifyNet is also robust to users dropping out at any subprocess of whole work process.

B. Classification Accuracy

In this section, we select data from MNIST database to test the classification accuracy of our VerifyNet. The experiments were conducted on a CNN network [**?**], [**?**], which consists of two convolutional layers and two fully connected layers with



(a) (b)

Fig. 5: (a) No dropout, $|\mathcal{U}|$=100, classification accuracy with the different number of gradients per user. (b) No dropout, $|\mathcal{U}|$=100, running time with the different number of gradients per user.

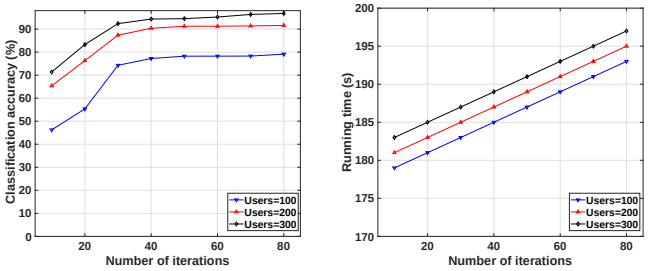
128 neurons each layer. Definitely, in federated learning, the accuracy of model's outputs is closely related to two factors, i.e., the number of users participating in the training and the size of the local gradients owned by each user. In general, the accuracy of the model's output is proportional to the number of gradients/users involved in the training, and also proportional to the computation and communication overhead generated by the system. To analyze the relationship between these factors, we record the classification accuracy and running time of our VerifyNet under different number of users/gradients per user. Here we use the symbol $|\mathcal{U}|$ and $|\mathcal{G}|$ to indicate the number of users and gradients per user in our experiments, respectively.

Fig. 5 shows the classification accuracy and running time with the different number of gradients per user, where an iteration means that a parameter update (i.e., **Round 0** to **Round 4**) is completed. For simplicity, here we only consider the case of no users dropping out. Clearly, the increase in the number of gradients facilitates the higher accuracy of the model output, but it also incurs more computation overhead (shown in Fig. 5(b)). However, by comparing classification accuracy with different gradients (See gradients=2000 and gradients=3000, respectively), the number of gradients involved in training is not the more the better, because the accuracy of the model will converge when the number of gradients increases to a certain amount. Therefore, in practical applications, we can empirically choose the appropriate number of gradients to avoid unnecessary overhead. Fig. 6 shows the classification accuracy and running time with the different number of users. Similarly, increasing the number of users in the system is beneficial to improve the model's classification accuracy, but it also requires additional computation overheads. Note that for the sake of simplicity, we do not record the total amount of data transmitted in the system under different numbers of users/gradients. However, since VerifyNet is an interactive protocol, in theory, our scheme will inevitably generate a certain communication overhead as the number of users/gradients increases.

C. Verification Accuracy

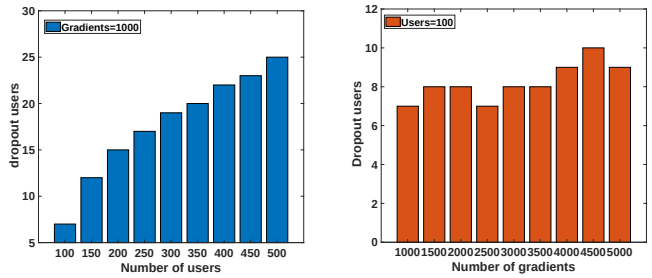
As discussed before, to prevent malicious spoofing by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security



(a) (b)

Fig. 6: (a) No dropout, $|\mathcal{G}|$=1000, classification accuracy with the different number of users. (b) No dropout, $|\mathcal{G}|$=1000, running time with the different number of users.



(a) (b)

Fig. 8: Dropout users. (a) $|\mathcal{G}|$=1000, with the different number of users. (b) $|\mathcal{U}|$=100, with the different number of gradients per user.

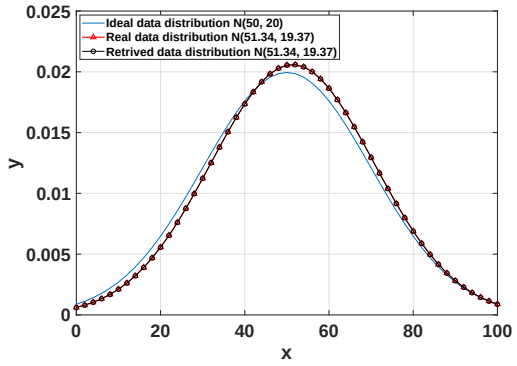
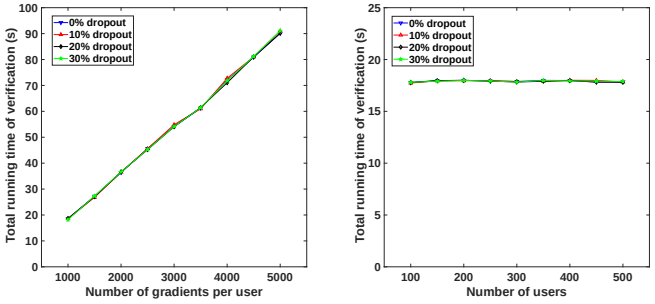


Fig. 7: Verification accuracy



(a) (b)

Fig. 9: Total running time of each user (Verification process). (a) $|\mathcal{U}|$=100, with the different number of gradients per user. (b) $|\mathcal{G}|$=1000, with the different number of users.

the server, our VerifyNet supports each user to verify the correctness of the results returned by the server. In specific, the cloud server is required to provide the *Proof* about the correctness of its aggregated results to each user, and each user can reject or accept the results by checking the *Proof*. To give a simple presentation for the verification accuracy, we simulate 200 users uploading encrypted local data to the server, where all the data are randomly selected from normal distribution $N(50, 20)$. For simplicity, here we also only consider the case of no users dropping out. Since the randomly selected data are discrete points, their real distribution $(N(51.34, 19.37)$, red line in Fig. 7) is slightly different from the original ideal distribution. Then, we require the cloud server to calculate the aggregated results along with corresponding *Proof* for each user. If the verification is passed, the distribution of uploaded data used to generate the aggregated results should be the same as the real data. Based on this, we further use the aggregated results to calculate the mean and variance of the uploaded data. As shown in Fig. 7, we can find that retrieved data distribution is exactly overlapping with the real data distribution, which also confirms that a result returned from the server is correct once its *Proof* is verified.

D. Probability of Users Dropping Out

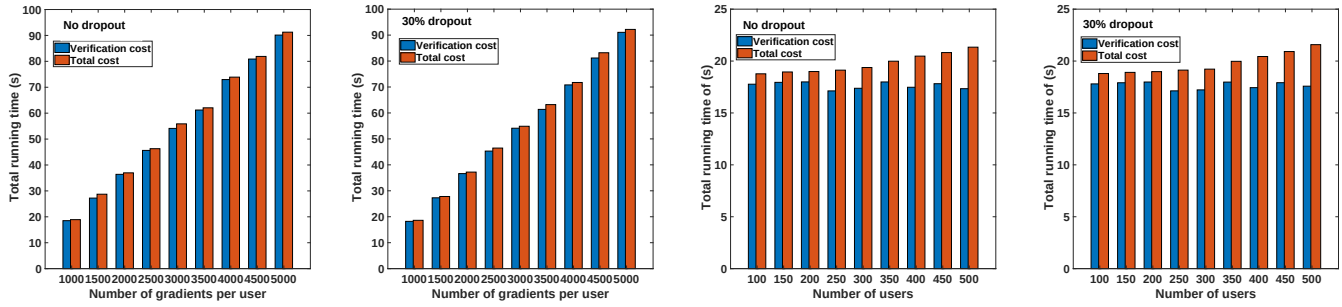
Our VerifyNet is robust to users dropping out in training process, because users dropping out is very common due to users' device battery issues, hardware quality problems and the like occurring in workflow. To evaluate the universality of this phenomenon, we record the number of users who logged out under different number of users/gradients per user in whole system. In specific, we require all users to upload data to the server multiple times within the specified time, and record the average of dropout users under repeated experiments. As shown in Fig. 8, we find that as the number of users/gradients increases, a certain number of users dropping out are inevitable, which is more pronounced as the number of users increases. However, Fig. 8 shows that the proportion of users dropping out is not significant relative to the total number of users. Hence, this also provides a basis for using the secret sharing protocol to manage the problem of users dropping out.

E. Performance Analysis of Client

We analyze the performance of the client from both computation and communication overhead, where we test VerifyNet under different proportions of users dropping out.

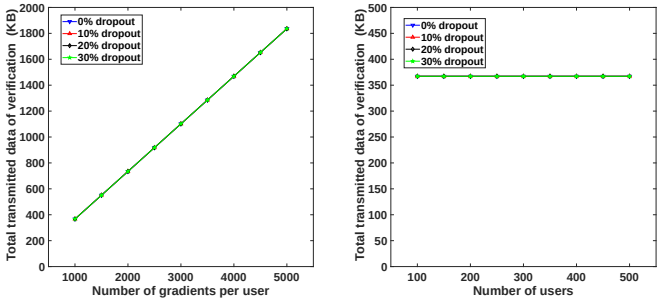
1) Computation Overhead: Fig. 9 shows the running time of each user during verification process. Clearly, the user's

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security



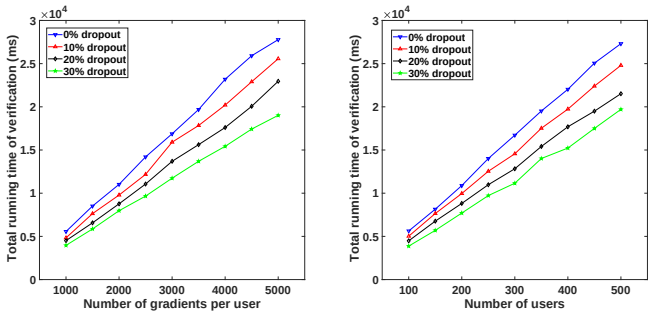
(a) (b) (c) (d)

Fig. 10: Comparison between verification computation overhead and total overhead for each user. (a) No dropout, $|\mathcal{U}|$=100, with the different number of gradients per user. (b) 30% dropout, $|\mathcal{U}|$=100, with the different number of gradients per user. (c) No dropout, $|\mathcal{G}|$=1000, with the different number of users. (d) 30% dropout, $|\mathcal{G}|$=1000, with the different number of users.



(a) (b)

Fig. 11: Total transmitted data of each user (Verification process). (a) $|\mathcal{U}|$=100, with the different number of gradients per user. (b) $|\mathcal{G}|$=1000, with the different number of users.



(a) (b)

Fig. 13: Total running time of the cloud server (Verification process). (a) $|\mathcal{U}|$=100, with the different number of gradients per user. (b) $|\mathcal{G}|$=1000, with the different number of users.

running time increases linearly with the increasing of the number of gradients, but keeps a constant as the number of users increases. One of main reasons is that the verification overhead is only related to the number of gradients owned by each user, since each user n needs to generate (A_n, B_n, L_n, Q_n) for newly added gradient x_n. Fig.10 shows the comparison between the computation overhead of verification and the total overhead. For simplicity, we consider no user dropping out and 30% users dropping out in our experiments. We can see that the main computation cost of each user comes from the verification process, regardless of the number of users or gradients. In addition, our VerifyNet maintains good performance in terms of computation overhead. For example, when the number of users is 500 and the total number of gradients in our system is 500000, each user only needs about 17 seconds to complete one iteration of parameter update.

2) Communication Overhead: Fig. 11 shows the total transmitted data of each user during verification process. Similar to Fig.9, the user's total transmitted data also increases linearly with the increasing of the number of gradients, but keeps a constant as the number of users increases. Fig.12 shows the comparison between verification communication overhead and total overhead of each user. We can see that the proportion of overhead generated in verification process is not obvious to total overhead, and even can be ignored as the number of users increases. Moreover, experiments demonstrate that our VerifyNet still maintains good performance in terms of communication overhead. For instance, when the number of users is 500 and the total number of gradients in our system is 500000, each user only needs about $70MB$ to complete one iteration of parameter update.

F. Performance Analysis of Server

1) Computation Overhead: Fig.13 shows the running time of verification process of the cloud server. We can see that the server's running time increases linearly with the increasing of the number of gradients or users. The main reason is that as the number of gradients or users increases, the cloud server needs to generate the *Proof* of aggregated result for each new added gradients and users. Fig. 14 shows the comparison between verification computation overhead and total overhead of the cloud server. We can find that the proportion of users dropping

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

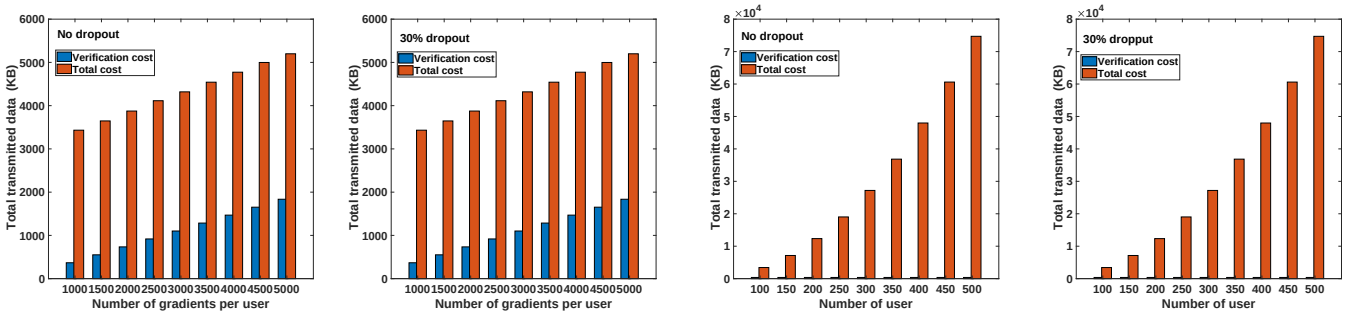


Fig. 12: Comparison between verification communication overhead and total overhead for each user. (a) No dropout, $|\mathcal{U}|$=100, with the different number of gradients per user. (b) 30% dropout, $|\mathcal{U}|$=100, with the different number of gradients per user. (c) No dropout, $|\mathcal{G}|$=1000, with the different number of users. (d) 30% dropout, $|\mathcal{G}|$=1000, with the different number of users.

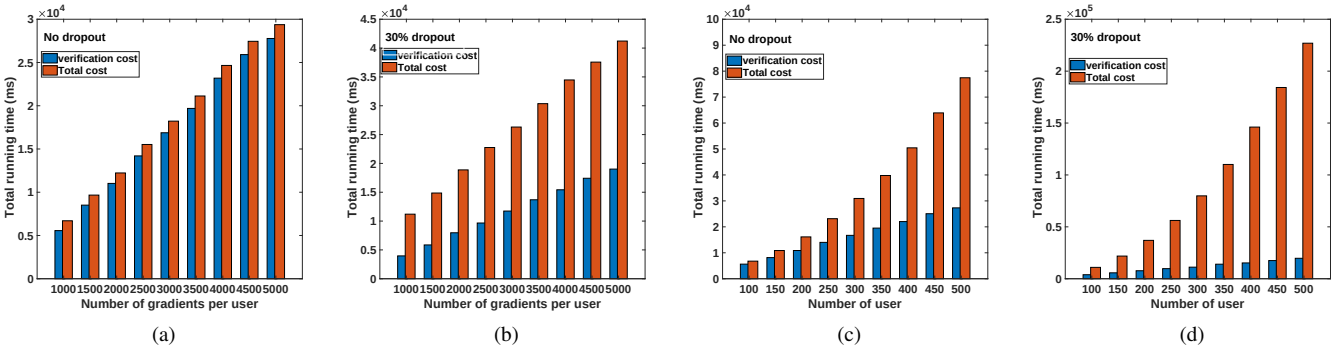


Fig. 14: Comparison between verification computation overhead and total overhead for the cloud server. (a) No dropout, $|\mathcal{U}|$=100, with the different number of gradients per user. (b) 30% dropout, $|\mathcal{U}|$=100, with the different number of gradients per user. (c) No dropout, $|\mathcal{G}|$=1000, with the different number of users. (d) 30% dropout, $|\mathcal{G}|$=1000, with the different number of users.

out greatly determine the trend of overall cost of the cloud server, which is also obvious by comparing with Fig.14(c) and Fig.14(d). For example, when no user dropouts, the cloud sever only needs about $75000ms$ to complete an iteration of parameter updates, but it will take $220000ms$ if 30% of users dropout.

2) Communication Overhead: Fig. 15 shows the total transmitted data of verification process of the cloud server. We can see that as the number of users or gradients increases, the communication overhead of the cloud server also grows linearly. TABLE.I and TABLE.II show the computation and communication overhead of each round, respectively, where the red font indicates the overhead during the verification process. For each user, both the computational and communication overhead are mainly from the *Masked Input* and *Verification*, since each user n needs generating σ_n and verifying *Proof* for each gradient, and sending the encrypted results to the cloud server. For the cloud server, after receiving all the messages on the *Masked Input* round, it needs to aggregate the encrypted gradients of all users and restore the secrets of all the online users in the *Unmasking* round, which results in large computational/

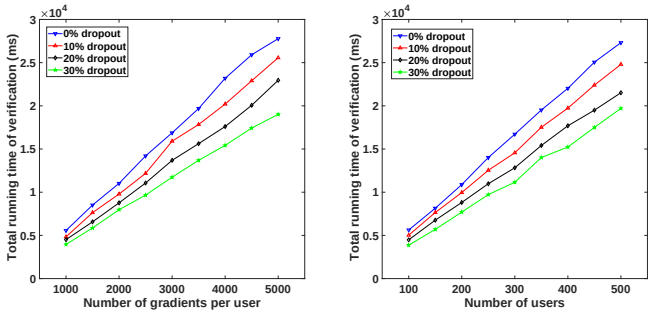


Fig. 15: Total transmitted data of the cloud server (Verification process). (a) $|\mathcal{U}|$=100, with the different number of gradients per user. (b) $|\mathcal{G}|$=1000, with the different number of users.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

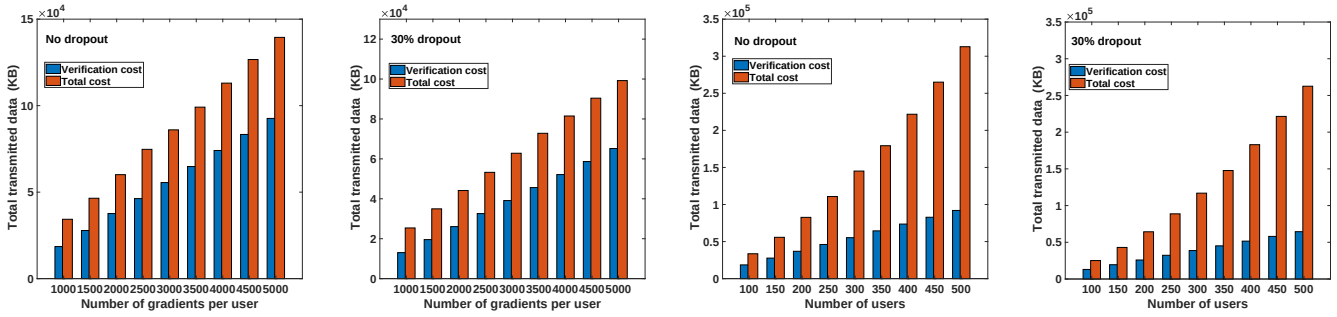


Fig. 16: Comparison between verification communication overhead and total overhead for the cloud server. (a) No dropout, $|\mathcal{U}|$=100, with the different number of gradients per user. (b) 30% dropout, $|\mathcal{U}|$=100, with the different number of gradients per user. (c) No dropout, $|\mathcal{G}|$=1000, with the different number of users. (d) 30% dropout, $|\mathcal{G}|$=1000, with the different number of users.

TABLE II: Computation Overhead of Each Round

	Dropout	Key Sharing	Masked Input	Unmasking	Verification	Total
Client	0%	2508(ms)	$(1311 + 11919)$ (ms)	4(ms)	5941(ms)	21683 (ms)
Client	10%	2510(ms)	$(1263 + 11942)$(ms)	4(ms)	5830 (ms)	21549 (ms)
Client	20%	2492(ms)	$(1250 + 11892)$(ms)	4(ms)	5942 (ms)	21580 (ms)
Client	30%	2480(ms)	$(1245 + 11971)$ (ms)	4(ms)	5942 (ms)	21642 (ms)
Server	0%	0(ms)	0 (ms)	$(50136 + 27311)$(ms)	0 (ms)	77477 (ms)
Server	10%	0(ms)	0 (ms)	$(112379 + 24799)$(ms)	0 (ms)	137178 (ms)
Server	20%	0(ms)	0 (ms)	$(163551 + 21519)$(ms)	0 (ms)	185070 (ms)
Server	30%	0(ms)	0 (ms)	$(207222 + 19705)$(ms)	0 (ms)	226927 (ms)

TABLE III: Communication Overhead of Each Round

	Dropout	Key Sharing	Masked Input	Unmasking	Verification	Total
Client	0%	274(KB)	$(74002 + 184)$ (KB)	65(KB)	$(4 + 184)$(KB)	73 (MB)
Client	10%	273(KB)	$(73998 + 184)$ (KB)	65(KB)	$(4 + 184)$(KB)	73 (KB)
Client	20%	274(KB)	$(73999 + 184)$(KB)	65(KB)	$(4 + 184)$ (KB)	73 (MB)
Client	30%	274(KB)	$(73998 + 184)$(KB)	65(KB)	$(4 + 184)$ (KB)	73 (MB)
Server	0%	72(MB)	$(111 + 90)$ (MB)	$(32 + 0.18)$(MB)	0 (MB)	305.18 (MB)
Server	10%	72(MB)	$(110 + 81)$ (MB)	$(29 + 0.18)$(MB)	0 (MB)	292.18 (MB)
Server	20%	72(MB)	$(102 + 72)$ (MB)	$(25 + 0.18)$(MB)	0 (MB)	271.18 (MB)
Server	30%	72(MB)	$(98 + 63)$ (MB)	$(22 + 0.18)$(MB)	0 (MB)	255.18 (MB)

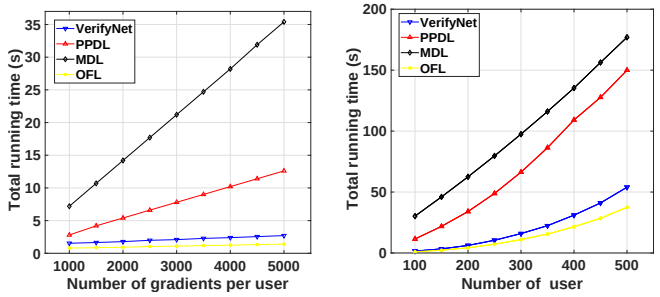
communication overheads.

G. Performance Analysis by Comparing with Existing Approaches

In this section, we analyze the cost of VerifyNet by comparing with the state-of-the-art approaches MDL [**?**], PPDL [**?**], SafetyNets [**?**] and Original Federated Learning model OFL[**?**], where OFL is the original model for performing federated learning in the plaintext environment. Here we use OFL to describe the performance differences between federated learning in plaintext and ciphertext. MDL [**?**] and PPDL [**?**] are consistent with the scenarios considered in our VerifyNet, and their goal is also to protect the privacy of user's local gradients by using privacy protection techniques. However, they do not consider the verifiability issue of the results returned by the server. Conversely, SafetyNets aims [**?**] to verify the correctness of results returned from the cloud, which is the first approach for verifiable execution of deep neural networks on untrusted cloud. It can convert certain types of deep neural networks into arithmetic circuits, and then verify the correctness of returned results through multiple interactions with the server.

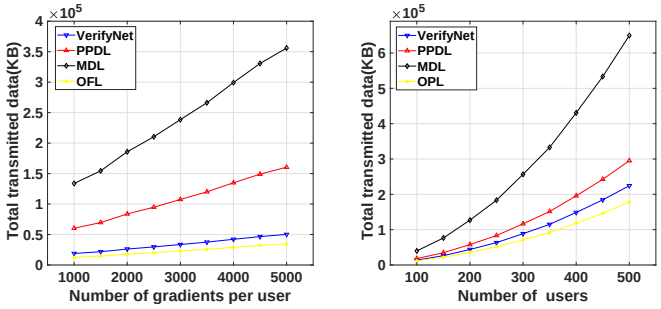
1) Performance of Encryption Process: As shown in Fig.17 and Fig.18, we record the running time and total transmitted data of VerifyNet, MDL, PPDL and OFL with different number of users/gradients per user. Since verifiable calculations

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security



(a) (b)

Fig. 17: Running time. (a) $|\mathcal{U}|$=100, with the different number of gradients per user. (b) $|\mathcal{G}|$=1000, with the different number of users

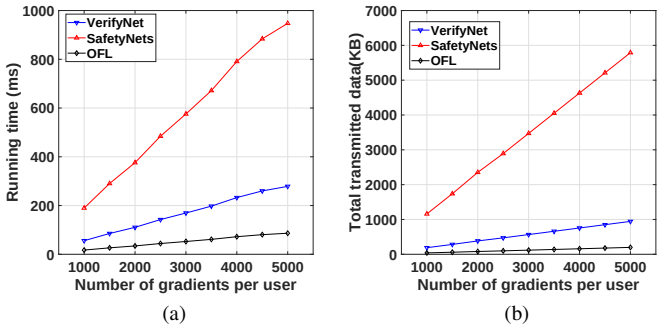


(a) (b)

Fig. 18: Total transmitted data. (a) $|\mathcal{U}|$=100, with the different number of gradients per user. (b) $|\mathcal{G}|$=1000, with the different number of users

are not considered in MDL and PPDL, the computation and communication overhead required for verification are also excluded from our VerifyNet. In addition, here we only consider the case of no users dropping out because MDL and PPDL are not supportive for users dropping out in training process. We can see that the cost of VerifyNet is significantly smaller than MDL and PPDL, while not much larger than the original solution OFL. This is mainly due to the high efficiency of our double-masking protocol compared with technologies used in MDL and PPDL. Specifically, ElGamal cryptosystem [**?**] is used in MDL to encrypt users' local gradients, while guaranteeing multiplicative homomorphism over encrypted domain. However, since encrypting each gradient involves multiple exponential operations, ElGamal is not suitable for federated learning that is driven by large-scale data. LWE-based homomorphic encryption [**?**] is exploited in PPDL, which is faster than ElGamal cryptosystem. However, its computation/ communication overhead also grows significantly as the number of users/gradients per user increases. Compared with MDL and PPDL, we design a double-masking protocol to encrypt users' local gradients. Since we do not consider the case of users dropping out in training process, each user n only needs to calculate the shares of N_n^{PK} once. As a result, the encryption operation is equivalent to adding several random values to each gradient, which greatly reduces the computation and communication overhead in the encryption process.

2) Performance of Verification Process: By comparing with the works SafetyNets [**?**] and OFL[**?**], we evaluate the performance of VerifyNet during the verification process. The scenario implemented in SafetyNets is different from our VerifyNet, which aims to verify the correctness of the results returned by the server during the prediction process, and only considers the number of users in whole system is 1. In order to compare the overhead in the same experimental environment, we set $|\mathcal{U}|$=1, and exploit the verifiable technologies of SafetyNets to accomplish the same task of our VerifyNet. In addition, here we only consider the case of no users dropping out. Then, we record the running time and total transmitted data of three schemes with different number of gradients per user. Fig.19 shows that the cost of VerifyNet and SafetyNets



(a) (b)

Fig. 19: (a) $|\mathcal{U}|$=1, with the different number of gradients per user. (b) $|\mathcal{U}|$=1, with the different number of gradients per user.

are significant compared with original model OFL. However, the performance of our VerifyNet is significantly better than SafetyNets. One reason for this is the technical limitations of SafetyNets, and the other reason is the combination of Homomorphic Hash and pseudo-random functions exploited in our proposed protocol. Specifically, SafetyNets uses the Interactive Proof Systems [**?**] to check the correctness of the calculated result returned by the cloud server. It requires multiple interactions and calculations with the server to complete the verification task, and has been shown to be flawed in computation and communication overheads [**?**]. However, we exploit the homomorphic hash function integrated with pseudorandom technology as the underlying structure of VerifyNet, which are well known for efficiently processing of data. Hence, our VerifyNet can ensure users to verify the correctness of results returned by the cloud server with relatively low overhead.

VIII. RELATED WORKS

In this section, we introduce the latest related works of deep learning in terms of privacy protection and verifiability.

A. Privacy-Preserving Deep Learning

Most deep learning-based privacy protection algorithms focus on protecting users' data privacy. The main tools used in their protocols are differential privacy, secure multi-party computing [**?**], [**?**], and cryptographic primitives [**?**]. However, the issue of privacy leakage is still not completely addressed. For example, Shokri et al. [**?**] proposed a privacy-preserving deep learning approach by utilizing differential privacy to achieve the balance between security and accuracy. Unfortunately, any differential privacy-based strategy has been exposed to be insecure [**?**] if adversaries utilize the GAN network to attack the protocol. Trieu Phong et al. [**?**] proposed a more secure deep learning system by utilizing additively homomorphic encryption and asynchronous stochastic gradient, but the implementation requires all users to share the same key for expected security level. Recently, Keith Bonawitz et al. [**?**] designed a federated deep learning approach utilizing secure multi-party computing to protect the local gradient of each user, which is supportive for users offline during the training process.

B. Verifiable Deep Learning

In deep learning, the cloud server may return incorrect results to the user due to unexpected situations. To combat that, Several schemes [**?**], [**?**] have been successively proposed to alleviate this problem. For example, Zahra Ghodsi et al. [**?**] designed a framework called SafetyNets. It uses the Interactive Proof Systems [**?**] to check the correctness of the calculated result returned by the cloud server. Later, Florian Tramr et al. [**?**] proposed a verifiable scheme called Slalom to perform verification by exploiting trusted hardware such as SGX, TrustZone and Sanctum. However, these schemes either support a small variety of activation functions or require additional hardware assistance. More notably, to the best of our knowledge, for a neural network being trained, there is no solution which supports the verifiability to the correctness of computation results from the cloud. Compared with existing approaches, we propose VerifyNet, the first privacy-preserving approach supporting verification in the process of training neural networks. We first utilize homomorphic hash function integrated with pseudorandom technology to support the verifiability for each user. Then, we use a variant of secret sharing technology along with key agreement protocol to protect the privacy of users' local gradients, and deal with the users dropping out problem during the training process.

IX. Conclusion

In this paper, we have proposed VerifyNet which supports verification of the server's calculation results to each user. Besides, VerifyNet is supportive for users dropping out in training process. Security analysis shows the high security of our VerifyNet under the honest but curious security setting. In addition, experiments conducted on real data also demonstrate the practical performance of our proposed scheme. As part of future research work, we will focus on reducing the communication overhead of the entire protocol.

ACKNOWLEDGMENT

This work is supported by the National Key R&D Program of China under Grants 2017YFB0802300 and 2017YFB0802000, the National Natural Science Foundation of China under Grants 61802051, 61772121, 61728102, and 61472065, the Fundamental Research Funds for Chinese Central Universities under Grant ZYGX2015J056.

References

[1] K. Ahiska, M. K. Ozgoren, and M. K. Leblebicioglu, "Autopilot design for vehicle cornering through icy roads," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 3, pp. 1867–1880, 2018.

[2] G. Xu, H. Li, and R. Lu, "POSTER:practical and privacy-aware truth discovery in mobile crowd sensing systems," in *Proceedings of ACM CCS*, 2018, pp. 2312–2314.

[3] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *proceedings of IEEE S&P*, 2017, pp. 19–38.

[4] Y. Zhang, C. Xu, H. Li, K. Yang, J. Zhou, and X. Lin, "Healthdep: An efficient and secure deduplication scheme for cloud-assisted ehealth systems," *IEEE Transactions on Industrial Informatics*, 2018.

[5] G. Xu, H. Li, S. Liu, M. Wen, and R. Lu, "Efficient and privacy-preserving truth discovery in mobile crowd sensing systems," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 4, pp. 3854–3865, 2019.

[6] A. Datta, M. Fredrikson, G. Ko, P. Mardziel, and S. Sen, "Use privacy in data-driven systems: Theory and experiments with machine learnt programs," in *proceedings of ACM CCS*, 2017, pp. 1193–1210.

[7] G. Xu, H. Li, Y. Dai, K. Yang, and X. Lin, "Enabling efficient and geometric range query with access control over encrypted spatial data," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 870–885, 2019.

[8] Y. Zhang, C. Xu, X. Liang, H. Li, Y. Mu, and X. Zhang, "Efficient public verification of data integrity for cloud storage systems from indistinguishability obfuscation," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 3, pp. 676–688, 2017.

[9] G. Xu, H. Li, C. Tan, D. Liu, Y. Dai, and K. Yang, "Achieving efficient and privacy-preserving truth discovery in crowd sensing systems," *Computers & Security*, vol. 69, pp. 114–126, 2017.

[10] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.

[11] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. Mcmahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of ACM CCS*, 2017, pp. 1175–1191.

[12] Y. Liu, S. Ma, Y. Aafer, W. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *proceedings of NDSS*, 2018.

[13] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *proceedings of NDSS*, 2018.

[14] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the GAN: information leakage from collaborative deep learning," in *proceedings of ACM CCS*, 2017, pp. 603–618.

[15] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *proceedings of IEEE S&P*, 2017, pp. 3–18.

[16] Z. Ghodsi, T. Gu, and S. Garg, "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud," in *Advances in Neural Information Processing Systems*, 2017, pp. 4672–4681.

[17] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.

[18] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of ACM CCS*, 2015, pp. 1310–1321.

[19] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[20] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT*. Springer, 2010, pp. 177–186.

[21] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2019.2929409, IEEE Transactions on Information Forensics and Security

[22] H. Li, D. Liu, Y. Dai, and T. H. Luan, "Engineering searchable encryption of mobile cloud networks: when qoe meets qop," *IEEE Wireless Communications*, vol. 22, no. 4, pp. 74–80, 2015.

[23] H. Li, D. Liu, Y. Dai, T. Luan, and S. Yu, "Personalized search over encrypted data with efficient and secure updates in mobile clouds," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 1, pp. 97–109, 2018.

[24] A. Yun, J. H. Cheon, and Y. Kim, "On homomorphic signatures for network coding," *IEEE Transactions on Computers*, vol. 59, no. 9, pp. 1295–1296, 2010.

[25] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *Proceedings of the ACM CCS*, 2014, pp. 844–855.

[26] A. Shamir, *How to share a secret*. Communications of the ACM, 1979.

[27] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[28] H. Li, D. Liu, Y. Dai, T. H. Luan, and X. S. Shen, "Enabling efficient multi-keyword ranked search over encrypted mobile cloud data through blind storage," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 1, pp. 127–138, 2015.

[29] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," in *Symposium on Foundations of Computer Science*, 2008, pp. 112–117.

[30] P. Rewagad and Y. Pawar, "Use of digital signature with diffie hellman key exchange and aes encryption algorithm to enhance data security in cloud computing," in *Proceedings of the IEEE CSNT*, 2013, pp. 437–439.

[31] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," *Journal of Cryptology*, vol. 21, no. 4, pp. 469–491, 2008.

[32] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology — CRYPTO 2001*, J. Kilian, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 213–229.

[33] X. Zhang, S. Ji, H. Wang, and T. Wang, "Private, yet practical, multiparty deep learning," in *Proceedings of IEEE ICDCS*. IEEE, 2017, pp. 1442–1452.

[34] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[35] Z. Brakerski, C. Gentry, and S. Halevi, "Packed ciphertexts in lwe-based homomorphic encryption," in *International Workshop on Public Key Cryptography*. Springer, 2013, pp. 1–13.

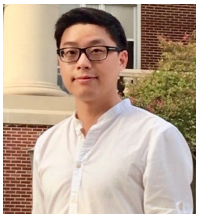
[36] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *Advances in Cryptology - CRYPTO*, 2013, pp. 71–89.

[37] J. Keuffer, R. Molva, and H. Chabanne, "Efficient proof composition for verifiable computation," in *European Symposium on Research in Computer Security*. Springer, 2018, pp. 152–171.

Guowen Xu (S'15) received his B.S. degree in information and computing science from Anhui University of Architecture in 2014. Currently, he is a Ph.D. student at the School of Computer Science and Engineering, University of Electronic Science and Technology of China , China. His research interests include Cryptography, Searchable Encryption, and the Privacy-preserving Deep Learning.

Hongwei Li (M'12-SM'18) is currently the Head and a Professor at Department of Information Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China. Dr. Li serves as the Associate Editor of IEEE Internet of Things Journal, and Peer-to-Peer Networking and Applications, the Guest Editor of IEEE Network and IEEE Internet of Things Journal. He also serves/served the technical symposium co-chair of ACM TUR-C 2019, IEEE ICCC 2016, IEEE GLOBECOM 2015 and IEEE BigDataService 2015, and many technical program committees for international conferences, such as IEEE INFOCOM, IEEE ICC and IEEE GLOBECOM. He won the Best Paper Award from IEEE MASS 2018 and IEEE HELTHCOM 2015. He is the Senior Member of IEEE, Distinguished Lecturer of IEEE Vehicular Technology Society.

Sen Liu (S'17) received the BS degree in information security from Guizhou University in 2017. Currently, he is working toward the Masters degree at the School of Computer Science and Engineering, University of Electronic Science and Technology of China. His research interests include Cryptography, Searchable Encryption.



Kan Yang(M'13) received the B.Eng. degree in information security from the University of Science and Technology of China in 2008 and the Ph.D. degree in computer science with Outstanding Research Thesis Award from the City University of Hong Kong in 2013. He is an Assistant Professor with the Department of Computer Science and the Associate Director of the Center for Information Assurance, University of Memphis. His research interests focus on security and privacy in cloud computing, big data, Internet of Things, information-centric network, and distributed systems.

Xiaodong Lin (M'09-SM'12-F'17) received the PhD degree in Information Engineering from Beijing University of Posts and Telecommunications, China, and the PhD degree (with Outstanding Achievement in Graduate Studies Award) in Electrical and Computer Engineering from the University of Waterloo, Canada. He is currently an associate professor in the School of Computer Science at the University of Guelph, Canada. His research interests include computer and network security, privacy protection, applied cryptography, computer forensics, and software security. He is a Fellow of the IEEE.