PriVeriFL: Privacy-Preserving and Aggregation-Verifiable Federated Learning

Lulu Wang, Mirko Polato, *Associate Member, IEEE,* Alessandro Brighente, *Member, IEEE,* Mauro Conti, *Fellow, IEEE,* Lei Zhang, *Member, IEEE,* and Lin Xu

***Abstract*—Federated learning provides a collaborative way to build machine learning models without sharing private data. However, attackers might infer private information from model updates submitted by participants, and the aggregator might maliciously forge the final aggregation results. Federated learning still faces data privacy and aggregation integrity challenges. In this paper, we combine inference attacks and information theory to analyze the sensitivity of different bits of model parameters. We conclude that not all bits of model parameters will leak privacy. This realization inspires us to propose a novel low-expansion homomorphic aggregation scheme based on Paillier homomorphic encryption (PHE) for safeguarding participants' data privacy. Building upon this, we develop PriVeriFL-A, a privacy-preserving and aggregation-verifiable federated learning scheme that combines homomorphic hash function and signature. To prevent collusion attacks between the aggregator and malicious participants, we further improve our PHE-based scheme into a threshold PHE-based one, named PriVeriFL-B. Compared with the privacy-preserving federated learning scheme based on classic PHE, PriVeriFL-A reduces the communication overhead to 1.65%, and the encryption/decryption computation overhead to 0.88%. Both PriVeriFL-A and PriVeriFL-B can effectively verify the integrity of the global model, while maintaining an almost negligible communication overhead for integrity verification and protecting the privacy of participants' data.**

***Index Terms*—Federated learning, data privacy, aggregation integrity, homomorphic encryption, homomorphic hash.**

I. Introduction

Traditional machine learning (ML) collects data on a central server to train a model. In many industries, however, data is fragmented and locked in multiple organizations, resulting in *data islands* (aka *data silos*). Due to growing concerns about data privacy and legal restrictions [1], [2], data collection processes that could lead to data leakage and violate privacy are strictly prohibited.

Federated learning (FL) [3]–[7] provides a novel approach to build personalized models without directly collecting private data, thus effectively dealing with data silos. Since its inception, FL has found diverse applications, ranging from Google's keyboard application (Gboard [8]), to pharmaceutical labs (MELLODDY [9]). Unlike traditional ML, which relies on centralized model training, FL enables users to collaboratively leverage global models trained on their data without the necessity of centralizing data storage. This distributed approach not only facilitates collaboration but also enhances privacy and data security. A typical FL architecture consists of a central server (usually referred to as the aggregator and assumed to be malicious in our scheme) and multiple participants who own local private data. When the system is initialized, the aggregator sends an initial model to all participants. Each participant then uses its local dataset to train a local model and sends a corresponding model update to the aggregator. Then, in the aggregation phase, the aggregator combines the model updates from the participants into an updated global model using a suitable aggregation function. Afterward, the updated global model is sent back to all participants, and a new federated round may start. This iterative process terminates when the maximum number of epochs or training time is reached, or the accuracy of the global model is high enough.

While offering significant advantages such as collaborative learning and data locality, FL is inherently challenged by serious data privacy concerns [10]. During the training process, participants need to submit local model updates, which, if transmitted without robust privacy safeguards, could become vulnerable to interception by attackers. Such vulnerabilities allow attackers to infer the participants' local data-related information indirectly [11], [12]. Existing works have demonstrated that attackers can exploit these model updates to extract critical details, including the labels and membership of participants' local datasets, and potentially even reconstruct the original training data. The threat landscape in FL is diverse. It includes not only external adversaries but also internal threats from curious participants within the federated network or a curious aggregator overseeing the learning process. The risk escalates when multiple attackers collaborate. They may target an honest participant, aiming to extract their data in a collusive attack. These security risks highlight the urgent need for developing and integrating more effective privacy protection mechanisms.

In addition, FL also faces challenges related to aggregation integrity [13], [14]. In the standard centralized FL architecture, the role of the aggregator is pivotal, but this can also make it a potential single point of failure. This vulnerability becomes particularly apparent in the real world, where the aggregator may be operated by untrusted organizations. In the absence of robust guarantees for aggregation integrity, a malicious aggregator could compromise the integrity of the global model. Such an entity might not only falsify the aggregated results but

This work was supported in part by the NSF of China under Grant 62372177.

Lulu Wang, Lei Zhang, and Lin Xu are with the Software Engineering Institute, East China Normal University, Shanghai 200062, China. Lei Zhang is the corresponding author.

Mirko Polato is with the Department of Computer Science, University of Turin, Turin 10124, Italy.

Alessandro Brighente and Mauro Conti are with the Department of Mathematics, University of Padua, Padua 35121, Italy.

could also manipulate the outcomes returned to participants, possibly driven by improper motives. For instance, within an FL task designed for image recognition, an unscrupulous aggregator could deliberately alter the aggregation process. This manipulation could lead to the misclassification of images or introduce biases against certain categories of images, potentially skewing the model's effectiveness and fairness. Similarly, in the context of financial analysis, an aggregator might tamper with the results to create an undue advantage in market transactions.

A. *Related Work*

Several FL schemes [15]–[29] have been proposed to tackle fundamental data privacy challenges. These schemes incorporate differential privacy (DP) or cryptographic tools like secret sharing and homomorphic encryption within secure multiparty computation (MPC) frameworks. Focusing first on solutions based on DP, the scheme proposed in [15] achieves a balance between privacy and usability in deep neural networks through selective parameter sharing and differential privacy. However, a subsequent study shows that adversaries can recover information about the training data from gradients [24]. Other schemes are [16]–[18], which integrate deep learning and DP to avoid privacy leakage. However, as the performance results show, the noise injected to get a differentially private solution seriously affects the model's accuracy. In scenarios such as ad recommendation or credit management, a small loss in accuracy greatly impacts results.

Turning to schemes based on secret sharing, in [19], the authors design a secure aggregation scheme combining secret sharing and key agreement, which requires four rounds of interactions to aggregate model updates from one iteration of training and it is only suitable for scenarios where the number of participants is greater than two. Other secret sharing-based schemes [20], [21] also exhibit similar challenges, including the need for multiple interaction rounds to facilitate decryption and mitigate the impact of users dropping out. Moreover, to decrease the frequency of these interactions, some schemes [22], [23] are compelled to employ a dual-server configuration. Besides, secret sharing can also be deployed in the inference phase [30]–[32], protecting the model privacy of the model provider and the data privacy of the model users [33], [34]. Despite its significant importance, privacy preservation during the inference phase is orthogonal to that in the training phase. We stress that these secret sharing-based privacy-preserving algorithms can be leveraged as a plug-in mechanism to further enhance the system privacy, which leaves further work.

As for the privacy-preserving approaches that utilize homomorphic encryption, schemes [24], [25] based on the Paillier homomorphic encryption (PHE) can protect data privacy but introduce significant computation and communication overhead. The approach proposed in [25] attempts to reduce the overhead of PHE through batch encryption and gradient clipping. However, this method may lead to a loss of accuracy, cannot resist collusion attacks, and still incurs substantial overhead. On the other hand, fully homomorphic encryption (FHE) technologies such as BFV [35], BGV [36], TFHE [37], and CKKS [38] are also employed to safeguard privacy in FL. Although TFHE is robust, it lacks support for packing encryption techniques, rendering it less suitable for processing large-scale parameters in FL. The CKKS-based solutions [26], [27] are specially designed to process floating-point numbers and support packaging encryption. However, they will introduce noise during decryption operations, thereby affecting model accuracy. In contrast, schemes [28], [29] based on BGV and BFV typically rely on integer operations but present higher computational complexities when handling packed data. Moreover, each encryption operation introduces noise, which especially accumulates during the model aggregation process when numerous participants are involved. Overall, despite the broader application possibilities offered by fully homomorphic encryption schemes, the Paillier scheme provides a simpler approach in terms of noise management and computational complexity. Most importantly, none of the mentioned schemes analyze the composition of parameters or assess whether each component is linked to privacy leakage. Additionally, they do not support verification of aggregate integrity.

More recently, some studies [13], [14], [39], [40] have attempted to guarantee aggregation integrity while preserving data privacy. Xu et al. [13] add the aggregation integrity verification based on the homomorphic hash function and the pseudo-random technologies proposed in [19], which can verify the correctness of the results returned by the aggregator while ensuring the confidentiality of the model updates. Guo et al. [14] further improve communication efficiency by designing a verification function whose overhead is independent of the dimension of the model updates. However, the above two schemes, like [19], require four rounds of interactions to aggregate model updates from one iteration of training, and participants need to synchronize secret keys and masks. VFL proposed in [39] implements verification through Lagrange interpolation polynomials. However, it requires that the central server and clients do not collude and that the x-coordinates of all interpolation points remain consistent. Jiang et al. [40] proposed PFLM, which achieves model aggregation verification through ElGamal encryption and identity-based aggregate signatures. Nonetheless, due to the ciphertext expansion problem, the communication overhead for system verification is linearly related to the dimension of the gradient vectors.

To our knowledge, existing privacy-preserving methods based on DP invariably lead to a reduction in model accuracy. Secret sharing approaches, while effective, introduce substantial communication overhead, particularly in scenarios with a large user base. Homomorphic encryption methods, though capable of addressing the drawbacks of both DP and secret sharing, still encounter significant computational costs. PriVeriFL offers a novel solution by assessing the sensitivity of individual bits within parameters and selectively encrypting only those that are sensitive, thereby markedly reducing the PHE workload. Furthermore, by employing batch encryption techniques, the data volume requiring encryption is reduced to less than 1% of its original size. Unlike existing aggregation integrity verification schemes, PriVeriFL utilizes linear homomorphic hashing and signature algorithms, making communication overhead independent of model dimensions and enabling integrity verification without additional interactions.

B. Our Contribution

In order to address the limitations in the existing literature, we analyze the sensitivity of different bits of model parameters, and propose PriVeriFL, a privacy-preserving and aggregation-verifiable FL scheme based on PHE, homomorphic hash function, and signature to ensure both data privacy and aggregation integrity. Our main contributions are as follows:

1) We analyze the sensitivity of different bits of model parameters by combining inference attacks and information theory. Specifically, we implement membership inference attacks and model inversion attacks on different bits of model parameters to verify whether they will leak model privacy. We utilize an information-theoretical model, specifically mutual information, to quantify the degree of information leakage about the input data leaked by the different bits of the model parameter. Our results demonstrate that exposing high bits of model parameters does not reveal sensitive information.
2) Inspired by the above realization, we propose a novel low-expansion homomorphic aggregation scheme based on PHE to protect the data privacy of participants. This scheme can effectively reduce the computation and communication overhead of PHE. Compared with the privacy-preserving FL scheme based on classic PHE, our scheme can reduce the communication overhead to 1.65%, and reduce the encryption and decryption calculation overhead to 0.88%.
3) We design PriVeriFL-A, a privacy-preserving and aggregation-verifiable FL scheme by combining the above homomorphic aggregation scheme, homomorphic hash function, and signature. PriVeriFL-A can effectively verify the integrity of the aggregation returned by the aggregator while protecting the privacy of participants' data. Moreover, the communication overhead of integrity verification is independent of the dimension of the model updates submitted by the participants, so the communication overhead of integrity verification is almost negligible.
4) We further improve the above PHE-based scheme into a threshold PHE-based FL scheme, named PriVeriFL-B, to further resist collusion attacks between the aggregator and participants. In PriVeriFL-B, even if the aggregator colludes with $t-1$ malicious participants, the plaintext model update of an honest participant will not be leaked, where t refers to the decryption threshold of the threshold PHE.

The rest of this paper is organized as follows. Section II presents the relevant background, followed by preliminary in Section III. In Section IV, we analyze the sensitivity of different bits of model parameters. Section V presents PriVeriFL-A, a privacy-preserving and aggregation-verifiable FL scheme. We further improve the scheme in Section VI and name it PriVeriFL-B to deal with possible collusion attacks. Section VII is the security analysis. The experimental results in Section VIII show the overhead of our scheme. Finally, Section IX concludes the paper.

II. Background

In this section, we provide our system architecture, threat model, and design goals.

A. System Architecture

In the following, we describe the architecture of PriVeriFL. As shown in Figure 1, the system consists of the following entities.

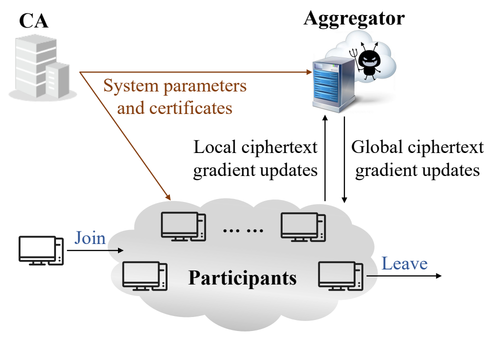

CA
Aggregator
System parameters and certificates
Local ciphertext gradient updates
Global ciphertext gradient updates
Join
Participants
Leave

Fig. 1: System architecture.

- *Certification Authority*: The Certification Authority (CA) is a trusted third party. It is used to generate the system parameters and issue certificates for the aggregator and participants.
- *Participants*: Each participant, denoted as P_i, has his own dataset D_i. A group of participants might decide to initialize an FL task for the training of a global model. The participants in the group will generate encrypted updates based on the latest global model and their respective datasets, and send the encrypted updates to the aggregator. Participants can join or leave the group at any moment during the overall learning procedure.
- *Aggregator*: The aggregator (Ag) is used to initialize the parameters of the global model. Further, with the encrypted updates from the participants in a group, it can aggregate encrypted updates into two encrypted updated global model shares which are used for the participants in the group to generate an updated global model.

B. Threat Model

As discussed earlier, common threats to FL come from attackers seeking to obtain local model updates as well as the aggregator compromising the integrity of aggregation. Threat summaries from different entities are described in the following.

- *Threats from participants*: Driven by interests of profit, even malicious participants need to perform local training correctly according to the protocol and send their models to the aggregator, to benefit from the final aggregated model. However, malicious participants may steal local model updates from other participants. Once malicious participants get model updates of specific participants, they can launch inference-based attacks [41], [42] to obtain information about the training data. Furthermore, malicious participants might also launch more covert backdoor attacks. In response, honest participants can implement backdoor detection after receiving the aggregated model, which is beyond the scope of the current discussion.

- *Threats from the aggregator*: The malicious aggregator attempts to obtain local model updates of specific participants to launch inference-based attacks. Additionally, the aggregator may be managed by an untrusted organization, causing it to deliberately manipulate the aggregation result. This could be done either to minimize its computational efforts–referred to as laziness–or to falsify the global model, undermining the integrity of the entire FL process.
- *Threats from outsiders*: An external attacker tries to obtain local model updates of specific participants by monitoring the communication between the participant and the aggregator to launch inference-based attacks.
- *Threats from colluding attackers*: Multiple malicious participants may collude, potentially with the malicious aggregator. Their goals are twofold: attacking specific honest participants to steal their local model updates, and falsifying the global model. The latter can occur in two ways. First, the aggregator might directly tamper with the aggregation result, which is our primary focus. Secondly, corrupted participants might introduce errors into their local models before aggregation, potentially reducing the global model's utility. We believe that even if such an attack occurs, honest participants can assess the aggregated model's quality by verifying its performance after an aggregation round. If the aggregated model fails to provide the expected performance improvement, honest participants may opt out to minimize the impact of the attack. For task sustainability, even malicious participants are expected to provide accurate local model training results.

C. Design Goals

Our design goals are twofold: first, to mitigate the threats previously discussed, and second, to minimize the computation and communication overhead resulting from the incorporation of homomorphic encryption in the FL system. The specific goals are as follows.

- *Efficient model update privacy:* If the privacy of a participant's model update is not protected, an attacker may launch inference-based attacks to extract sensitive information about the participant. Efficient and lossless model update privacy guarantees that 1) no one except the generator of a model update can learn the update, 2) even if a privacy mechanism is applied to protect the privacy of a model update, the efficiency of the scheme will not be significantly reduced.
- *Global model privacy:* It guarantees that, except for the global model initialized by the aggregator, only the participants of an FL task can learn the latest global model in each training round of the task.
- *Lossless:* The accuracy of the final global model will not be reduced compared with the final global model trained using the plain FL scheme.
- *Efficient integrity checking:* A malicious aggregator could dishonestly generate encrypted global model shares. Efficient integrity checking guarantees that 1) if encrypted global model shares corresponding to an FL task are dishonestly generated, then any participant of the task can efficiently find the misbehavior of the aggregator, 2) this property can be achieved with low message expansion.
- *Collusion resistance:* Obviously, the aggregator can get the global model corresponding to an FL task if it can collude with a participant involved in this task. Collusion resistance here means that even if $t-1$ participants in a set $\mathbb{P}$ collude with the aggregator, they cannot get the model update of a participant who is not in $\mathbb{P}$, where t is a threshold and defines the least number of participants needed to have a successful collusion. We note that all the participants in PriVeriFL-A share the same public-private key pair. In this setting, if a participant colludes with the aggregator, they can decrypt the model updates of all the participants. PriVeriFL-B resists this attack and achieves the property of collusion resistance.

III. Preliminary

To facilitate the understanding of PriVeriFL, we introduce the cryptographic primitives used and the background knowledge of FL in this section.

A. (Threshold) Paillier Homomorphic Encryption

Our scheme is based on Paillier homomorphic encryption (PHE) [43], which is an additive homomorphic encryption scheme. The functionality of PHE is denoted as $\mathcal{F}_{\text{PHE}}$. It consists of five algorithms PHE = (PHE.KeyGen, PHE.Enc, PHE.Dec, PHE.Add, PHE.Mul), as follows:

- PHE.KeyGen(ℓ): Define $L(x) = \frac{(x-1)}{n}$. Choose two large prime numbers p and q s.t. $\gcd(pq, (p-1)(q-1)) = 1$. Calculate $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$. Randomly generate $g \in \mathbb{Z}_{n^2}^*$ s.t. $\gcd(L(g^\lambda \mod n^2), n) = 1$. Output the public key $p_{\text{PHE}} = (n, g)$ and private key $s_{\text{PHE}} = \lambda$.
- PHE.Enc(p_{PHE}, m): On input a message $m \in \mathbb{Z}_n$, it generates a random number $r \in \mathbb{Z}_n^*$ and computes the ciphertext
$$c = g^m r^n \mod n^2.$$
- PHE.Dec(s_{PHE}, c) : On input a ciphertext c, it calculates
$$m = \frac{L(c^\lambda \mod n^2)}{L(g^\lambda \mod n^2)} \mod n.$$
- PHE.Add(c_1, c_2): On input two ciphertexts c_1, c_2 corresponding to plaintexts $m_1, m_2 \in \mathbb{Z}_n$ respectively, it computes
$$c_1 c_2 \mod n^2 = \text{PHE.Enc}(p_{\text{PHE}},\ (m_1 + m_2) \mod n).$$
- PHE.Mul(c, k): On input a ciphertext c corresponding to the plaintext m and $k \in \mathbb{Z}_n$, it satisfies
$$c^k \mod n^2 = \text{PHE.Enc}(p_{\text{PHE}},\ km \mod n).$$

In PriVeriFL-A, PHE is used to encrypt the model update of a participant. Further, all participants share the same public-private key pair. If a participant P_i gets an encrypted model update of another participant P_j, then the encrypted model update can be decrypted by P_i. Therefore, P_i could collude with the aggregator and hence obtain a model update of P_j. Threshold PHE (TPHE) may be applied to solve this problem [44]. A (t, n) TPHE scheme consists of five algorithms TPHE $=$ (TPHE.KeyGen, TPHE.Enc, TPHE.Dec, TPHE.Add, TPHE.Mul), in which the algorithms TPHE.Enc, TPHE.Add and TPHE.Mul are the same as the algorithms PHE.Enc, PHE.Add and PHE.Mul in PHE, respectively. The algorithms TPHE.KeyGen, TPHE.Dec are defined as follows:

- TPHE.KeyGen(ℓ): On input a security parameter ℓ, a group of participants $P_1, \ldots, P_n$ perform a (t, n) distributed key generation protocol, e.g., the protocol in [44], to generate a public key p_{TPHE} and each participant P_i's secret key share s^i_{TPHE}.
- TPHE.Dec$(\{s^i_{\text{TPHE}}\}_{|\{s^i_{\text{TPHE}}\}|\geq t},\ c)$: On input a ciphertext c, each participant decrypt c with his/her own secret key share s^i_{TPHE} and announces partial decryption result c^i_{TPHE}. Each participant combines the partial decryption results of t participants to get the plaintext.

B. Homomorphic Hash Function

Homomorphic hashing (HH) is a one-way, collision-resistant, and homomorphic function that supports computing the hash of a composite block based on the hashes of the individual blocks. We use $\mathcal{F}_{\text{HH}}$ to denote the functionality of HH. As constructed in [45], a homomorphic hash scheme consists of three algorithms H $=$ (H.Gen, H.Hash, H.Eval), as follows:

- H.Gen(ϑ, θ) : On input security parameters ϑ, θ, it generates public parameters p_{HH}, including cyclic group $\mathbb{G}$ of prime order p, a generator $g \in \mathbb{G}$, and θ distinct elements $g_1, \ldots, g_\theta \in \mathbb{G}$.
- H.Hash(m_i) : On input a θ-dimensional vector $m_i = [m_i[1], m_i[2], \ldots, m_i[\theta]]$, it outputs the hash value of the homomorphic hash function, i.e. $h_i \leftarrow \prod_{l=1}^{\theta} g_l^{m_i[l]}$, where $m_i[l] \in Z_p$.
- H.Eval$(h_1, \ldots, h_\xi, w_1, \ldots, w_\xi)$: The homomorphic hash function satisfies the property that for hashes $h_1, \ldots, h_\xi$ output by H.Hash and coefficients $w_1, \ldots, w_\xi \in Z_p$, it can compute their linear combination, i.e. $h^* \leftarrow \prod_{i=1}^{\xi} h_i^{w_i}$. Therefore, the size of the hash is independent of the dimension θ of the input vector m_i.

C. Model Average Algorithm

The federated averaging algorithm [3] is proposed for model training in federated learning, specifically aggregating multiple models into a global model. According to the different aggregation parameters, federated averaging algorithms can be divided into model averaging algorithms [46] and gradient averaging algorithms [47], [48]. Here we take the traditional model averaging algorithm as an example. As shown in Algorithm 1, multiple participants train models with local data and send model updates to the aggregator. Then, the aggregator aggregates the received model updates into a global model according to the model averaging algorithm. The model averaging algorithm mainly includes the following steps:

1) The aggregator initializes the training, that is, it randomly initializes the global model and broadcasts basic information about model training to all participants along with the model.
2) Each participant trains the model locally, gets the model update and sends it to the aggregator.
3) The aggregator computes a weighted average of all participants' models based on the size of every participant's local dataset, resulting in an updated global model.

The above steps (2) and (3) are terminated after being repeated T times or a convergence criterion is reached.

Our scheme achieves the same aggregation effect as the model averaging algorithm, but sends encrypted model updates during the aggregation process, thus protecting the data privacy of participants.

Algorithm 1 Model Averaging Algorithm

```
Require: Participants $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$, dataset $\mathcal{D} = \{D_1, D_2, .., D_N\}$, where $P_i \in \mathcal{P}$ holds dataset $D_i$ and the size of $D_i$ is $n_i$; $n$: sum of participant dataset size; $\digamma$: deep learning algorithm; $\varphi$: learning rate; $E$: number of loacl epochs; $T$: number of rounds.
Ensure: Global model $m_T$
 1: function AGGREGATOR EXECUTES:
 2:     Initialize $m_0$
 3:     for t=1 to $T$ do
 4:         for every participant $P_i \in \mathcal{P}$ in parallel do
 5:             $m_t^i \leftarrow$ LocalUpdate$(m_{t-1})$
 6:         end for
 7:         $m_t \leftarrow \sum_{i=1}^{N} \frac{n_i}{n} m_t^i$
 8:         return $m_t$ to all participants.
 9:     end for
10: end function
11: function LOCALUPDATE($i$, $m$):
12:     for iteration=1 to $E$ do
13:         for each batch $\boldsymbol{b}$ in participant $P_i$'s split do
14:             $m \leftarrow m - \varphi \cdot \bigtriangledown \digamma(m, \boldsymbol{b})$
15:         end for
16:     end for
17:     return $m$ to the aggregator
18: end function
```

IV. Sensitivity Analysis for Model Parameters

Prior to designing a privacy-preserving FL scheme, it is important to determine whether it is necessary to encrypt all the bits of the model. One way to analyze this is by examining the composition of the model parameters. Typically, a deep learning model trained by participants consists of several matrices, each comprising multiple floating-point numbers with eight decimal places (e.g., 0.12345678, and we represent 1 to 8 as from low bit to high bit). Our intuition is that different bits of the model parameters exhibit varying sensitivity, meaning that they carry different amounts of information about the model training data.

To verify this intuition, we conducted membership inference attacks and model inversion attacks on different bits of model parameters and evaluated their sensitivity based on the results of these attacks. It is important to note that while these attacks are common and effective, they do not provide a rigorous proof of the sensitivity of different bits of model parameters.

Therefore, we utilized mutual information from information theory to quantify the amount of information leaked about the model training data by different bits of model parameters. This approach enabled us to determine the sensitivity of each bit of the model parameters.

A. *Membership Inference Attacks*

Membership inference attacks [41] against machine learning models aim to determine whether a target data sample is used to train the target model. Formally speaking, given a data sample x and the access to the target model M, determine whether x belongs to the training set of model M. The disclosure of identity information will seriously threaten the privacy of users. For example, if a model is trained on patient-specific disease data, judging that a person is included in the training set can reveal that person's health status, which seriously violates personal privacy.

We first introduce the background knowledge and main idea of the attack. Both knowledges of the target model and training data are crucial for an attacker to implement membership inference attacks effectively. In our experiment, we describe the adversary's knowledge with the most common setting that fits our scheme, i.e., the adversary has black-box access to the target model, and masters the training data distribution to generate a shadow dataset with the same distribution. We adopt the membership inference attacks scheme proposed in [41]. The attacker trains the shadow model using part of the shadow dataset and then queries the shadow model with the entire shadow dataset to obtain prediction vectors $pred$. If a sample of the shadow dataset belongs to the training set of the shadow model, the adversary labels it $member$, otherwise, label it as $non\text{-}member$. Taking the shadow dataset labels y, prediction vectors $pred$, and membership set $\{member/non\text{-}member\}$ tuple $(y, pred, \{member/non\text{-}member\})$ as a new training set, the attacker uses it to train a binary classification attack model. After that, in order to confirm whether a target sample belongs to the training set of the target model, it can be fed to the target model to obtain the prediction vector, and then the label of the sample and the prediction vector can be fed to the attack model.

We train the classification model using the convolutional neural network ResNet-18 [49] on the CIFAR100 [50] dataset. CIFAR100 contains 100 classes of images, each class contains 600 images, divided into 500 training images and 100 testing images. We choose PyTorch [51] as the experimental platform, cross-entropy as the loss function, and SGD as the optimizer with a momentum of 0.9. The learning rate is 0.01, and the training epochs are 40.

First, we consider the relationship between attack accuracy and the training dataset size of the target model. As shown in Figure 2, we select 10000, 30000, and 50000 pieces of training data to train the target model, and then conduct membership inference attacks on it. Result shows that the accuracy of the attack decreases as the target model training dataset size increases.

Then we verify the impact of different bits of model parameters on the attack accuracy to analyze the sensitivity of different bits of model parameters. We replace the 8th bit, 8th to 7th bits, ..., and the 8th to the 1st bits after the decimal point of all parameters with 0, respectively. Figure 2 shows that the attack against the original model and the model in which the 8th to 3rd bits of all parameters are replaced with 0 is effective. When the 2nd and 1st bits are replaced, the attack accuracy drops to the baseline 0.5. This means that from the perspective of membership inference attacks, the 1st and 2nd bits after the decimal point of model parameters are sensitive, while the 3rd to 8th digits are insensitive because the presence or absence of this part does not greatly affect the attack result.

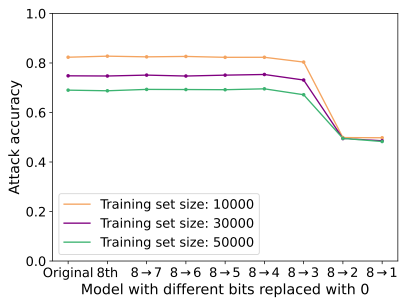


Fig. 2: Accuracy of the membership inference attacks against the original model and the model in which different bits after the decimal point of all parameters are replaced by 0.

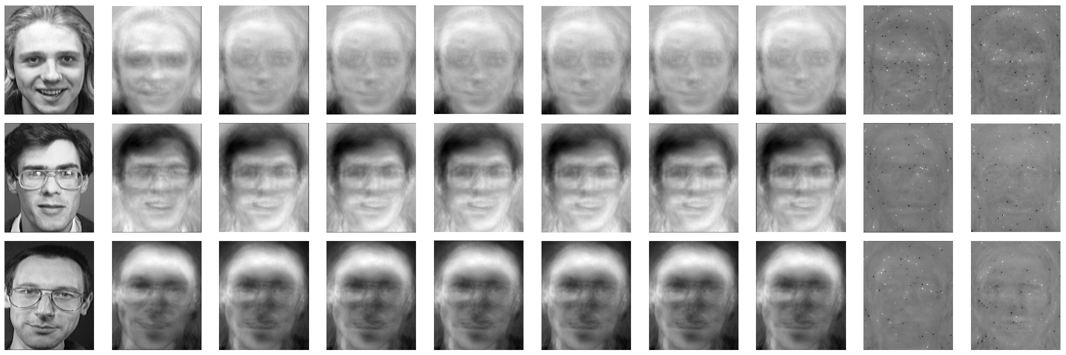
B. *Model Inversion Attacks*

Model inversion attacks [42] against machine learning models aim to reconstruct information about the training data from the target model. Formally, given the dimension and label of the target sample, and the access of the target model M, reconstruct the representative data of this class of sample. Model inversion attacks will indirectly leak victim privacy. For example, in a neural network-based face recognition system, the adversary can recover the victim's face data contained in the target model's training set.

Again, we first introduce the background knowledge and idea of this attack. In the experiments, we adopt the most common setting for the adversary's knowledge, that is, the adversary has access to the target model M and grasps the dimension and label of the target sample. We use the model inversion attacks scheme proposed in [42]. The attacker generates a random sample x according to the dimension of the target sample and feeds it into the target model M to calculate the loss value. The attacker then optimizes the random sample x by using a backpropagation algorithm based on the model parameters. This process is repeated until the loss value reaches the threshold.

We train the classification model using a two-layer neural network on the AT&T Laboratories Cambridge database of faces[1]. This dataset contains 40 classes of images, each class contains 10 images, divided into 7 training images and 3 testing images. We choose PyTorch as the experimental platform, Negative Log-Likelihood as the loss function, and Adam as the optimizer. The learning rate is 0.0003 and the training epochs are 500.

[1]http://cam-orl.co.uk/facedatabase.html



(a) Training (b) Original (c) 8th (d) $8 \rightarrow 7$ (e) $8 \rightarrow 6$ (f) $8 \rightarrow 5$ (g) $8 \rightarrow 4$ (h) $8 \rightarrow 3$ (i) $8 \rightarrow 2$ (j) $8 \rightarrow 1$

Fig. 3: Model inversion attacks against the original model and the model in which different bits after the decimal point of all parameters are replaced by 0.

TABLE I: MI and NMI of the original model and the model in which different bits after the decimal point of all parameters are replaced by 0.

		Original	8th	$8 \rightarrow 7$	$8 \rightarrow 6$	$8 \rightarrow 5$	$8 \rightarrow 4$	$8 \rightarrow 3$	$8 \rightarrow 2$	$8 \rightarrow 1$
CIFAR100	MI	3.867	3.867	3.867	3.867	3.867	3.867	3.861	0.469	0.0
	NMI	0.999	0.999	0.999	0.999	0.999	0.999	0.997	0.210	0.0
AT&T Face	MI	3.678	3.678	3.678	3.678	3.678	3.678	3.619	0.494	0.0
	NMI	0.997	0.997	0.997	0.997	0.997	0.997	0.981	0.207	0.0

We perform the model inversion attacks on the above-trained model. As shown in Figure 3(b), the adversary can reconstruct the victims' images after obtaining the necessary information, which seriously violates the victims' privacy.

Then we verify the impact of different bits of model parameters on the attack result to analyze the sensitivity of different bits. As in the membership inference attack, we replace the 8th bit, 8th to 7th bits, ..., and the 8th to the 1st bits after the decimal point of all parameters with 0, respectively. It can be seen from Figures 3(c) to 3(h) that the absence of the 8th to 3rd bits of the model parameters does not affect the attack results. And when the 2nd and 1st bits are replaced, the reconstructed images (Refer to Figures 3(i) and 3(j)) basically cannot display effective information about the victims, which means that from the perspective of model inversion attacks, the first two bits after the decimal point of the model parameter are highly sensitive, while other positions are non-sensitive.

C. Information Theoretic Analysis

Information theory is a mathematical theory that deals with the quantification and communication of information. It offers a mathematical framework for the examination of information, encompassing its quantification. One of the central tenets of information theory is mutual information (MI) [52], which assesses the degree of information that two sets of cluster share.

MI measures the reduction in uncertainty about one cluster that can be achieved by knowing the other. MI between X and Y is a non-negative. A high value of MI indicates a strong association between X and Y, while a low value indicates little or no association. More formally, given two clusters X and Y, MI between them is defined as:

$$\mathrm{MI}(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)},$$

where $P(x)$ and $P(y)$ are the marginal probabilities of X and Y, respectively, and $P(x,y)$ is the joint probability of X and Y.

Normalized Mutual Information (NMI) is a variation of MI, which takes into account the sizes of the clusters and normalizes the MI value to a range between 0 (no mutual information) and 1 (perfect correlation). NMI between two sets of cluster labels X and Y is given by:

$$\mathrm{NMI}(X;Y) = \frac{2 \cdot \mathrm{MI}(X;Y)}{-\sum_{x \in X} P(x) \log P(x) - \sum_{y \in Y} P(y) \log P(y)}.$$

From the perspective of inference attacks, the aim is to infer the membership attribute of the data or reproduce the original data based on the labels. We consider the labels as sensitive information that needs to be protected. Subsequently, we modify certain bits of the model parameters and calculate MI and NMI between the predicted labels and the true labels of the modified model. This analysis allows us to evaluate the sensitivity of different bits of the model parameters.

We analyze the sensitivity of different bits of the model parameters with the target models used in Sections IV-A and IV-B. As shown in Table I, the predicted labels and the true labels of the original target model are highly correlated. We perform a bit replacement process on various parameters of two models: one trained on the CIFAR100 dataset, used for membership inference attacks, and the other trained on the AT&T Laboratories Cambridge database of faces, used for model inversion attacks. During this process, we replace the 8th bit, the 8th to 7th bits, and so on, up to the 8th to 1st bits after the decimal point, with 0 for each parameter. Upon replacing the 2nd and 1st bits of all parameters with 0, we observe that the MI and NMI scores between the predicted labels and true labels become close to 0. This finding sheds light on the reasons behind the success of the membership inference attacks and model inversion attacks.

We need to point out that although the above method of gradually replacing the 8th bit, 8th to 7th bits, ..., and the 8th to the 1st bits can reflect the sensitivity of each bit, it cannot fully capture the synergistic effect between bits, especially the

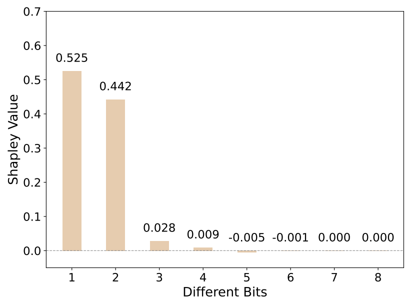


Fig. 4: Shapley value of the normalized mutual information of each bit.

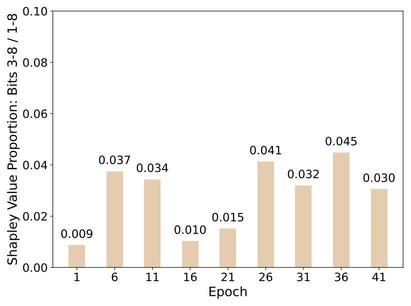


Fig. 5: The proportion of the sum of Shapley values for bits 3-8 as a function of the training epoch.

interaction between non-adjacent bits. The Shapley value [53], [54] of NMI is a promising tool that quantifies the contribution of each bit across all possible combinations. This method not only assesses the impact of a single bit but also captures their synergistic effects, thus providing a more comprehensive understanding of bit sensitivity. We implement this by considering all 256 combinations of the 8 bits post-decimal and calculating the NMI between the model's predicted labels and the actual labels, from which we derive the Shapley values for each bit.

Our calculations of the Shapley value of NMI are performed on models trained on the CIFAR100 and AT&T Face datasets. Given the similarity in Shapley values between the two models, we only present the analysis of the ResNet-18 model trained on CIFAR100. As shown in Figure 4, the Shapley values of the 1st and 2nd bits post-decimal are significantly higher than those of the other bits, indicating that they contribute substantially to model performance, carrying the bulk of the model's critical information. The 3rd and 4th bits show smaller Shapley values, indicating lesser contributions. Notably, the Shapley values of the 5th and 6th bits are negative, suggesting that these bits might introduce noise or redundant information in certain combinations, thus degrading model performance. Although these bits may have a certain effect in specific cases, they could disrupt model predictions in others. The Shapley values of the 7th and 8th bits are almost 0, indicating minimal to no impact on model performance.

To verify the robustness of the above analysis, we also explore the potential variability in the sensitivity of insensitive bits as the model's accuracy changes (reflected through training epochs). To assess the overall sensitivity/contribution of the aforementioned non-sensitive bits, we still use the Shapley value as a measurement tool. The Shapley value, which quantifies the contribution or sensitivity of each bit, is determined by calculating its marginal contribution across all possible combinations. Thus, by summing the Shapley values of several bits and dividing this sum by the total Shapley values of all bits, we can gauge the proportionate contribution of these bits to the model. To assess the sensitivity changes in bits beyond the most significant (i.e., bits 3 to 8), we calculated the proportion of the sum of the Shapley values for bits 3 to 8 over the sum of the Shapley values for all bits across various training epochs. In our research, compared to the two-layer neural network model trained on the AT&T Face dataset, the ResNet-18 trained on the CIFAR100 dataset presents a higher training challenge and can better reflect the changes in the training accuracy curve. We select this model for detailed analysis. As illustrated in Figure 5, the proportion of the sum of the Shapley values for 3-8 bits does not show significant variation with the increase in training epochs (i.e., as the training accuracy curve flattens) and consistently remains below 5%. Regardless of the model's accuracy level, the primary information is still represented by the 1st and 2nd bits. This demonstrates that even as the accuracy curve tends to be flat, the most significant bits (lower bits) are still the main information carriers of the model, while the sensitivity of other bits changes minimally.

We conclude that the success of inference attacks depends on the accuracy of model predictions. However, the sensitivities of different bits of the model parameters are inconsistent. By protecting the low bits, which carry most of the model information, the prediction results become invalid for attackers, preventing successful inference attacks. Thus, it is only necessary to implement privacy protection for the first sensitive t bits after the decimal point of the model. To generalize the above conclusions to different privacy-preserving tasks, we develop an adaptive method to provide appropriate bit sensitivity analysis for various tasks, which is particularly crucial for special cases. For example, when the first two bits of the model parameter are almost all 0, it may be necessary to consider sensitivity from the third bit. To accommodate varying privacy protection tasks, model providers could calculate the Shapley values of NMI for different bits to determine the primary sensitive bits. When the sum of the Shapley values from the 1st bit to the t-th bit exceeds a set threshold (such as 95%), these bits can be regarded as sensitive bits. Based on our experience, $t=2$ or $t=3$ should suffice for most scenarios. This finding suggests that selectively encrypting highly sensitive bits of parameters in privacy-preserving FL schemes can significantly reduce computation and communication overhead, without compromising the privacy of the training data.

V. Proposed Framework

PriVeriFL-A comprises the system setup, and training and aggregation stages. Below is a detailed description of each stage.

A. System Setup

Suppose each participant has a public-private key pair and a certificate issued by a CA. Then, the CA initializes the

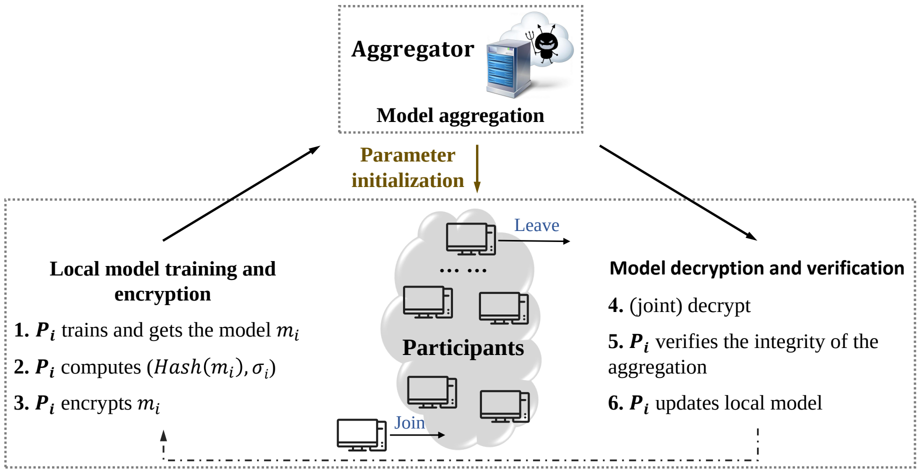
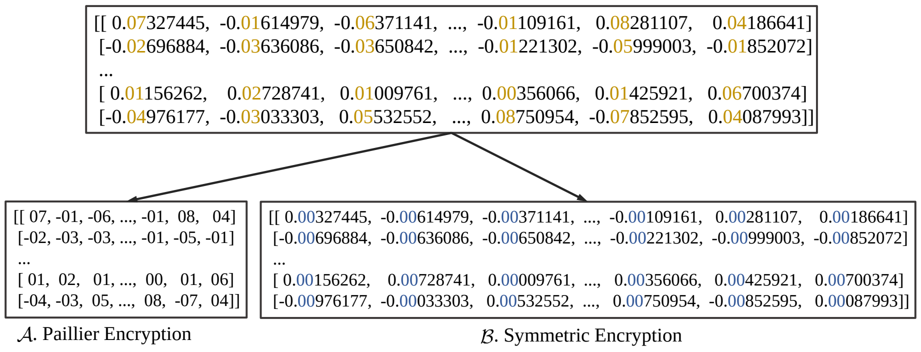

Aggregator
Model aggregation
Parameter initialization
Local model training and encryption
1. P_i trains and gets the model m_i
2. P_i computes $(Hash(m_i), \sigma_i)$
3. P_i encrypts m_i
Leave
Participants
Join
Model decryption and verification
4. (joint) decrypt
5. P_i verifies the integrity of the aggregation
6. P_i updates local model

Fig. 6: Training and aggregation stage, which includes four sub-stages, namely: parameter initialization, local model training and encryption, model aggregation, model decryption and verification.



```
[[ 0.07327445,  -0.01614979,  -0.06371141,  ...,  -0.01109161,   0.08281107,   0.04186641]
 [-0.02696884,  -0.03636086,  -0.03650842,  ...,  -0.01221302,  -0.05999003,  -0.01852072]
 ...
 [ 0.01156262,   0.02728741,   0.01009761,  ...,   0.00356066,   0.01425921,   0.06700374]
 [-0.04976177,  -0.03033303,   0.05532552,  ...,   0.08750954,  -0.07852595,   0.04087993]]
```

```
[[ 07, -01, -06, ..., -01,  08,  04]
 [-02, -03, -03, ..., -01, -05, -01]
 ...
 [ 01,  02,  01, ...,  00,  01,  06]
 [-04, -03,  05, ...,  08, -07,  04]]
```

```
[[ 0.00327445,  -0.00614979,  -0.00371141,  ...,  -0.00109161,   0.00281107,   0.00186641]
 [-0.00696884,  -0.00636086,  -0.00650842,  ...,  -0.00221302,  -0.00999003,  -0.00852072]
 ...
 [ 0.00156262,   0.00728741,   0.00009761,  ...,   0.00356066,   0.00425921,   0.00700374]
 [-0.00976177,  -0.00033303,   0.00532552,  ...,   0.00750954,  -0.00852595,   0.00087993]]
```

$\mathcal{A}$. Paillier Encryption
$\mathcal{B}$. Symmetric Encryption

Fig. 7: Model segmentation.

hash function (see Section III-B) and publishes the basic parameters to all participants. That is, all participants have the same hash function security parameters ϑ. Moreover, the aggregator uses any symmetric encryption (SE) scheme SE $= (\text{SE.Gen}(\tau), \text{SE.Enc}(k, m), \text{SE.Dec}(k, c))$ [55] to establish a secure channel with participants, where k is the symmetric key, m is the plaintext and the c is the ciphertext. As for participants, they use PHE scheme with the same public-private key pairs $(p_{\text{PHE}}, s_{\text{PHE}})$ to protect local model updates. Additionally, each participant has a public-private key pair (mpk_i, msk_i) of the signature scheme.

B. Training and Aggregation

As shown in Figure 6, this stage can be divided into four sub-stages, namely: parameter initialization, local model training and encryption, model aggregation, model decryption and verification.

1) Parameter initialization: The initialization of the participants should be completed before each round of training. The aggregator sends the initial neural network model to the participants, and specifies the corresponding loss function, optimization function, learning rate, and other necessary parameters for the training.

2) Local model training and encryption: Each participant will execute the TRAIN&ENCRYPT function in Algorithm 2 (see lines 1 to 14). Participant P_i trains his/her local model m_i by multiple iterations on the dataset D_i. In order to ensure the correctness of the global model M sent by the aggregator, each participant P_i needs to use the homomorphic hash function (see Section III-B) to generate the hash value h_i of the local model update m_i, and finally verify whether the global model is calculated correctly through all the hash values $\{h_i\}_{1 \le i \le N}$ of particpants' local model updates. The specific process of calculating the hash value is as follows:

- When the participant P_i gets the trained model update m_i, he/she multiplies each parameter in the m_i by 10^8 to get an new model m_i' (here it is assumed that the parameters of model update m_i are kept to eight decimal places).
- Participant P_i calculates and obtains the hash value h_i of the model m_i' by computing $\text{H.Hash}(m_i')$, and generates the signature σ_i of the hash value h_i through the signature scheme $\text{Sig}(msk_i, h_i)$. At the same time, P_i generates the signature ϱ_i of the local dataset size n_i through the signature scheme $\text{Sig}(msk_i, n_i)$.

In order to protect the privacy of the model update, each participant combines the Paillier homomorphic encryption algorithm and the symmetric encryption algorithm to encrypt the local model update and then send it to the aggregator for aggregation. As shown in Figure 7, participant P_i splits the parameters in the model update m_i into two parts, extracts and saves the t decimal places of all parameters into matrix $\mathcal{A}_i$ (Here take t=2 as an example), fills the extracted part with 0, and saves the filled parameters to matrix $\mathcal{B}_i$. Each participant will use the PHE scheme to encrypt the data in matrix $\mathcal{A}_i$, and use the symmetric encryption scheme to encrypt the data in matrix $\mathcal{B}_i$. The specific process is as follows:

Algorithm 2 Training, aggregation and verification.

```
Require: Participants $\mathcal{P} = \{P_1, \ldots, P_N\}$, dataset $\mathcal{D} = \{D_1, \ldots, D_N\}$, where $P_i \in \mathcal{P}$ holds his/her own dataset $D_i$ and the size of dataset $D_i$ is $n_i$; $\digamma$: deep learning algorithm; $(p_{\text{PHE}}, s_{\text{PHE}})$: Paillier based public-private key pair of all participants; $(mpk_i, msk_i)$: The public-private key pair of each participant's signature scheme; H: Hash function.
Ensure: Global model $\mathcal{M}$.
 1: function TRAIN&ENCRYPT:
 2:     for $P_i \in \mathcal{P}$ do
 3:         $m_i \leftarrow \digamma(D_i)$
 4:         $m_i' \leftarrow m_i \cdot 10^8$
 5:         $h_i \leftarrow$ H.Hash$(m_i')$
 6:         $\sigma_i \leftarrow$ Sig$(msk_i, h_i)$
 7:         $\varrho_i \leftarrow$ Sig$(msk_i, n_i)$
 8:         $\mathcal{A}_i, \mathcal{B}_i \leftarrow$split$(m_i)$
 9:         $\{v_{ij}\}_{1 \le j \le \eta} \leftarrow$encode and pack$(\mathcal{A}_i)$
10:         $\{\hat{v}_{ij}\}_{1 \le j \le \eta} \leftarrow \{$PHE.Enc$(p_{\text{PHE}}, v_{ij})\}_{1 \le j \le \eta}$
11:         $\hat{\mathcal{B}}_i \leftarrow$ SE.Enc$(k, \mathcal{B}_i)$
12:         send $(\{\hat{v}_{ij}\}_{1 \le j \le \eta}, \hat{\mathcal{B}}_i, h_i, \sigma_i, n_i, \varrho_i)$ to Aggregator
13:     end for
14: end function
15: function AGGREGATE:
16:     Get $\{\{\hat{v}_{ij}\}_{1 \le j \le \eta}, \hat{\mathcal{B}}_i, h_i, \sigma_i, n_i, \varrho_i\}_{1 \le i \le N}$
17:     $\{\{\hat{v}_{ij}^{w}\}_{1 \le j \le \eta}\}_{1 \le i \le N} \leftarrow$
18:         $\{\{$PHE.Mul$(\hat{v}_{ij}, n_i)\}_{1 \le j \le \eta}\}_{1 \le i \le N}$
19:     $\{\hat{V}_j\}_{1 \le j \le \eta} \leftarrow \{\sum_{i=1}^{N} \hat{v}_{ij}^{w}\}_{1 \le j \le \eta}$
20:     $\{\mathcal{B}_i\}_{1 \le i \le N} \leftarrow \{$SE.Dec$(k, \hat{\mathcal{B}}_i)\}_{1 \le i \le N}$
21:     $\{\mathcal{B}_i{}^{w}\}_{1 \le i \le N} \leftarrow \{\mathcal{B}_i * n_i\}_{1 \le i \le N}$
22:     $M_{\mathcal{B}} \leftarrow \sum_{i=1}^{N} \mathcal{B}_i{}^{w}$
23:     Broadcast $\{h_i, \sigma_i, n_i, \varrho_i\}_{1 \le i \le N \& z_i = 1}, n, M_{\mathcal{B}}$,
24:         $\{\hat{V}_j\}_{1 \le j \le \eta}$ to all participants
25: end function
26: function DECRYPT&VERIFY:
27:     Get $\{h_i, \sigma_i, n_i, \varrho_i\}_{1 \le i \le N \& z_i = 1}, n, M_{\mathcal{B}}, \{\hat{V}_j\}_{1 \le j \le \eta}$
28:     for $P_i \in \mathcal{P}$ do
29:         if $Ver(mpk_i, \sigma_i)$ or $Ver(mpk_i, \varrho_i) = \perp$ then
30:             Terminate this round of training
31:         end if
32:     end for
33:     $M_{\mathcal{A}} \leftarrow$decode$(\{$PHE.Dec$(s_{\text{PHE}}, \hat{V}_j)\}_{1 \le j \le \eta})$
34:     Global model $M \leftarrow$combine$(M_{\mathcal{A}}, M_{\mathcal{B}})$
35:     $M' \leftarrow M \times 10^8$
36:     if H.Hash$(M') =$ H.Eval$(h_1, \ldots, h_i, n_1, \ldots, n_i)$ then
37:         Globao model $M \leftarrow \frac{M}{n}$
38:     end if
39: end function
```

- In the PHE scheme, the ciphertext length is roughly the same as the key length, and as of 2019, the minimum secure key size for Paillier is 2048 [56]. If Paillier homomorphic encryption is performed on each element in matrix $\mathcal{A}_i$ separately, it will cause dozens of times of ciphertext expansion. In order to avoid redundancy, when the plaintext is short and of fixed length, we make full use of the plaintext space and pack multiple plaintexts into one for encryption and decryption calculations.
- Participant P_i converts each parameter a_i in matrix $\mathcal{A}_i$ into a two's complement b_i of length l bits with two sign bits. Take $\rho = \lfloor 2048/(l+2) \rfloor$ two's complement as a group, and concatenate padding values and a group of two's complement as shown in Figure 8 to obtain a large integer $v = 00\|b_1\|00\|b_2\| \ldots \|00\|b_\rho$. The padding value is used here to avoid overflow, and two sign bits are used to distinguish positive overflow and negative overflow in subsequent calculations. We assume that the total number of parameters in matrix $\mathcal{A}_i$ is Q, and so does matrix $\mathcal{B}_i$. Set $\eta = \lceil Q/\rho \rceil$, the parameters in the matrix $\mathcal{A}_i$ are divided into η groups with the number ρ as a group (the number of last group is $Q - \rho * (\eta - 1)$), and are respectively packed into large integers v_{i1}, v_{i2}, …,$v_{i\eta}$ as previously described.

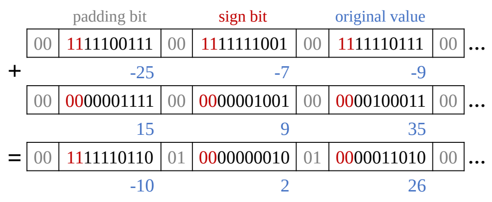


Fig. 8: BatchCrypt.

- Participant P_i encrypts each large integers v_{ij} $(1 \le j \le \eta)$ by computing PHE.Enc$(p_{\text{PHE}},\ v_{ij})$ of the PHE scheme to obtain PHE-based ciphertext $\hat{v}_{ij}$, and encrypts matrix $\mathcal{B}_i$ by computing SE.Enc$(k,\ \mathcal{B}_i)$ of the symmetric encryption scheme to obtain the SE-based ciphertext $\hat{\mathcal{B}}_i$.

Finally, participant P_i sends $\{\hat{v}_{ij}\}_{1 \le j \le \eta}$, $\hat{\mathcal{B}}$, hash and signature pair (h_i, σ_i), and local dataset size and signature pair (n_i, ϱ_i) to the aggregator.

3) Model aggregation: If either the aggregator receives model-related information $\{\{\hat{v}_{ij}\}_{1 \le j \le \eta}, \hat{\mathcal{B}}, h_i, \sigma_i, n_i, \varrho_i\}_{1 \le i \le N}$ from all participants, or the maximum waiting time of aggregator is reached, the aggregator executes the *Aggregate* function (see lines 15 to 25). The aggregator generates a vector Z to indicate which model-related information of participants it has received, and it can be set as $Z = (z_1, z_2, \ldots, z_N) = (1, 0, \ldots, 1)$, s.t. $|Z| = N$, where N is the total number of participants. The aggregator may not receive some participants' model-related information due to network delay, participants dropping out, etc. If the aggregator has received the model-related information submitted by participant P_i , $z_i = 1$, otherwise $z_i = 0$. Therefore, the impact of some participants leaving the system is small, which enables dynamic participation. Then, the aggregator calculates the sum of the dataset sizes of the participants included in vector Z and denotes it by n. It needs to calculate the weight of each participant based on his/her dataset size, and sets $weight = (n_1, \ldots, n_N)$, where $n_i > 0$ if $z_i = 1$ and $n_i = 0$ otherwise. Now the aggregator needs to aggregate the encrypted model updates of the participants included in Z, which is done as follows:

- For the $\{\hat{v}_{ij}\}_{1 \le j \le \eta}$ sent by participant P_i, the aggregator multiplies each element $\hat{v}_{ij}$ in it by the corresponding weight n_i by computing PHE.Mul$(\hat{v}_{ij}, n_i)$ to obtain the *weighted* PHE-based ciphertext $\{\hat{v}_{ij}^{w}\}_{1 \le j \le \eta}$. The aggregator performs this calculation for the PHE-based ciphertexts of all participants in vector Z.
- Next, the aggregator adds each element $\hat{v}_{ij}^{w}$ $(1 \le j \le \eta,\ 2 \le$

$i \le N$) of participants P_2 to P_N with corresponding element $\hat{v}^w_{1j}(1 \le j \le \eta)$ of participant P_1 by computing $\hat{v}^w_{1j} =$ PHE.Add$(\hat{v}^w_{1j}, \hat{v}^w_{ij})$, thereby aggregating the weighted PHE-based ciphertexts of all participants to obtain $\{\hat{v}^w_{1j}\}_{1\le j\le\eta}$ (it will be denoted as $\{\hat{V}_j\}_{1\le j\le\eta}$ later).

- The aggregator decrypts all participants' SE-based ciphertext $\{\hat{\mathcal{B}}_i\}_{1\le i\le N}$ by computing SE.Dec$(k, \hat{\mathcal{B}}_i)$ to obtain $\{\mathcal{B}_i\}_{1\le i\le N}$. Then, it uses the model average algorithm mentioned in Section III-C to aggregate $\{\mathcal{B}_i\}_{1\le i\le N}$ with weight $n_i \ne 0$.
- Specifically, the aggregator multiplies each element b_j $(1 \le j \le \mathcal{Q})$ in $\mathcal{B}_i$ by the corresponding weight n_i to obtain the *weighted* ${\mathcal{B}_i}^w$. It performs the above for all participants to obtain $\{{\mathcal{B}_i}^w\}_{1\le i\le N}$. Next, it aggregates $\{{\mathcal{B}_i}^w\}_{1\le i\le N}$ to obtain $M_{\mathcal{B}}$.

Finally, the aggregator sends $\{h_i, \sigma_i, n_i, \varrho_i\}_{1\le i\le N \& z_i=1}, n, M_{\mathcal{B}}, \{\hat{V}_j\}_{1\le j\le\eta}$ to all participants.

4) Model decryption and verification: When the participant P_i receives $\{h_i, \sigma_i, n_i\}_{1\le i\le N\&z_i=1}, M_{\mathcal{B}}$, and $\{\hat{V}_j\}_{1\le j\le\eta}$ returned by the aggregator, he/she will execute the DECRYPT&VERIFY function (see lines 26 to 39 of Algorithm 2). He/She first determines whether the hash values and dataset size of all participants' model updates have been tampered by computing Ver$(mpk_j, \sigma_j)_{1\le j\le N}$ or Ver$(mpk_j, \varrho_j)_{1\le j\le N} \overset{?}{=} \perp$. If it is confirmed that the hash value or dataset size of one participant model update has been tampered, P_i will terminate the current round of training and aggregation. Otherwise, he/she decrypts and decodes $(\{\hat{V}_j\}_{1\le j\le\eta}, M_{\mathcal{B}})$ to obtain the global model M.

Specifically, participant P_i decrypts each element $\hat{V}_j$ in $\{\hat{V}_j\}_{1\le j\le\eta}$ by computing PHE.Dec$(s_{\text{PHE}}, \hat{V}_j)$ to obtain $\{V_j\}_{1\le j\le\eta}$. According to the reverse operation in Figure 8, P_i divides each element in $\{V_j\}_{1\le j\le\eta}$ into ρ parts, each of which is $l+2$ bits in length, and removes the first two padding values in each part to obtain the matrix $M_{\mathcal{A}}$ as same as the matrix $\mathcal{A}$ in Figure 7. Then, according to the reverse operation in Figure 7, the participants combine the matrix $M_{\mathcal{A}}$ and the matrix $M_{\mathcal{B}}$ into a global model M.

Finally, participant P_i need to verify the integrity of the global model M to determine whether the aggregator has tampered with the aggregation result. P_i multiplies each parameter in the M by 10^8 to get an new model M', and generates a hash value H.Hash(M') for the new model M'. Then P_i determines whether the result returned by the aggregator has been tampered with by calculating H.Hash$(M') \overset{?}{=}$ H.Eval$(h_1, \dots, h_i, n_1, \dots, n_i)_{1\le i\le N\&z_i=1}$. If the above equation does not hold, it can be determined that the aggregator maliciously forged the aggregation results. Otherwise, the global model $M = \frac{M}{n}$.

VI. Improved Framework

In PriVeriFL-A (see Section V), there may be a risk of privacy leakage due to the fact that some malicious participants and the aggregator launch collusion attacks to observe local model updates of some honest participants and launch inference attacks against these model updates. In the improved framework, we address this problem by enhancing the *system setup* proposed in Section V-A, and *training and aggregation* proposed in Section V-B, name it PriVeriFL-B.

The improved *system setup* is the same as that in Section V-A, with the exception that all participants execute the threshold Paillier homomorphic encryption distributed key generation protocol (see Section III-A) to generate the public key p_{TPHE} and the secret key share s^i_{TPHE} of each participant.

The improved *training and aggregation* stage is the same as that in Section V-B, except for the following sub-stage:

- In the *model decryption and verification* sub-stage, participant P_i decrypts $\{\hat{V}_j\}_{1\le j\le\eta}$ with his/her secret key share s^i_{TPHE}, and broadcast the partial decryption result. After receiving the partial decryption result sets $(|sets| \ge t-1)$ of other participants, P_i obtains the decrypted result $\{V_j\}_{1\le j\le\eta}$ by computing $\{\text{TPHE.Dec}(\{s^r_{\text{TPHE}}\}_{|\{s^r_{\text{TPHE}}\}|\ge t}, \hat{V}_j)\}_{1\le j\le\eta}$.

VII. Security Analysis

In this section, we first analyze how our scheme guarantees the confidentiality of each participant's local update. Then, we briefly describe the aggregation integrity against the malicious adversary. We formally prove the security of our schemes in the malicious setting using the simulation-based paradigm [57]. The designed security goals of our protocol are formally captured by an ideal functionality $\mathcal{F}$, where a trusted entity receives inputs from parties, performs the designed computation, and sends outputs to parties. In the real world, an adversary $\mathcal{A}$ will on behalf of the corrupted parties and run the protocol Π with honest parties. Then, a simulator $\mathcal{S}$ plays the same role as the corrupted parties and interacts with $\mathcal{F}$ in the ideal world to get the input/output results. So $\mathcal{S}$ can communicate with $\mathcal{A}$ in the real world to simulate the view of $\mathcal{A}$. The formal definition of malicious security is as follows.

Definition 1: A Protocol (or scheme) Π securely computes $\mathcal{F}$ in the presence of a malicious adversary if for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$, such that for every subset of corrupt parties $\mathcal{C}$, all inputs $x_1, \dots, x_n$, auxiliary input z and security parameters k:

$$\text{IDEAL}_{\mathcal{F},\mathcal{S}}(k, C, z; x_1, \dots, x_n)$$
$$\overset{c}{\equiv} \text{REAL}_{\pi,\mathcal{ADV}}(k, C, z; x_1, \dots, x_n).$$

We say that the protocol Π can securely compute the functionality $\mathcal{F}$ with a satistical error 2^{λ} and a negligible function negl$(\cdot)$ such that the probability that an adversary can distinguish the views of the ideal and real world is less than $2^{\lambda} + \text{negl}(\kappa)$.

Next, we need to prove our protocol $\Pi_{\text{PriVeriFL}}$ is secure as we claimed in the security goals. So we present the ideal functionality of our protocol $\mathcal{F}_{\text{PriVeriFL}}$ as shown in Figure 9. Specifically, $\Pi_{\text{PriVeriFL}}$ represents the PriVeriFL-B protocol, which has more functionality than PriVeriFL-A. If PriVeriFL-B has been proven to be secure, then the security of PriVeriFL-A is guaranteed.

Theorem 1: Under the $(\mathcal{F}_{\text{PHE}}, \mathcal{F}_{\text{HH}})$-hybrid model, the protocol $\Pi_{\text{PriVeriFL}}$ implements $\mathcal{F}_{\text{PriVeriFL}}$ correctly and securely in the presence of malicious adversaries.

$\mathcal{F}_{\text{PriVeriFL}}$

Parameters: For each party P_i, it has a database D_i and a local trained model m_i. An aggregated result $\mathcal{M}$. **PriVeriFL:**

1. On receiving inputs $(\text{PriVeriFL}, m_i)$ from P_i, the functionality outputs the encrypted model $\{\hat{v}_{ij}\}_{1\le j\le\eta}$, $\hat{\mathcal{B}}_i$, H.Hash (m_i), σ_i, n_i, ϱ_i $\}_{1\le i\le N}$ to participant P_i if it does not abort;
2. On receiving (Aggregation, $\{\{\hat{v}_{ij}\}_{1\le j\le\eta}$, $\hat{\mathcal{B}}_i\}_{1\le i\le N}$) from aggregator Ag, the functionality outputs the aggregated values $\{\hat{V}_j\}_{1\le j\le\eta}$ and $M_{\mathcal{B}}$ to all parties if it does not abort;
3. On receiving $(\text{Decryption \& Verification}, \text{M}_{\mathcal{B}}$, $\{\hat{V}_j\}_{1\le j\le\eta})$ from P_i, the functionality verifies the aggregated values and sends the global model M to parties. If the verification process fails, $\perp$ is output to parties.

Fig. 9: Ideal functionality $\mathcal{F}_{\text{PriVeriFL}}$.

Proof: **Participant is Corrupted.** We can construct an ideal world simulator $\mathcal{S}_{\text{pa}}$ to simulate the view of a participant.

1) $\mathcal{S}_{\text{pa}}$ invokes the real-world adversary $\mathcal{A}_{\text{pa}}$ with input model m and is given the public parameters p_{PHE}, p_{HH}, mpk.
2) $\mathcal{S}_{\text{pa}}$ is given the encrypted model $\{\hat{v}_j\}_{1\le j\le\eta}$ and records the results $\hat{\mathcal{B}}$, $h = \text{Hash}(m)$, σ, n, ϱ.
3) $\mathcal{S}_{\text{pa}}$ invokes the adversary $\mathcal{A}_{\text{pa}}$ with p_{PHE}, p_{HH}, mpk, $\{\hat{v}_j\}_{1\le j\le\eta}$, $\hat{\mathcal{B}}$, h, σ, n, ϱ.
4) $\mathcal{S}_{\text{pa}}$ receives a verification call from the adversary $\mathcal{A}_{\text{pa}}$, and generates Ver(mpk,σ), Ver(mpk,ϱ), and returns the results of Ver(mpk,σ), Ver(mpk,ϱ) to $\mathcal{A}_{\text{pa}}$.

As we can see, in this simulation, the view of the adversary in the real execution is the same as that in the simulation, because it has been proved that the results of PHE are random and indistinguishable when different values are input.

Aggregator is Corrupted. We can construct an ideal world simulator $\mathcal{S}_{\text{ag}}$ to simulate the view of an aggregator, and the adversary is denoted as $\mathcal{A}_{\text{ag}}$.

1) $\mathcal{S}_{\text{ag}}$ is given the encrypted results $\{\hat{v}_j\}_{1\le j\le\eta}$, $\hat{\mathcal{B}}$, $h = \text{Hash}(m)$, σ, n, ϱ.
2) $\mathcal{S}_{\text{ag}}$ invokes the adversary $\mathcal{A}_{\text{ag}}$ with $\{\hat{v}_j\}_{1\le j\le\eta}$, $\hat{\mathcal{B}}$, $h = \text{Hash}(m)$, σ, n, ϱ, and receives an aggregation call from the adversary.
3) $\mathcal{S}_{\text{ag}}$ invokes $\mathcal{F}_{\text{PHE}}$ to compute an encrypted aggregated model $\{\hat{V}_j\}_{1\le j\le\eta}$ and $M_{\mathcal{B}}$, then sends them to $\mathcal{A}_{\text{ag}}$.

As we can see, in this simulation, the view of the adversary in the real execution is the same as that in the simulation, because it has been proved that the results of HH and PHE are random and indistinguishable when different values are input.

Collusion of Aggregator and $t-1$ Participants. We can construct an ideal world simulator $\mathcal{S}'$ to simulate the views of an aggregator and corrupted participants. The adversaries are denoted as $\mathcal{A}'_{\text{pa}}$ and $\mathcal{A}'_{\text{ag}}$, and the set of corrupted participants is denoted as $\mathcal{C}$.

1) $\mathcal{S}'$ invokes the real-world adversary $\mathcal{A}'_{\text{pa}}$ with input models $\{m_i\}_{i\in\mathcal{C}}$ and is given the public parameters p_{PHE}, p_{HH}, $\{mpk_i\}_{i\in\mathcal{C}}$.
2) $\mathcal{S}'$ receives an invocation to $\mathcal{F}_{\text{PHE}}$ and $\mathcal{F}_{\text{HH}}$, and is given the results $\{\{\hat{v}_{ij}\}_{1\le j\le\eta}$, $\hat{\mathcal{B}}_i$, $h_i = \text{Hash}(m_i)$, σ_i, n_i, $\varrho_i\}_{i\in\mathcal{C}}$.
3) $\mathcal{S}'$ invokes the adversary $\mathcal{A}'_{\text{pa}}$ with p_{PHE}, p_{HH}, $\{mpk_i\}_{i\in\mathcal{C}}$ and $\{\{\hat{v}_{ij}\}_{1\le j\le\eta}$, $\hat{\mathcal{B}}_i$, $h_i = \text{Hash}(m_i)$, σ_i, n_i, $\varrho_i\}_{i\in\mathcal{C}}$.
4) $\mathcal{S}'$ invokes the adversary $\mathcal{A}'_{\text{ag}}$ with $\{\{\hat{v}_{ij}\}_{1\le j\le\eta}$, $\hat{\mathcal{B}}_i$, $h_i = \text{Hash}(m_i)$, σ_i, n_i, $\varrho_i\}_{i\in\mathcal{C}}$, and receives an aggregation call from $\mathcal{A}'_{\text{ag}}$.
5) $\mathcal{S}'$ invokes $\mathcal{F}_{\text{PHE}}$ to compute an encrypted aggregated model $\{\hat{V}_j\}_{1\le j\le\eta}$ and $M_{\mathcal{B}}$, then and sends them to $\mathcal{A}'_{\text{ag}}$ and $\mathcal{A}'_{\text{pa}}$.
6) $\mathcal{S}'$ invokes $\mathcal{F}_{\text{PHE}}$ and $\mathcal{F}_{\text{HH}}$ to do a verification on all inputs receives from last steps. If the verification fails, $\mathcal{S}'$ sends an abort to $\mathcal{F}$PriVeriFL and terminates the protocol execution with the adversary.

As we can see, in this simulation, the view of the adversary in the real execution is the same as that in the simulation, because it has been proved that the results of PHE and HH are random and indistinguishable when different values are input. Besides, less than t corrupted participants can not decrypt the results and change the verification results. After simulating all scenarios of different corruptions, we can prove that our protocol Π_{PriVerFL} is as secure as the ideal functionality $\mathcal{F}_{\text{PriVerFL}}$ defined in Fig. 9.

Aggregation Integrity Against Malicious Adversary. In the next theorem, we assert that integrity is attained where any adversary attempting to manipulate the aggregation result can be promptly detected once all participants have generated signatures of the homomorphic hash value and the weight.

Theorem 2 (Integrity of Aggregation, Against Malicious Aggregator): Let M be the aggregation result of the inputs of all participants that successfully send the message to the aggregator, and $\{h_i, n_i, \sigma_i, \varrho_i\}$ be the hashes, weights, and their signatures of these participants. We say that if the adversary cannot break the hash and signature scheme, the integrity of the aggregation result can be guaranteed.

Proof: During the collaborative process, each honest-but-curious participant undertakes a series of steps to ensure the security and authenticity of the model aggregation. Initially, before proceeding with encryption, each participant generates a cryptographic hash of the model. Subsequently, each of them create signatures denoted as σ_i, ϱ_i corresponding to the hash h_i and the associated weight n_i.

Upon reaching the aggregation phase, the aggregator assumes the responsibility of aggregating the models and subsequently delivering all associated messages. It is important to emphasize that, concurrently, each individual participant carries out a comprehensive verification procedure. This verification involves confirming the authenticity of the hashes and weights by all other participants and verifying the integrity of the aggregation result by calculating H.Eval$(h_1,\ldots,h_i,n_1,\ldots,n_i)$. Should the aggregator engage in tampering with the aggregation result, it is noteworthy that this manipulation can solely extend to the forging signatures of hash and weight attributed to a specific participant. Importantly, the security of the signature scheme employed in this context renders malicious forgery an implausible endeavor.

TABLE II: Comparison of computation and communication overhead between PHE scheme and our scheme.

		Plaintext Size	Ciphertext Size	Encode	Encryption	Decryption	$C * P^{\dagger}$	$C_1 + C_2$	Decode
PHE	1 Thread	3.815MB	493.353MB	–	8317.736s	2373.330s	41.874s	28.276s	–
	16 Threads	3.815MB	493.353MB	–	1193.921s	357.837s	35.920s	35.060s	–
PriVeriFL-A	1 Thread	3.815MB	4.319+3.815MB[§]	3.873s	72.831s	20.847s	0.362s	0.245s	3.114s
	16 Threads	3.815MB	4.319+3.815MB[§]	1.376s	9.708s	2.942s	0.577s	0.570s	0.678s
PriVeriFL-B [ℵ]	1 Thread	3.815MB	4.32*20+3.815MB[§]	3.892s	73.847s	858.241s	0.375s	0.241s	3.241s
	16 Threads	3.815MB	4.32*20+3.815MB[§]	1.355s	9.596s	127.563s	0.594s	0.568s	0.655s

[†] Here C represents a ciphertext and P represents a constant.
[§] The ciphertext in our scheme consists of two parts, namely the homomorphically encrypted ciphertext of matrix $\mathcal{A}$ and the symmetrically encrypted ciphertext of matrix $\mathcal{B}$ in Figure 7.
[ℵ] The number of participants is 120 and the decryption threshold is 20.

VIII. Performance Evaluation

In this section, we begin by systematically analyzing the experimental performance of our scheme. We then proceed to compare it with the state-of-the-art scheme.

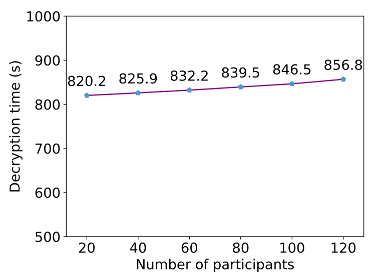
A. Experimental Performance

The experiments were performed on a PC with 11th Gen Intel(R) Core(TM) i7-11700K @ 3.60GHz, 16G RAM, 500G SSD, 2T HDD, and NVIDIA Quadro P5000 graphics card. We evaluate the overhead of PriVeriFL-A, the threshold generation and threshold decryption overhead of PriVeriFL-B, and the overhead of the integrity verification of the aggregated results in both schemes respectively. Specifically, we first compare PriVeriFL-A with standard PHE-based scheme in terms of computation and communication overhead to demonstrate the advantages brought by PriVeriFL-A. We then evaluate the factors that affect the time required to generate the threshold for PriVeriFL-B. Then, we assess the overhead of PriVeriFL-B to complete the threshold decryption. Finally, we evaluate the overhead required for participants to complete the integrity verification of aggregated results.

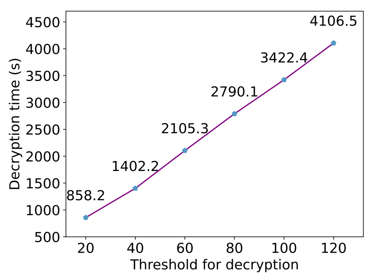
First, we calculate the computation and communication overhead of PriVeriFL-A and PriVeriFL-B, and compare them with the classic PHE-based scheme. In subsequent experiments, we set the key size for Paillier to the current minimum security key size of 2048 [56]. We encrypt and decrypt 1000K 32-bit floating point numbers using the classical PHE scheme implemented by python-Paillier-library [58] and PriVeriFL-A. At the same time, in order to reflect the impact of the number of threads on the encryption and decryption time, we also use 1 thread and 16 threads to complete the above encryption and decryption tests. As shown in Table II, compared with the classic PHE-based scheme, our ciphertext overhead is reduced to 1.65%. Under the 1-thread setting, our encryption time and decryption time are both reduced to 0.88%. Under the 16-thread setting, our encryption time is reduced to 0.81%, and our decryption time is reduced to 0.82%. In addition, we also verify the calculation times of PHE.Mul(C, P) and PHE.Add(C_1, C_2). Under the 1-thread setting, the time required by our scheme is reduced to 0.86% and 0.87%, respectively. Under the 16-thread setting, the time required by our scheme is reduced to 1.61% and 1.63% respectively. PriVeriFL-A only needs to introduce additional low-cost encoding and decoding operations, which can greatly reduce the amount of data that needs to be encrypted, thereby reducing the computation and communication overhead. In our analysis of PriVeriFL-B, we set the system with 120 participants and the decryption threshold to 20. Our comparative study reveals that relative to PriVeriFL-A, the size of the homomorphically encrypted ciphertext and the decryption time in PriVeriFL-B both increase linearly as the decryption threshold is raised. Despite these increases, PriVeriFL-B continues to maintain a high level of performance across other metrics, comparable to that of PriVeriFL-A.

Next, to evaluate the overhead of generating distributed keys in PriVeriFL-B, we implement distributed key generation using Paillier Threshold Encryption Toolbox[2]. We first consider the relationship between the time required to generate distributed keys and the number of participants. Since the distributed key generation phase involves random selection and checking processing, the time required is not constant. We, therefore, repeat the experiment 50 times to calculate the mean and standard deviation of the times required to generate the distributed keys. We set the number of participants to 20, 40, 60, 80, 100, and 120, respectively, and fix the decryption threshold to 20. As shown in Figure 10(a), the average time to generate distributed keys grows slowly as the number of participants grows. At the same time, we consider the relationship between the time required to generate the distributed keys and the decryption threshold when the number of participants is fixed. We fix the number of participants to 120, and the decryption threshold to 20, 40, 60, 80, 100, and 120, respectively. As shown in Figure 10(b), changing the decryption threshold has little effect on the time required to generate the distributed keys when the total number of participants is fixed. That is, the average time to generate a distributed key is only positively related to the total number of participants.

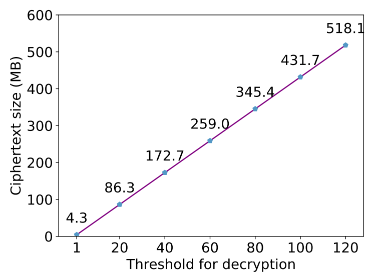
[2]https://cs.utdallas.edu/dspl/cgi-bin/Pailliertoolbox/



(a) Decryption time as a function of the number of participants, with a decryption threshold of 20 and a plaintext size of 3.815 MB.

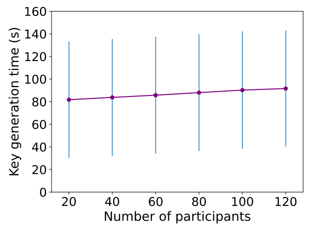


(b) Decryption time as a function of the decryption threshold, with 120 participants and a plaintext size of 3.815 MB.

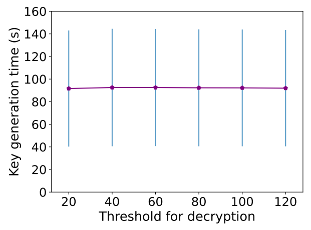


(c) Ciphertext size as a function of the decryption threshold, with 120 participants and a plaintext size of 3.815 MB.

Fig. 11: The decryption time and ciphertext size of threshold Paillier.

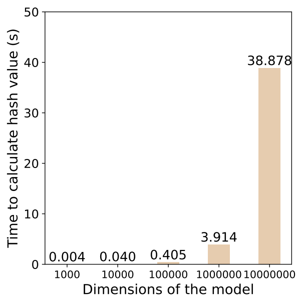


(a) Key generation time as a function of the number of participants, where the threshold for decryption is 20.

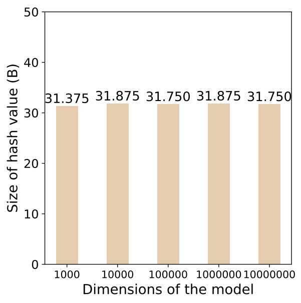


(b) Key generation time as a function of the threshold for decryption, where the number of participants is 120.

Fig. 10: The key generation time of threshold Paillier.



(a) The time cost of calculating the hash value.



(b) The size of the hash value.

Fig. 12: The overhead of computing hash values with different dimensions.

Then, we also evaluate the overhead of threshold decryption. In order to reflect the change in the required time to complete decryption from PriVeriFL-A to PriVeriFL-B (that is, from PHE to TPHE), we calculate how long it takes for the PriVeriFL-B to decrypt the data that can be decrypted by PriVeriFL-A in 20.847s under the 1-thread setting. We first consider the relationship between the required time for threshold decryption and the number of participants. We set the number of participants to 20, 40, 60, 80, 100, and 120, respectively, and fix the decryption threshold at 20. As shown in Figure 11(a), the time required for threshold decryption will increase slightly with the increase in the number of participants. At the same time, we consider the relationship between the required time for threshold decryption and the decryption threshold when the number of participants is fixed. We set the number of participants to 120, and the decryption thresholds to 20, 40, 60, 80, 100, and 120, respectively. As shown in Figure 11(b), the time required for threshold decryption increases linearly with the increase of the decryption threshold.

At the same time, we evaluate the relationship between the additional communication overhead caused by threshold decryption and the decryption threshold. We fix the number of participants to 120 and set the decryption thresholds to 20, 40, 60, 80, 100, and 120, respectively. As shown in Figure 11(c), as the decryption threshold increases, the threshold decryption communication overhead also increases linearly. When the decryption threshold is 1, the threshold decryption communication overhead is exactly the same as that of PriVeriFL-A.

Finally, we evaluate the overhead of computing model hash value for participants to verify the integrity of aggregation. We choose Charm [59] as the development platform and implement the homomorphic hash function based on the NIST P-256 curve. We need to consider the relationship between the time required to compute the hash value and the model dimension, and the relationship between the size of the hash value and the model dimension. We set the model dimensions to $10^3, 10^4, 10^5, 10^6$, and 10^7, respectively, and calculate the hash values of models with different dimensions. As shown in Figure 12, as the model dimension grows exponentially, so does the time required to compute the hash value, while the size of the hash value barely changes. Therefore, the computation overhead of integrity verification in PriVeriFL is related to the model dimension, while the communication overhead is fixed and almost negligible.

B. *Comparison*

In the realm of privacy-preserving federated learning, BatchCrypt [25] stands as a leading scheme, primarily due to its use of parameter quantization and packed encryption, which notably lower both computational and communication overhead. To assess the effectiveness of our scheme, PriVeriFL, we perform a thorough comparison with BatchCrypt across key metrics: accuracy, computational overhead, communication overhead, data privacy, and aggregation integrity.

When it comes to accuracy, BatchCrypt uses gradient quantization and pruning to compress encrypted data, which leads to some loss in accuracy. In contrast, PriVeriFL preserves the model's accuracy without any degradation. When compared to traditional PHE, BatchCrypt reduces computational overhead to 4.29% and communication overhead to 1.52% on the FM-NIST dataset, while capping accuracy loss at 1%. PriVeriFL, however, employs sensitivity analysis of model parameters to selectively encrypt only the most sensitive bits, reducing computational overhead to 0.88% and communication overhead to 1.65%, without compromising accuracy. While both BatchCrypt and PriVeriFL prioritize data privacy, PriVeriFL takes security a step further by incorporating aggregation integrity verification. This feature ensures protection against maliciously altered aggregation results by the aggregator, providing an additional layer of security for each client.

IX. Conclusion

We combine inference attacks and information theory to analyze the sensitivity of different bits of model parameters. Based on the analysis results, we clarify that not all bits of model parameters will leak privacy. This inspires us to propose a privacy-preserving and aggregation-verifiable federated learning scheme, which can protect the data privacy of participants and verify the integrity of aggregation returned by the aggregator. We further improve the scheme to resist possible collusion attacks. Our scheme dramatically reduces the computation and communication overhead caused by the introduction of homomorphic encryption, and the overhead caused by integrity verification is almost negligible.

References

[1] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, vol. 10, p. 3152676, 2017.

[2] L. de la Torre, "A guide to the california consumer privacy act of 2018," *Available at SSRN 3275571*, 2018.

[3] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, 2017, pp. 1273–1282.

[4] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[5] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[6] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[7] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan *et al.*, "Towards federated learning at scale: System design," *Proceedings of machine learning and systems*, vol. 1, pp. 374–388, 2019.

[9] E. Cordis, "Machine learning ledger orchestration for drug discovery," 2019.

[10] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.

[11] C. Song and V. Shmatikov, "Overlearning reveals sensitive attributes," in *ICLR*, 2020.

[12] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *IEEE Symposium on Security and Privacy*. IEEE, 2019, pp. 691–706.

[13] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2019.

[14] X. Guo, Z. Liu, J. Li, J. Gao, B. Hou, C. Dong, and T. Baker, "VeriFl: Communication-efficient and fast verifiable aggregation for federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1736–1751, 2020.

[15] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.

[16] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[17] Y. Gao, L. Zhang, L. Wang, K.-K. R. Choo, and R. Zhang, "Privacy-preserving and reliable decentralized federated learning," *IEEE Transactions on Services Computing*, vol. 16, no. 4, pp. 2879–2891, 2023.

[18] R. Liu, Y. Cao, H. Chen, R. Guo, and M. Yoshikawa, "FLAME: Differentially private federated learning in the shuffle model," in *AAAI*, vol. 35, no. 10, 2021, pp. 8688–8696.

[19] 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 the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[20] P. Xu, M. Hu, T. Chen, W. Wang, and H. Jin, "LaF: Lattice-based and communication-efficient federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2483–2496, 2022.

[21] C. Wu, L. Zhang, L. Xu, K.-K. R. Choo, and L. Zhong, "Privacy-preserving serverless federated learning scheme for internet of things," *IEEE Internet of Things Journal*, 2024.

[22] Z. Zhang, L. Wu, C. Ma, J. Li, J. Wang, Q. Wang, and S. Yu, "LSFL: A lightweight and secure federated learning scheme for edge computing," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 365–379, 2022.

[23] L. Zhong, L. Wang, L. Zhang, J. Domingo-Ferrer, L. Xu, C. Wu, and R. Zhang, "Dual-server based lightweight privacy-preserving federated learning," *IEEE Transactions on Network and Service Management*, 2024.

[24] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[25] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "BatchCrypt: Efficient homomorphic encryption for Cross-Silo federated learning," in *2020 USENIX Annual Technical Conference*, 2020, pp. 493–506.

[26] X. Hao, C. Lin, W. Dong, X. Huang, and H. Xiong, "Robust and secure federated learning against hybrid attacks: a generic architecture," *IEEE Transactions on Information Forensics and Security*, 2023.

[27] Y. Miao, Z. Liu, H. Li, K.-K. R. Choo, and R. H. Deng, "Privacy-preserving byzantine-robust federated learning via blockchain systems," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2848–2861, 2022.

[28] M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu, "Efficient and privacy-enhanced federated learning for industrial artificial intelligence," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, pp. 6532–6542, 2019.

[29] Y. M. Saputra, D. N. Nguyen, D. T. Hoang, Q.-V. Pham, E. Dutkiewicz, and W.-J. Hwang, "Federated learning framework with straggling mitigation and privacy-awareness for ai-based mobile application services," *IEEE Transactions on Mobile Computing*, vol. 22, no. 9, pp. 5296–5312, 2022.

[30] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "Deep learning-based medical diagnostic services: A secure, lightweight, and accurate realization 1," *Journal of Computer Security*, vol. 30, no. 6, pp. 795–827, 2022.

[31] ——, "MediSC: Towards secure and lightweight deep learning as a medical diagnostic service," in *26th European Symposium on Research in Computer Security*. Springer, 2021, pp. 519–541.

[32] ——, "Securely outsourcing neural network inference to the cloud with lightweight techniques," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 620–636, 2022.

[33] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX security symposium*, 2018, pp. 1651–1669.

[34] Q. Zhang, C. Xin, and H. Wu, "GALA: Greedy computation for linear algebra in privacy-preserved neural networks," *arXiv preprint arXiv:2105.01827*, 2021.

[35] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM Journal on computing*, vol. 43, no. 2, pp. 831–871, 2014.

[36] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[37] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[38] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *23rd International Confer-*

ence on the Theory and Applications of Cryptology and Information Security*. Springer, 2017, pp. 409–437.

[39] A. Fu, X. Zhang, N. Xiong, Y. Gao, H. Wang, and J. Zhang, "VFL: A verifiable federated learning with privacy-preserving for big data in industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3316–3326, 2020.

[40] C. Jiang, C. Xu, and Y. Zhang, "PFLM: Privacy-preserving federated learning with membership proof," *Information Sciences*, vol. 576, pp. 288–311, 2021.

[41] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy*. IEEE, 2017, pp. 3–18.

[42] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1322–1333.

[43] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[44] T. Nishide and K. Sakurai, "Distributed paillier cryptosystem without trusted dealer," in *International Workshop on Information Security Applications*. Springer, 2010, pp. 44–60.

[45] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography: The case of hashing and signing," in *Advances in Cryptology-CRYPTO'94: 14th Annual International Cryptology Conference Santa Barbara*. Springer, 1994, pp. 216–233.

[46] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *arXiv preprint arXiv:1602.05629*, vol. 2, 2016.

[47] H. Su and H. Chen, "Experiments on parallel training of deep neural network using model averaging," *arXiv preprint arXiv:1507.01239*, 2015.

[48] C. Yu, H. Tang, C. Renggli, S. Kassing, A. Singla, D. Alistarh, C. Zhang, and J. Liu, "Distributed learning over unreliable networks," in *ICML*. PMLR, 2019, pp. 7202–7212.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[50] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[51] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[52] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical review E*, vol. 69, no. 6, p. 066138, 2004.

[53] H. W. Kuhn and A. W. Tucker, *Contributions to the Theory of Games*. Princeton University Press, 1953, no. 28.

[54] M. Sundararajan and A. Najmi, "The many shapley values for model explanation," in *International conference on machine learning*. PMLR, 2020, pp. 9269–9278.

[55] J. Daemen and V. Rijmen, *The design of Rijndael*. Springer, 2002, vol. 2.

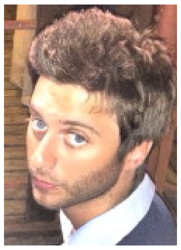
[56] E. Barker, E. Barker, W. Burr, W. Polk, M. Smid *et al.*, *Recommendation for key management: Part 1: General*. National Institute of Standards and Technology, Technology Administration, 2006.

[57] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pp. 277–346, 2017.

[58] CSIRO's Data61, "Python paillier library," https://github.com/data61/python-paillier, 2013.

[59] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.

Lulu Wang is currently pursuing his Ph.D. degree with the School of Software Engineering, East China Normal University, Shanghai, China. He was a Visiting PhD Student with SPRITZ Security and Privacy Research Group, Department of Mathematics, University of Padova, Italy. Since January 2024, he has been a visiting scholar with the Information Systems Technology and Design Pillar, Singapore University of Science and Technology, Singapore. His current research interests are at the intersection of security, privacy, and machine learning.



Mirko Polato received his Ph.D. in Brain, Mind, and Computer Science from the University of Padova (Italy) in 2018. He was Post-doc at the University of Padova and he is currently an Assistant Professor at the University of Turin (Italy). His research interests include federated learning, recommender systems, kernel methods, and machine learning in general.

Alessandro Brighente is assistant professor at the University of Padova. He received his Ph.D. degree in Information Engineering from the University of Padova in Feb. 2021. He was visiting researcher at Nokia Bell Labs, Stuttgart and University of Washington, Seattle in 2019 and 2022, respectively. He has been involved in European projects and industrial projects with the University of Padova. He served as TPC for several conferences, including Globecom and VTC. He is guest editor for IEEE Transactions on Industrial Informatics. His current research interests include security and privacy in cyber-physical systems, vehicular networks, blockchain, and physical layer security.

Mauro Conti is Full Professor at the University of Padua, Italy. He is also affiliated with TU Delft and University of Washington, Seattle. He obtained his Ph.D. from Sapienza University of Rome, Italy, in 2009. After his Ph.D., he was a Post-Doc Researcher at Vrije Universiteit Amsterdam, The Netherlands. In 2011 he joined as Assistant Professor at the University of Padua, where he became Associate Professor in 2015, and Full Professor in 2018. He has been Visiting Researcher at GMU, UCLA, UCI, TU Darmstadt, UF, and FIU. He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a Fellowship by the German DAAD (2013). His research is also funded by companies, including Cisco, Intel, and Huawei. His main research interest is in the area of Security and Privacy. In this area, he published more than 600 papers in topmost international peer-reviewed journals and conferences. He is Editor-in-Chief for IEEE Transactions on Information Forensics and Security, Area Editor-in-Chief for IEEE Communications Surveys & Tutorials, and has been Associate Editor for several journals, including IEEE Communications Surveys & Tutorials, IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Information Forensics and Security, and IEEE Transactions on Network and Service Management. He was Program Chair for TRUST 2015, ICISS 2016, WiSec 2017, ACNS 2020, CANS 2021, CSS 2021, WiMob 2023 and ESORICS 2023, and General Chair for SecureComm 2012, SACMAT 2013, NSS 2021, ACNS 2022, RAID 2024, NDSS 2026 and 2027. He is Fellow of the IEEE, Fellow of the AAIA, Distinguished Member of the ACM, and Fellow of the Young Academy of Europe.

Lei Zhang received the Ph.D. degree in computer engineering from Universitat Rovira i Virgili, Tarragona, Spain. Since then, he has been with Universitat Rovira i Virgili, as a Postdoctoral Researcher. He is currently a Full Professor with the School of Software Engineering, East China Normal University, Shanghai, China. He has been a holder/coholder of more than ten China/Spain-funded (key) projects. His fields of activity are information security, VANET security, cloud security, data privacy, and network security. He has authored over 80 publications. He has served in the program committee of more than 70 international conferences in information security and privacy.

Lin Xu received the Master degree in computer science and technology from Shandong Normal University, Jinan, China. She is currently pursuing her Ph.D. degree with the School of Software Engineering, East China Normal University, Shanghai, China. Her current research interests include information security, multi-party computation and privacy-preserving machine learning.