Hindawi
Security and Communication Networks
Volume 2022, Article ID 8380239, 10 pages
https://doi.org/10.1155/2022/8380239

*Research Article*

# A Homomorphic Signcryption-Based Privacy Preserving Federated Learning Framework for IoTs

**Weidong Du**, [1,2] **Min Li**, [1] **Yiliang Han**, [2] **Xu An Wang**, [2] **and Zhaoying Wei** [3]

[1] *Xi'an Hi-Tech Research Institute, Xi'an 710025, China*
[2] *College of Cryptography, Engineering University of PAP, Xi'an 710086, China*
[3] *College of Science, Xi'an Shiyou University, Xi'an 710065, China*

Correspondence should be addressed to Min Li; proflimin@163.com

Received 23 May 2022; Accepted 17 August 2022; Published 22 September 2022

Academic Editor: Chen Chen

Federated learning (FL) enables clients to train a machine learning model collaboratively by just aggregating their model parameters, which makes it very useful in empowering the IoTs with intelligence. To prevent privacy information leakage from parameters during aggregation, many FL frameworks use homomorphic encryption to protect client's parameters. However, a secure federated learning framework should not only protect privacy of the parameters but also guarantee integrity of the aggregated results. In this paper, we propose an efficient homomorphic signcryption framework that can encrypt and sign the parameters in one go. According to the additive homomorphic property of our framework, it allows aggregating the signcryptions of parameters securely. Thus, our framework can both verify the integrity of the aggregated results and protect the privacy of the parameters. Moreover, we employ the blinding technique to resist collusion attacks between internal curious clients and the server and leverage the Chinese Remainder Theorem to improve efficiency. Finally, we simulate our framework in FedML. Extensive experimental results on four benchmark datasets demonstrate that our framework can protect privacy without compromising model performance, and our framework is more efficient than similar frameworks.

## 1. Introduction

Traditional machine learning (ML) uses huge amounts of data collected from various sources to train models. However, data sharing from different devices or organizations may disclose privacy information about the owners. To solve the dilemma between protecting data privacy and leveraging the AI benefits to these data sensitive domains, federated learning (FL) [1, 2] is proposed to train ML models without sharing data directly. Because FL can protect the privacy while utilizing data, it has great application prospect in many scenarios [3–6], especially in IoTs [7, 8].

FL enables devices to collaboratively build a global ML model by only aggregating their model parameters with their data kept at their local storage. However, it still has the problem of privacy leakage. Existing researches [9–11] revealed that the exposed gradients still retain sensitive information about the training data. To avoid exposing gradients of a client during the training process, many privacy preserving frameworks based on cryptographic techniques have been proposed. Because homomorphic encryption (HE) allows aggregating the gradients through encryption envelope, privacy preserving FL frameworks based on HE have aroused many researchers' interests [9, 12–14].

However, because HE consists of complex cryptographic operations, these frameworks always bring heavy overhead. On the other hand, all clients in these frameworks share the same key pairs. If the server colludes with any client to get the secret key, the privacy protections will be invalid. Apart from efficiency and collusion attack issues, most privacy preserving FL frameworks ignore verifying the integrity of the aggregated results. However, a malicious or compromised server may forge the aggregated results to seduce clients to expose more sensitive information. Therefore, a privacy preserving FL framework should consider efficiency, privacy, and integrity.

Table 1: Comparison of PPFLS.

| PPFL | Updates privacy | Model privacy | Collusion resistant | Verifiability | Model supported |
|---|---|---|---|---|---|
| Shokri & Shmatikov | No | No | No | No | Linear & deep model |
| Geyer et al. | No | No | No | No | Linear & deep model |
| Bonawitz et al. | Yes | No | Yes | No | Linear & deep model |
| Phong et al. | Yes | Yes | No | No | Linear & deep model |
| Batchcrypt | Yes | Yes | Yes | No | Linear & deep model |
| Ma et al. | Yes | Yes | No | Yes | Linear & deep model |
| Zheng et al. | Yes | No | Yes | Yes | Linear model |
| Xu et al. | Yes | No | Yes | Yes | Linear & deep model |
| CRT-Paillier | Yes | Yes | No | Yes | Linear & deep model |
| VFL | Yes | Yes | Yes | Yes | Linear & deep model |
| Our framework | Yes | Yes | Yes | Yes | Linear & deep model |

We note that, though our framework has the same property as VFL, our framework is more efficient both in computation and in communication, which will be proved in Section 5.

This paper aims to address these problems by designing an efficient homomorphic signcryption based privacy preserving framework for FL in IoTs. We compare our framework with others in Table 1. Our contributions are summarized as follows:

(i) We design a homomorphic signcryption mechanism to encrypt and sign the clients' gradients at the same time. According to its additive homomorphic property, our mechanism allows the server to aggregate the gradients and signatures securely and allows the clients to verify the integrity (correctness) of the aggregated results.

(ii) We employ the blinding technique to resist collusion attacks. Even if $n-2$ clients try to collude with the server, they cannot steal privacy information of other clients' parameters.

(iii) We combine the clipping and quantizing technique in Batchcrypt [13] and the Chinese Remainder Theorem to reduce the number of complex cryptographic operations and ciphertexts. Thus, our framework is more efficient in computation and communication than similar frameworks.

(iv) We present a comprehensive analysis for the security of our framework, which proves the privacy security of the gradients and the model as well as the integrity of the aggregated results. Finally, we simulate our framework in FedML on four benchmark datasets (and corresponding models): Federated MNIST, CINIC-10, CIFAR-10, and CIFAR-100. The experimental results demonstrate that our framework has high computation and communication efficiency.

The rest of the paper is organized as follows: the preliminaries and related works are briefly introduced in Section 2. The detailed procedures of our framework are presented in Section 3. The security analysis and experimental evaluation are provided in Sections 4 and 5, respectively. Finally, we conclude our paper in Section 6.

## 2. Preliminaries and Related Work

In this section, we introduce the preliminaries and related works of privacy preserving FL.

### 2.1. Paillier One-Way Trapdoor Permutation.

*2.1. Paillier One-Way Trapdoor Permutation.* In this paper, we employ the Paillier one-way trapdoor permutation [15] to realize the homomorphic signcryption mechanism. The details of the Paillier one-way trapdoor permutation are described as follows:

(i) Key generation: select two large prime numbers $p$ and $q$ randomly with $\gcd(pq, (p-1)(q-1)) = 1$ and compute the modulus $N = pq$ and the least common multiple $\lambda = \mathrm{lcm}(p-1, q-1)$. Select a random group generator $g \in \mathbb{Z}_{N^2}^*$, where the order of $g$ divides $n$. The key pairs is $((N, g), \lambda)$.

(ii) Encryption: for any message $m < N^2$, split $m$ into $m_1, m_2$ with $m = m_1 + Nm_2$. The ciphertext of $m$ is $c = g^{m_1} m_2^N \bmod N^2$.

(iii) Decryption: the decryption process is as follows:

$$\begin{aligned} m_1 &= \frac{L(c^\lambda \bmod N^2)}{L(g^\lambda \bmod N^2)} \bmod N, \\ c' &= c g^{-m_1} \bmod N, \\ m_2 &= c'^{\,N^{-1} \bmod \lambda} \bmod N, \\ m &= m_1 + N m_2. \end{aligned} \tag{1}$$

The $L$ function is defined as $L(u) = (u-1)/N$.

*2.2. Batchcrypt.* In Batchcrypt [13], the authors proposed a method to clip and quantize the gradients; it consists of two functions:

(i) $\alpha = dACIQ(s, V, v)$: C=compute the clipping threshold $\alpha$ according to the number $s$ of gradients $W$, the maximum gradient $V$, and the minimum gradient $v$ of each layer.

(ii) $\{\widetilde{w_i}\} = \text{Quantize}(\{w_i\}, \{\alpha_i\}, n)$: quantize the gradient $w_i$ into $r$ bits according to the scaled quantization range $[-n\alpha, n\alpha]$ in case the sum of gradients from all clients overflows. Here, $n$ denote the number of clients.

Batchcrypt [13] allows clipping and quantizing the gradients into fixed bit width integers, which can reduce the number of parameters in encryption and decryption.

*2.3. Chinese Remainder Theorem.* Suppose $m_0, m_2, \ldots, m_K$ are $K$ positive pairwise coprime integers. Let $M = m_1 \cdot m_2 \cdots m_K$. Then there exists one unique integer satisfying the following group of congruences: $y \equiv a_1 \bmod m_1, y \equiv a_2 \bmod m_2, \ldots, y \equiv a_K \bmod m_K$. Similarly, we can unpack $y$ to get $a_i \equiv y \bmod m_i\,(i = 1, 2, \ldots, K)$. To ease description, we denote the packing function as $y = CRT(\{a_1, a_2, \ldots, a_K\}, \{m_1, m_2, \ldots, m_K\})$ and the unpacking function as $\{a_1, a_2, \ldots, a_K\} = CRT\_\text{inverse}(y, \{m_1, m_2, \ldots, m_K\})$. For any two packed data $y_i$ and $y_j$, the equation $y_i + y_j \equiv a_l^i + a_l^j \bmod m_l\,(l = 1, 2, \ldots, K)$ holds. Thus, we know that the CRT satisfies additive homomorphism.

*2.4. Privacy Preserving Federated Learning.* Many researches have been devoted to privacy preserving FL (PPFL), and they are mainly based on cryptographic methods. Shokri and Shmatikov [16] employed selective parameter update atop differential privacy to protect training record privacy. Geyer et al. [17] applied differential privacy directly to guarantee client level differential privacy. But these differential privacy based approaches reduced model accuracy significantly. Bonawitz et al. [18] leveraged secure aggregation to protect privacy in FL. Nevertheless, it exposes the trained model in the plaintext form. To address this problem, Phong et al. [9] proposed an additively homomorphic encryption based FL framework. However, it brought heavy computation and communication overhead because the encryption scheme involves complex cryptographic operations and large ciphertexts. To improve efficiency, Batchcrypt [13] tried to encode groups of quantized gradients large integers, but their method cannot defeat collusion attacks among internal curious clients and the server.

*2.5. Verifiable Federated Learning.* In FL, a malicious or compromised server may falsify aggregated results returned to the clients. Several methods have been proposed to solve this problem.

Ma et al. [19] used bilinear aggregate signature [20] for integrity verification. But the verification process involves all clients. For a large number of clients, this process is time-consuming. Xu et al. [21] employed a homomorphic hash function to verify integrity, but it cannot protect the privacy of the jointly trained model. Zheng et al. [22] designed Helen, a framework utilized to secure multiparty computing protocol [23], to verify the correctness of the aggregated results. Nevertheless, the framework does not support training complex models such as neural networks. Zhang et al. (referred to as CRT-Paillier in Section 5) [24] combined bilinear aggregate signature and Paillier encryption [15] to protect privacy and integrity of the aggregated results. But they cannot resist collusion attacks between the server and internal curious clients. Fu et al. proposed VFL [25] by employing Lagrange interpolation to realize secure aggregation and check the integrity of the aggregated results, but the parameter splitting process is costly, making it unsuitable for large deep learning models.

To summarize, among PPFL frameworks, many of them ignored verifying the correctness of the aggregated results. For verifiable frameworks, they either cannot apply to large nonlinear models, or cannot protect the privacy of the model, or cannot resist collusion attacks between the server and the clients. Our work is to address these problems.

## 3. Homomorphic Signcryption-Based Privacy Preserving Federated Learning Framework for IoT

*3.1. Threat Model and Design Goals.* In our framework, we assume security threats from three different perspectives:

(i) Internal curious clients may try to steal privacy information about other clients by inspecting the ciphertexts transmitted during the training process. Here we should note that, because the ultimate goal of the clients in is to train a good ML model, the curious clients may collude with the server to steal private information about other clients, but they will not collude with the sever to tamper the aggregated results.

(ii) The server may try to steal the jointly trained model because of its great economic value. But the model should be the common property of the clients.

(iii) The server may return falsified aggregated results to clients driven by some unexpected motivations.

Consequently, to enable a privacy preserving FL framework under the aforementioned threat model, the design goals are summarized as follows:

(i) Correctness: when the server and clients operate strictly according to the protocol, the aggregated results should be correct. This ensures that the clients achieve their goal and the final model has good performance.

(ii) Data privacy: data privacy means the privacy of a client's training data and gradients. The server and curious clients cannot gather any private information about the data and gradients from messages they receive.

(iii) Model privacy: model privacy refers to the privacy of the jointly trained model. The server or any other party not in the framework cannot steal the model by inspecting the immediate data transmitted during the learning process.

(iv) Verifiability: all clients can verify the integrity of the aggregation of parameters so that clients can detect the malicious behavior if the server returns tampered aggregated results.

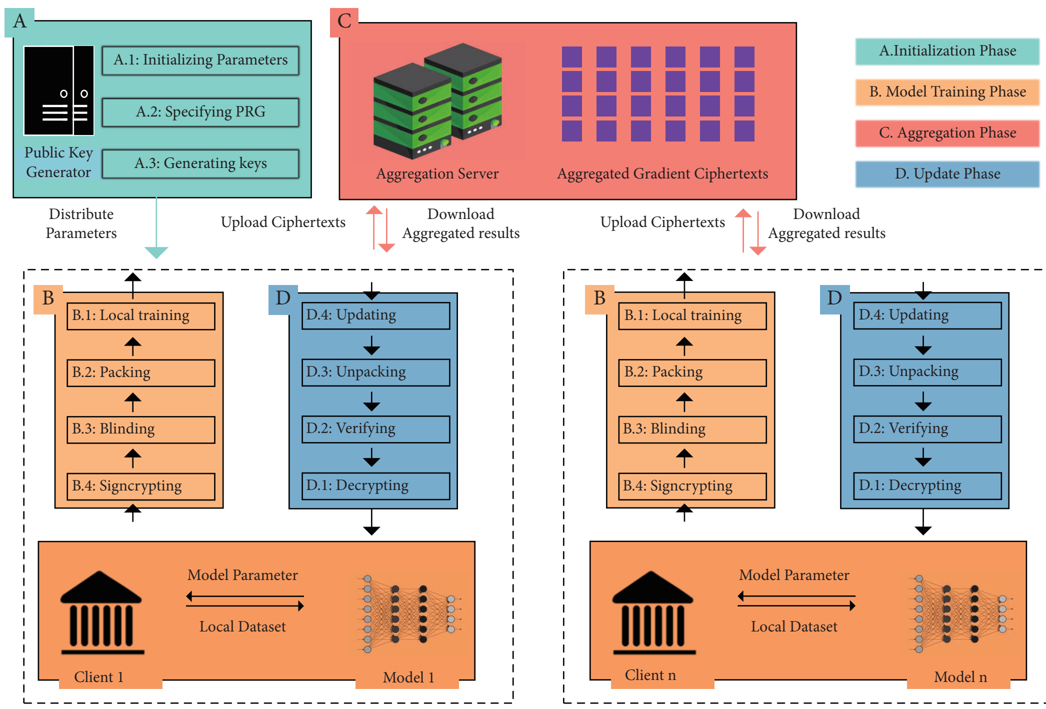

FIGURE 1: Overview of our framework.

*3.2. Overview.* As illustrated in Figure 1, our framework consists of 4 phases: initialization, model training, aggregation, and update. We summarize the 4 phases as follows:

(i) Initialization phase: the Public Key Generator (PKG) determines the model parameters and generates Paillier keypairs, Pseudo-Random Generator ($PRG$), random seed $S_0$, and $K$ pairwise coprime large numbers $\{m_l\}_{l=1,2,\ldots,K}$ for all clients.

(ii) Model training phase: all clients train the ML model locally clip and quantize the gradients into $r$ bits long integers, pack them using CRT, mask them with blinding factors, and then signcrypt them.

(iii) Aggregation phase: the server aggregates the ciphertexts from all clients and distributes the result to them.

(iv) Update phase: each client firstly decrypts the aggregated ciphertexts and then verifies the integrity of the aggregated gradients. If the result is correct, it unpacks them into plain aggregated gradients and updates their local model accordingly. Otherwise, it would terminate the learning process.

## 3.3. Detailed Construction

*3.3.1. Initialization.* For simplicity, we assume there are $n$ clients, and each client is uniquely indexed by a number $i$. The set of client index is represented by index $= \{1, 2, \ldots, n\}$. In our framework, the PKG needs to initialize the model parameters and generate keys and PRG as follows:

(1) Initializing model parameters: the PKG specifies the architecture of the model, randomly initializes the model parameters as $M_0 = \{M_i\}_{i=1,2,\ldots,T}$ ($T$ denotes the number of layers of the learning model), and specifies an appropriate learning rate $\eta$.

(2) Specifying packing parameters: the PKG generates $K$ positive integers $\{m_i\}_{i=1,2,\ldots,K}$ that are pairwise coprime $\gcd(m_i, m_j) = 1\,(\forall i \neq j)$. We define $M = \prod_{i=1}^{k} m_i$. These integers are used to pack gradients.

(3) Specifying blinding parameters: the PKG randomly chooses $n$ integer sets $\{s_1^j\}_{j=1,2,\ldots,|M_0|}$, $\{s_2^j\}_{j=1,2,\ldots,|M_0|}, \ldots, \{s_n^j\}_{j=1,2,\ldots,|M_0|} \in (\mathbb{Z}_N^*)^{|M_0|}$ such that $\sum_{i=1}^{n} s_i^j = 0$ for $j = 1, 2, \ldots, |M_0|$ ($|M_0|$ stands for the number of parameters of the model) and then sends $\{s_i^j\}_{j=1,2,\ldots,|M_0|}$ to client $i$.

(4) Specifying PRG: the PKG sends each client the same pseudorandom generator PRG$(\cdot)$ and seed $S_0$. The pseudorandom generator is used to synchronize random numbers among clients for signcrypting the gradients in each training round.

(5) Generating signcryption keys: the PKG sends each client a Paillier key pair $((N, g_1), \lambda)$ for encryption and decryption and a secret value $g_2^x$ to sign the gradients, where $N = pq, g_1 \in \mathbb{Z}_{N^2}^*$, $g_2 \in \mathbb{Z}_N^*$, and $x \in N$.

*3.3.2. Model Training Phase.* The model training phase of client $i\,(i \in$ index) consists of the following four steps:

(1) Local training: in $R-th$ round of training, client $i$ trains its local model on dataset $D_i$ and computes the gradients $G_i$.

(2) Packing: client $i$ sends the layer-wise maximum value $V_i^j$, minimum value $v_i^j$, and the size $s_j$ of layer $j\,(j = 1, 2, \ldots, T)$ to the server. After receiving those layer-wise clipping parameters from all clients, the aggregator server calculates the clipping threshold $\alpha_j = dACIQ(s_j, \mathrm{Max}(V_i^j)_{i=1,2,\ldots,n}, \mathrm{in}(v_i^j)_{i=1,2,\ldots,n})$, where $Max$ and $Min$ compute the maximum and minimum of a set, respectively. After receiving the clipping thresholds, client $i$ quantizes its gradients with $\widetilde{G}_i = \mathrm{Quantize}(G_i, \{\alpha_j\}_{j=1,2,\ldots,T}, n)$, client $i$ partitions the quantized gradients into $L = \lceil |\widetilde{G}_i|/K \rceil$ groups $P_i = \{p_i^1 = \{p_i^{1l}\}_{l=1,2,\ldots,K}, p_i^2 = \{p_i^{2l}\}_{l=1,2,\ldots,K}, \ldots, p_i^L = \{p_i^{Ll}\}_{l=1,2,\ \ldots,K}\}$. If $|\widetilde{G}_i|$ is not divisible by $K$, $\widetilde{G}_i$ should be padded with 0 s. Then client $i$ packs them into $W_i = \{w_i^1, w_i^2, \ldots, w_i^L\}$ with $w_i^j = CRT(p_i^j, \{m_l\}_{l=1,2,\ldots,K})$ and $w_i^j \in \mathbb{F}_M$. After packing, the number of cryptographic operations and ciphertexts is greatly reduced; thus the computation and communication efficiency is improved.

(3) Blinding: to resist collusion attacks between curious clients and the aggregation server, client $i$ blinds $w_i^j$ by adding its blinding factors to get $\widetilde{W}_i = \{\widetilde{w}_i^1, \widetilde{w}_i^2, \ldots, \widetilde{w}_i^l\} = \{w_i^1 + s_i^1, w_i^2 + s_i^2, \ldots, w_i^L + s_i^L\}$.

(4) Signcrypting: for each blinded gradient $\widetilde{w}_i^j \in \widetilde{W}_i$, client $i$ computes its signature $\sigma_i^j = g_2^{x w_i^j} \bmod N$ and uses the $PRG$ to synchronize with other clients a random number $r_R = S_R = PRG(S_{R-1})$ , where $S_R$ will be used as seed for the PRG in the next round of training. Finally, clients $i$ compute a signcryption of $\widetilde{w}_i^j$ as $\mathrm{cipher}_i^j = g_i^{\widetilde{w}_i} (\sigma_i^j r_R)^N \bmod N^2$.

We summarize the steps of model training phase in Algorithm 1.

*3.3.3. Aggregation Phase.*

(1) Aggregation: after receiving the ciphertexts from all clients, the server aggregates them according to the layer index $j\,(j = 1, 2, \ldots, L)$. Because Paillier trapdoor permutation is additively homomorphic, the sever computes

$$\begin{aligned}\mathrm{cipher}^j &= \prod_{i=1}^{n} \mathrm{cipher}_i^j,\\ &= \prod_{i=1}^{n} g_1^{w_i^j + s_i^j}\left(g_2^{x\left(w_i^j + s_i^j r_R\right)^N}\right),\\ &= g_1^{\sum_{i=1}^{n} w_i^j + \sum_{i=1}^{n} s_i^j}\left(g_2^{x\sum_{i=1}^{n} w_i^j + x\sum_{i=1}^{n} s_i^j} r_R^n\right)^N \bmod N^2.\end{aligned} \tag{2}$$

Because $\sum_{i=1}^{n} s_i^j = 0$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, |M_0|$, we have

$$\mathrm{cipher}^j = g_1^{\sum_{i=1}^{n} w_i^j}\left(g_2^{x\sum_{i=1}^{n} w_i^j} r_R^n\right)^N \bmod N^2. \tag{3}$$

*3.3.4. Update Phase.* After receiving the aggregated ciphertexts $\{\mathrm{cipher}^j\}_{j=1,2,\ldots,L}$, each client completes the update phase as follows.

(1) Decrypting: client $i$ firstly decrypts the $j-th$ ciphertext:

$$\begin{aligned}W^j &= \frac{L\left(\left(\mathrm{cipher}^j\right) \bmod N^2\right)}{L\left(g_1 \lambda \bmod N^2\right)} \bmod N^2 = \sum_{i=1}^{n} w^j + \sum_{i=1}^{n} s^j\\ &= \sum_{i=1}^{n} w^j.\end{aligned} \tag{4}$$

Then it computes the aggregated signature as follows:

$$\begin{aligned}\widetilde{\mathrm{cipher}}^j &= \mathrm{cipher}^j g_i^{-W^i} \bmod N,\\ \sigma^j &= \left(\widetilde{cipher}^j\right)^{N^{-1} \bmod \lambda} r_R^{-nN} \bmod N\\ &= g_2^{x\sum_{i=1}^{n} w_i^j \bmod \lambda} \bmod N.\end{aligned} \tag{5}$$

(2) Verifying: for each client $i$, if the equation $g_2^{xW^j \bmod \lambda} \bmod N \equiv \sigma^j$ holds, the aggregation results are correct. Otherwise, it is supposed to be falsified, and the learning process terminates.

(3) Unpacking: if the termination condition is not satisfied, client $i$ unpacks the plain aggregated gradients $G_R = \sum_{i=1}^{n} G_i = CRT\_\mathrm{inverse}(W, \{m_1, m_2, \cdots, m_K\})$, where $W = \{W^1, W^2, \ldots, W^L\} = \{\sum_{i=1}^{n} w_i^1, \sum_{i=1}^{n} w_i^2, \ldots, \sum_{i=1}^{n} w_i^L\}$.

(4) Updating: client $i$ updates the model parameters $M_R = M_{R-1} - (\eta/n) G_R$ and prepares for the next round of aggregation.

We summarize the steps of update phase in Algorithm 2.

## 4. Security Analysis

**Theorem 1.** *The privacy of the clients' gradients is protected against internal curious clients.*

*Proof.* Suppose an internal curious client $i$ intercepts a ciphertext from client $l$ by intercepting the messages between client $l$ and the server. Then client $i$ has $\mathrm{cipher}_l^j = g_1^{\widetilde{w}_l^j} (\sigma_l^j r_R)^N \bmod N^2\ (j = 1, 2, \ldots, L)$. Though it can decrypt them to get $\widetilde{w}_l^j = w_l^j + s_l^j$ and $g_2^{x(w_l^j + s_l^j)}$, it cannot infer any information about the gradients $w_l^j$ because the blinding factor $s_l^j$ is hidden from it. Therefore, our framework can protect the privacy of the clients' gradients against internal curious clients. □

```
Input: Training round $R$, model parameters $M_{R-1}$, dataset $D_i$,quantization bit width $r$, blinding factors $\{s_i^j\}_{j=1,2,\dots,|M_0|}$, $PRG$, random seed $S_{R-1}$, Paillier key pairs $((N,g_1),\lambda)$, secret value $g_2^x$
Output: Signcryption of the masked gradients $Cipher_i^j$
  function MODELTRAINING
    Compute gradients $G_i$ based on $M_{R-1}$ and $D_i$
    Send layer-wise gradients Max-Min values and sizes $\{V_i^j, v_i^j, s_j\}_{j=1,2,\dots,T}$ to the aggregation server
    Clip $G_i$ with corresponding threshold $\{n\alpha_j\}_{j=1,2,\dots,T}$ (Advance Scaling) and quantize them into $r$ bits
    Batch the quantized gradients $\widetilde{G}_i$ layer by layer into $W_i = \{w_i^1, w_i^2, \dots, w_i^L\} = \text{Encode}(\widetilde{G}_i)$
    Blind $W_i$ with $\{s_i^j\}_{j=1,2,\dots,|M_0|}$ to compute $\widetilde{W}_i = \{\widetilde{w}_i^1, \widetilde{w}_i^2, \dots, \widetilde{w}_i^l\} = \{w_i^1 + s_i^1, w_i^2 + s_i^2, \dots, w_i^L + s_i^L\}$
    Sign each blinded gradient $\widetilde{w}_i^j \in \widetilde{W}_i$ with $g_2^x$, use the PRG to generate the synchronizing random number $r_R = S_R = PRG(s_{R-1})$, compute the signcryption $\text{cipher}_i^j = g_i^{w_i} (\sigma_i^j r_R)^N \bmod N^2$.
    Send $\{cipher_i^j\}_{j=1,2,\dots,T}$ to the aggregation server
  end function
```

Algorithm 1: Model training phase of client *i*.

```
Input: Aggregated ciphertexts $\{cipher^j\}_{j=1,2,\dots,L}$
Output: Updated Model $M_R$
  function UPDATE
    Decrypt $\{\text{cipher}^j\}_{j=1,2,\dots,L}$ to get $W = \{W^j\}_{j=1,2,\dots,L} = \{\sum i = 1nw_i^j + s_i^j\}_{j=1,2,\dots,L}$, compute $\widetilde{\text{cipher}}^j = \text{cipher}^j g_i^{-W^i} \text{mod} N$ and $\sigma^j = (\widetilde{\text{cipher}}^j)^{N^{-1} \bmod \lambda} r_R^{-nN} \bmod N = g_2^{x \sum i=1nw_i^j \bmod \lambda} \bmod N$
    Verify the integrity of the aggregated gradients by checking if the equation $g_2^{xW^j \bmod \lambda} \bmod N \equiv \sigma^j$ holds. If not, terminate the learning process.
    Decode the aggregated gradients to get $G_R = \sum_{i=1}^n G_i = \text{Decode}(W)$.
    Update model with $M_R = M_{R-1} - (\eta/n)G_R$.
  end function
```

Algorithm 2: Update phase of client *i*.

**Theorem 2.** *The privacy of the clients' gradients is protected against collusion attack between $n-2$ clients.*

*Proof.* Assume the set of collusion clients is $|Adv|$ with $|Adv| = 2$. For clients $i, l \notin Adv$, their ciphertexts $\text{cipher}_i^j = g_{1,j}^{w_i} (\sigma_i^j r_R)^N \text{mod} N^2$ and $\text{cipher}_l^j = g_1^{w_l} (\sigma_l^j r_R)^N \text{mod} N^2\ (j = 1, 2, \dots, L)$ contain blinding factors $s_i^j$ and $s_l^j$ that are unknown to clients in $Adv$ and the server. For any aggregated ciphertexts include $cipher_i^j$ and $cipher_l^j$, the situation is similar. Thus, even when the server colludes with $n-2$ clients, they cannot infer any information about other clients' gradients. □

**Theorem 3.** *In our framework, each client can independently verify the correctness of the aggregated results, and the server cannot deceive the clients with tampered aggregated results.*

*Proof.* If the clients receive the correct aggregated results, obviously the equation $g_2^{xW^j} = \sigma^j\ (j = 1, 2, \dots, L)$ holds. Assume a client receives forged aggregated results, and without loss of generality, we assume the sever falsified the aggregated results to

$$\widehat{\text{cipher}} = g_1^{\Delta_1} g_2^{\Delta_2} \prod_{i=1}^{n} \text{cipher}_i^j = g_1^{\sum_{i=1}^{n}(\widetilde{w}_i^j + \Delta_1)} \left( g_2^{x \sum_{i=1}^{n}(\widetilde{w}_i^j + \Delta_2)} r_R^n \right)^N \text{mod} N^2. \quad (6)$$

If the malicious server tries to successfully fool the verification mechanism of our framework, it must make sure the following equation holds: $g_2^{\sum_{i=1}^n (x\widetilde{w}_i^j + x\Delta_1)} = g_2^{x \sum_{i=1}^n (\widetilde{w}_i^j + \Delta_2)}$; namely, the server should select $\Delta_1$ and $\Delta_2$ that satisfy $x\Delta_1 = \Delta_2$. However, it is impossible because $g_2^x$ is confidential to the server. Therefore, we prove that the server cannot deceive the clients with tampered aggregated results. □

## 5. Performance Evaluation

We evaluate the performance of our framework in this section. Specifically, we first compare its accuracy against that of the original FL framework in four benchmark datasets: Federated MNIST, CINIC-10, CIFAR-10, and CIFAR-100. The models corresponding to the datasets are listed in Table 2. Then we compare its computation and communication cost with that of the other two verifiable PPFL frameworks: VFL [25] and CRT-Paillier [24].

Table 2: Datasets and corresponding models.

| Datasets | Model structure | Model parameters (#) | Model size (MB) |
|---|---|---|---|
| CINIC-10 | ResNet-56 | 591322 | 18.05 |
| CIFAR-100 | ResNet-56 | 591322 | 18.05 |
| Federated EMNIST | CNN | 1206590 | 36.82 |
| CIFAR-10 | ResNet-18+ group normalization | 11232612 | 342.79 |

We should note that though ResNet-18 has less layers than ResNet-56; because of the group normalization operation, it has more parameters than ResNet-56.

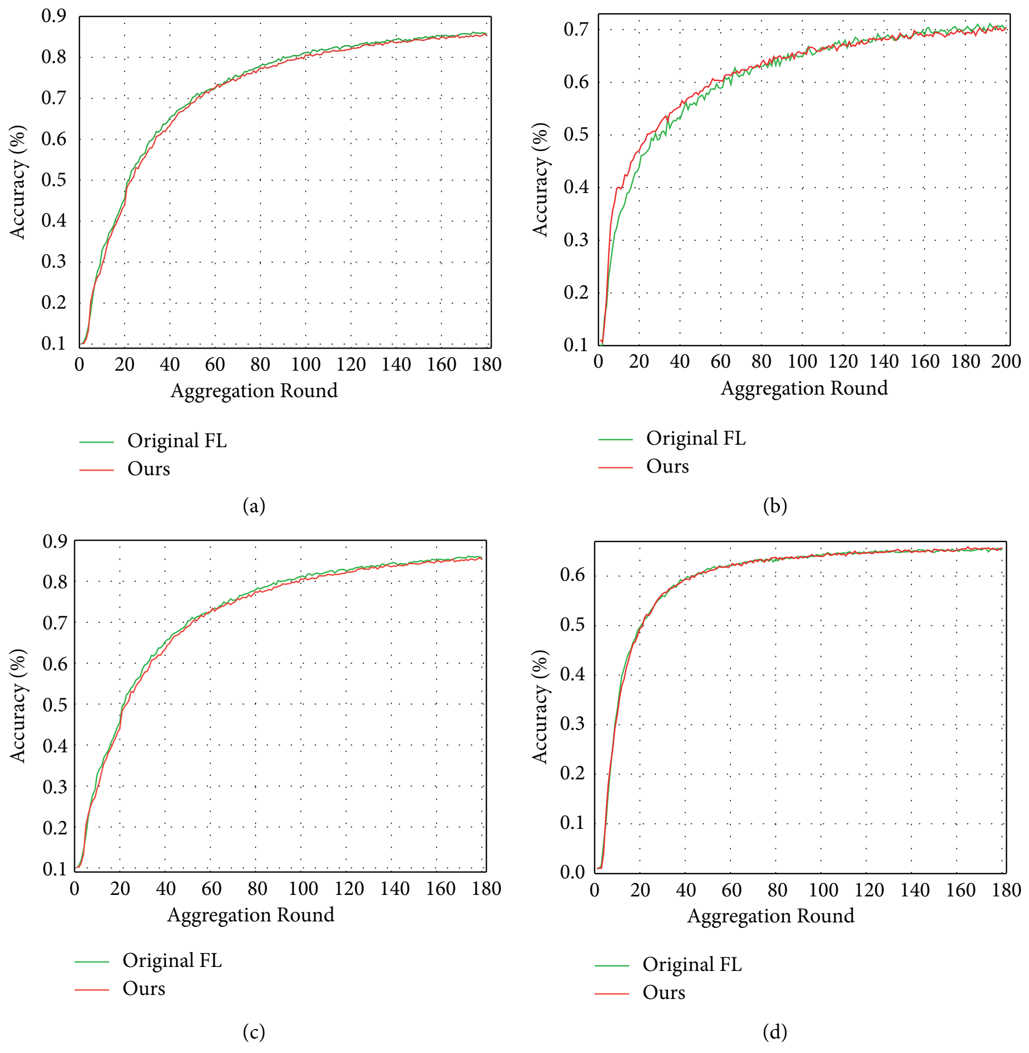

Figure 2: Comparison of performance of original FL and our framework. (a) Accuracy on FMNIST, (b) accuracy on CINIC-10, (c) accuracy on CIFAR-10, and (d) accuracy on CIFAR-100.

*5.1. Experimental Setup.* Our evaluation experiments are conducted on a Dell T7920 workstation with 1 Intel Xeon Silver 4210R CPU and 32 GB RAM. The OS is Ubuntu 18.04. We employ the FedML [26] in its standalone simulation computing paradigms to build the baseline framework. In both our and CRT-Paillier frameworks, the open-sourced python-Paillier [27] is adopted as the Paillier HE implementation. In our experiments, according to VFL [25] and CRT-Paillier [24], the gradients in their frameworks are of 32-bit length, while the gradients in our framework are clipped and quantized into 16-bit length with the dACIQ and Quantize function. The Paillier key size in both our framework and CRT-Paillier is set to 2048 bits, just as recommended in NIST [28].

*5.2. Discussion of Results.* Accuracy: we compared the performance of our framework with that of the original FL framework (Figure 2). For FMNIST, CINIC-10, CIFAR-10, and CIFAR-100, our framework can achieve an accuracy of 81.01%, 70.39%, 85.36%, and 65.60%, respectively, which is very close to that of the original FL, which has an accuracy of

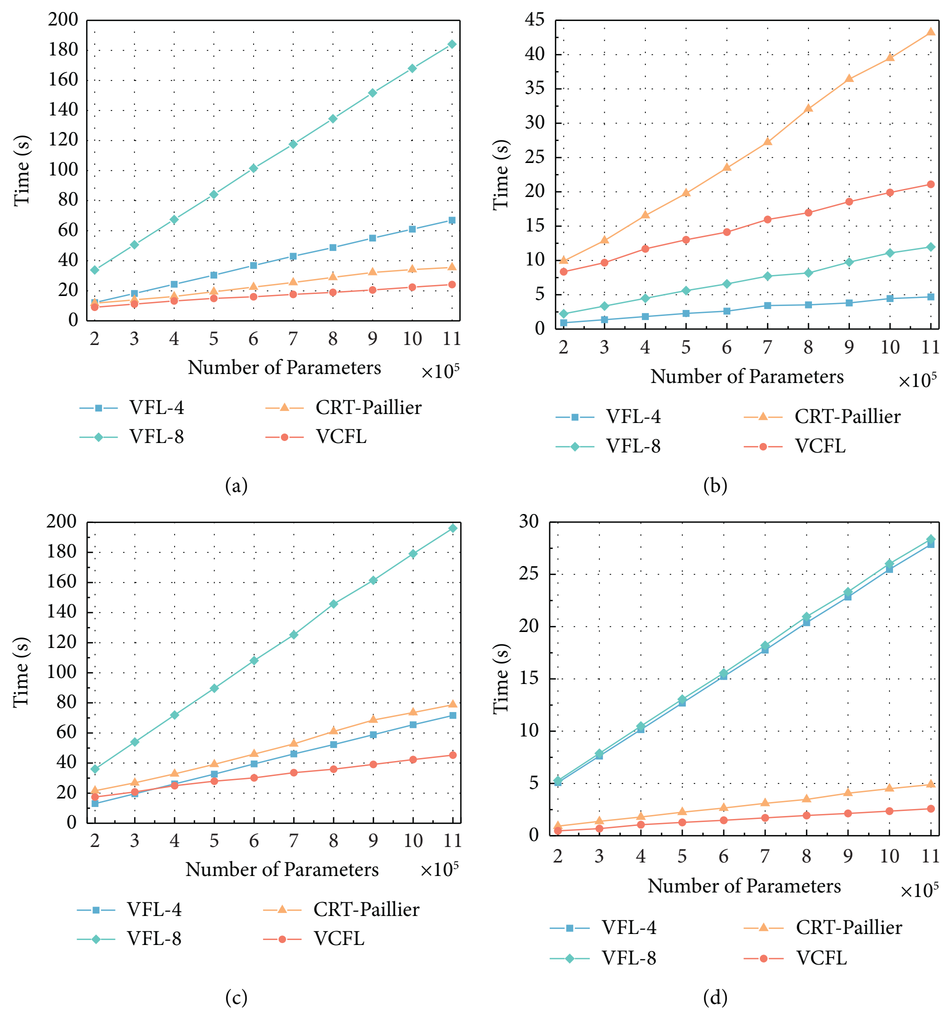

Figure 3: Comparison of time cost for VFL, CRT-Paillier, and our framework. (a) Encryption cost of a client, (b) decryption of a client, (c) total cost of a client, and (d) Aggregation cost of the server.

80.99%, 70.49%, 85.81%, and 65.23%, respectively. The results demonstrate that our framework can guarantee the privacy of the learned model without compromising its performance. In fact, our quantized training sometimes has better results. Prior quantization work has observed a similar phenomenon [29], where the stochasticity introduced by quantization can reduce overfitting, similar to the function of dropout layer [30].

Time cost, in Figure 3, we compared the time cost of our framework with that of VFL and CRT-Paillier. In VFL, the degree of the interpolation polynomials is set to $m = 4$ (VFL-4) and $m = 8$ (VFL-8), respectively. It is known that the security level and cost of VFL are higher with larger $m$. In CRT-Paillier, because the plaintext in Paillier encryption should be less than the modulus $N$, the maximum group size is 60 to avoid overflow error. While the gradient in our framework quantized into 16 bits, the maximum batch size is 120.

Figure 3(a) shows that the encryption cost per client of all frameworks increases linearly with the amount of parameters. The encryption costs of our framework are 9.06 s for $2 \times 10^5$ parameters and 24.17 s for $11 \times 10^5$ parameters, respectively. In addition, the encryption costs of CRT-Paillier are 11.73 s and 35.55 s, VFL-8 are 33.83 s and 184.15 s, and VFL-4 are 12.19 s and 66.99 s. Though both our framework and CRT-Paillier employ Paillier encryption to protect privacy, our framework needs fewer encryption operations and takes less time because its batch size is larger. For VFL, though the interpolation is very fast, the parameter splitting costs are expensive. Thus, our framework is more efficient than VFL-8 and VFL-4.

Figure 3(b) presents the decryption cost of all frameworks. The decryption costs of our framework are 8.35 s for $2 \times 10^5$ parameters and 21.10 s for $11 \times 10^5$ parameters. The encryption costs of CRT-Paillier are 9.91 s and 43.21 s, VFL-8 are 2.22 s and 11.87 s, and VFL-4 are 0.91 s and 4.68 s. Because our framework and CRT-Paillier utilize the Paillier scheme to protect privacy, which involves exponentiations and modular multiplications with large integers, the decryption costs of both our framework and CRT-Paillier are a little higher than VFL. But our framework takes less time than CRT-Paillier because of its larger batch size.

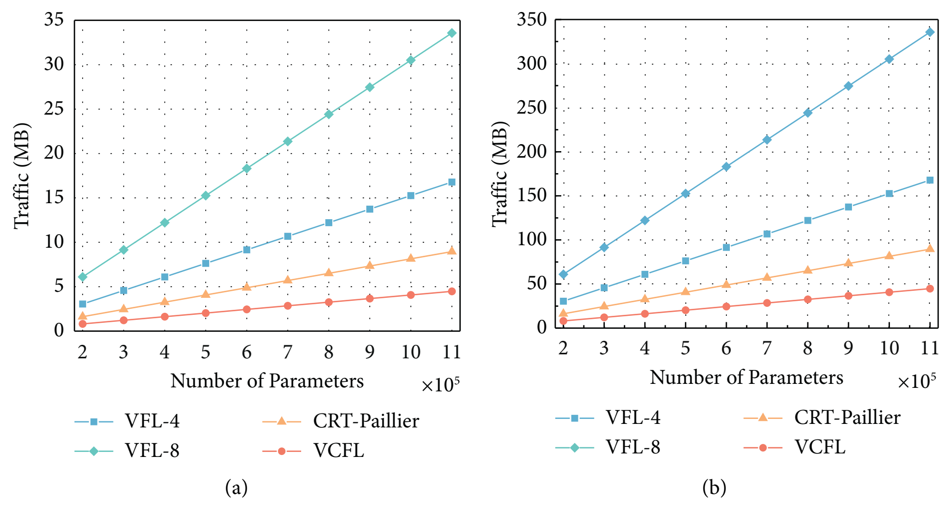

Figure 4: Comparison of communication cost of VFL, CRT-Paillier, and our framework. (a) Traffic of a client; (b) traffic of the server.

Figure 3(c) displays the total time cost of a client. From the figure we know that the total cost of a client of our framework is less than both VFL and CRT-Paillier, and the gap grows linearly with the number of parameters. We should note that, as giant companies or organizations prefer large deep learning models to boost their performance, our framework is more suitable for large AI models.

For the server, the overhead is caused by secure aggregation. Figure 3(d) shows the aggregation cost of the server. Because of larger batching size, our framework needs fewer sum operations on the ciphertexts. Thus, our framework is more efficient than CRT-Paillier and VFL for the server.

Communication cost: we compared the communication cost of our framework with that of VFL and CRT-Paillier in Figure 4. Since we simulate different FL frameworks in the standalone computing paradigms of FedML, we use the ciphertext size exchanged between the server and clients as the metric for communication cost. In VFL-4, each gradient is randomly split into 4 parameters. Though the authors keep the number of parameters the same as the original model by employing CRT to batch parameters, the size of the batched results grows accordingly. Thus, the ciphertext expansion rate for VFL-4 is 4. Similarly, the ciphertext expansion factor of VFL-8 is 8. In CRT-Paillier framework, every 60 gradients are grouped together and then are encrypted to get a $2048 \times 2$ bit-length ciphertext, the ciphertext expansion factor is approximately $2048 \times 260 \times 32 \approx 2.13$. Similarly, the ciphertext expansion factor of our framework is approximately $2048 \times 2120 \times 32 \approx 1.07$. Therefore, our framework is more communication efficient than VFL and CRT-Paillier.

## 6. Conclusion

In this paper, we have designed a preserving FL framework for IoT based on the homomorphic signcryption mechanism we designed. In our framework, each client can aggregate the gradients securely and verify the integrity of the aggregated results. Besides, our framework can also resist the collusion attacks between the server and at most $n - 2$ clients. Finally, experiments on four benchmark datasets show that our framework can protect the privacy and integrity of the learned model while guaranteeing its performance, and our framework is more efficient in computation and communication than existing similar frameworks. In future work, we will try to design more flexible privacy preserving framework that allows dynamic joining in and dropping out of clients.

## Data Availability

MNIST dataset is available at https://yann.lecun.com/exdb/mnist/.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work was supported by Innovative Research Team in Engineering University of PAP (KYTD201805) and Natural Science Foundation of Shaanxi Province (2020JM-537).

## References

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized Data,” 2017, https://arxiv.org/abs/1602.05629.

[2] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated Learning: Strategies for Improving Communication Efficiency,” 2017, https://arxiv.org/abs/1610.05492.

[3] V. Perifanis, G. Drosatos, G. Stamatelatos, and P. S. Efraimidis, “FedPOIRec: Privacy Preserving Federated POI Recommendation with Social Influence,” 2021, https://arxiv.org/abs/2112.11134.

[4] T. Yang, G. Andrew, H. Eichner et al., "Applied Federated Learning: Improving Google Keyboard Query Suggestions," p. 02903, 2018, https://arxiv.org/abs/1812.02903.

[5] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge AI: intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.

[6] G. Szegedi, P. Kiss, and T. Horváth, *Evolutionary Federated Learning on EEG-Data 8,* ITAT, 2019.

[7] C. Wang, C. Chen, Q. Pei, Z. Jiang, and S. Xu, "An information centric in-network caching scheme for 5g-enabled internet of connected vehicles," *IEEE Transactions on Mobile Computing*, vol. 1, p. 1, 2021.

[8] C. Chen, L. Liu, S. Wan, X. Hui, and Q. Pei, "Data dissemination for industry 4.0 applications in internet of vehicles based on short-term traffic prediction," *ACM Transactions on Internet Technology*, vol. 22, no. 1, pp. 1–18, 2022.

[9] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *Technical Reports Series*, vol. 715, 2017.

[10] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership Inference Attacks against Machine Learning Models," 2017, https://arxiv.org/abs/1610.05820.

[11] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond Inferring Class Representatives: User-Level Privacy Leakage from Federated Learning," 2018, https://arxiv.org/abs/1812.00535.

[12] C. Liu, S. Chakraborty, and D. Verma, "Secure model fusion for distributed learning using partial homomorphic encryption," *Policy-Based Autonomic Data Governance*, vol. 11550, pp. 154–179, 2019.

[13] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning 15," in *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, pp. 493–506, U S A, July 2020.

[14] J. Ma, S.-A. Naas, S. Sigg, and X. Lyu, "Privacy-preserving Federated Learning Based on Multi-Key Homomorphic Encryption," 2021, https://arxiv.org/abs/2104.06824.

[15] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology - EUROCRYPT '99*, J. Stern, Ed., vol. 1592, pp. 223–238, Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[16] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1310–1321, ACM, Denver Colorado USA, October 2015.

[17] R. C. Geyer, T. Klein, and M. Nabi, "Differentially Private Federated Learning: A Client Level Perspective," 2018, https://arxiv.org/abs/1712.07557.

[18] K. Bonawitz, V. Ivanov, B. Kreuter et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, ACM, Dallas Texas USA, October 2017.

[19] X. Ma, F. Zhang, X. Chen, and J. Shen, "Privacy preserving multi-party computation delegation for deep learning in cloud computing," *Information Sciences*, vol. 459, pp. 103–116, 2018.

[20] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Lecture Notes in Computer Science*, G. Goos, J. Hartmanis, J. van Leeuwen, and E. Biham, Eds., vol. 2656, pp. 416–432, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[21] 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, 2020.

[22] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: maliciously secure coopetitive learning for linear models," *2019 IEEE Symposium on Security and Privacy (SP)*, vol. 1, pp. 724–738, 2019.

[23] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Lecture Notes in Computer Science*, R. Safavi-Naini and R. Canetti, Eds., vol. 7417, pp. 643–662, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[24] X. Zhang, A. Fu, H. Wang, C. Zhou, and Z. Chen, "A privacy-preserving and verifiable federated learning scheme," in *Proceedings of the ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, Dublin, Ireland, June 2020.

[25] A. Fu, X. Zhang, N. Xiong, Y. Gao, and H. Wang, "VFL: A Verifiable Federated Learning with Privacy-Preserving for Big Data in Industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, 2022.

[26] C. He, S. Li, J. So et al., *FedML: A Research Library and Benchmark for Federated Machine Learning*, https://arxiv.org/abs/2007.13518, 2020.

[27] *DATA61, C. Python Paillier Library*, https://github.com/data61/python-paillier, 2013.

[28] E. Barker, L. Chen, A. Roginsky, A. Vassilev, R. Davis, and S. Simon, *Recommendation for pair-wise key establishment using integer factorization cryptography*, National Institute of Standards and Technology, Gaithersburg, MD, 2019.

[29] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," 2016, https://arxiv.org/abs/1606.06160.

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting 30," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.