




Article

# EVFL: Towards Efficient Verifiable Federated Learning via Parameter Reuse and Adaptive Sparsification

Jianping Wu <sup>1,\*</sup>, Chunming Wu <sup>1,\*</sup>, Chaochao Chen <sup>1</sup>, Jiahe Jin <sup>2</sup> and Chuan Zhou <sup>3,4,\*</sup>

<sup>1</sup> College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China; zjuccc@zju.edu.cn

<sup>2</sup> Key Laboratory of Key Technologies for Open Data Fusion in Zhejiang Province, Hangzhou 310007, China; jinjh@zj.gov.cn

<sup>3</sup> School of Microelectronics, Tianjin University, Tianjin 300100, China

<sup>4</sup> School of Information Engineering, Minzu University of China, Beijing 100081, China

\* Correspondence: wjpsself@zju.edu.cn (J.W.); wuchunming@zju.edu.cn (C.W.); zhouchuan@tju.edu.cn (C.Z.)

**Abstract:** Federated learning (FL) demonstrates significant potential in Industrial Internet of Things (IIoT) settings, as it allows multiple institutions to jointly construct a shared learning model by exchanging model parameters or gradient updates without the need to transmit raw data. However, FL faces risks related to data poisoning and model poisoning. To address these issues, we propose an efficient verifiable federated learning (EVFL) method, which integrates adaptive gradient sparsification (AdaGS), Boneh–Lynn–Shacham (BLS) signatures, and fully homomorphic encryption (FHE). The combination of BLS signatures and the AdaGS algorithm is used to build a secure aggregation protocol. These protocols verify the integrity of parameters uploaded by industrial agents and the consistency of the server’s aggregation results. Simulation experiments demonstrate that the AdaGS algorithm significantly reduces verification overhead through parameter sparsification and reuse. Our proposed algorithm achieves better verification efficiency compared to existing solutions.

**Keywords:** federated learning; data poisoning; situation prediction; privacy protection

**MSC:** 68P27



**Citation:** Wu, J.; Wu, C.; Chen, C.; Jin, J.; Zhou, C. EVFL: Towards Efficient Verifiable Federated Learning via Parameter Reuse and Adaptive Sparsification. *Mathematics* **2024**, *12*, 2479. <https://doi.org/10.3390/math12162479>

Academic Editors: Jialing He, Zhi Fang and Chunhai Li

Received: 10 July 2024

Revised: 3 August 2024

Accepted: 7 August 2024

Published: 10 August 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The Internet of Things (IoT) is rapidly gaining ground due to its ability to connect to other devices and operate autonomously, without human intervention. Within this landscape, the Industrial Internet of Things (IIoT) has found diverse applications across various sectors, including smart healthcare [1], intelligent manufacturing [2], and smart city [3]. Deep Learning (DL), a crucial branch of artificial intelligence, has been harnessed in conjunction with the Industrial Internet of Things to execute various industrial tasks such as fault diagnosis, intrusion detection, and intelligent control [4].

Although DL models have shown excellent performance, their effectiveness heavily relies on large-scale datasets. The traditional centralized training approach centralizes the data in a data center for training. However, data owners are often unwilling to share data due to privacy and business competition considerations. To address the challenge of achieving high-performance models while protecting data privacy, researchers have proposed federated learning (FL). This scheme allows multiple institutions to collaboratively develop DL models by sharing model parameters rather than raw data. This notable feature has sparked many applications of FL in the industry [5–9]. For instance, Zeng et al. [10] proposed a Homophilic Learning-based Federated Intelligence (HLFI) approach for industrial IoT device failure prediction, achieving remarkable predictive results. Zhang et al. [11] used FL to model virtual network embedding (VNE) and proposed a horizontal federated learning (HFL)-based VNE architecture (HFL-VNE). By deploying local servers in each

physical domain, they efficiently focused on local features, reduced resource fragmentation, and ensured local data privacy. Wang et al. [12] formulated a federated transfer learning framework for cross-domain prediction, addressing data scarcity and privacy concerns in contemporary smart manufacturing cross-domain applications.

However, in practical FL applications, problems such as inference attacks and poisoning attacks leading to privacy data leakage have occurred.

Firstly, the defense of FL against inference attacks from curious servers/clients during training is relatively weak [13,14]. The local model is injected with information on local data when training with local data. Agents may be attacked by adversaries when uploading local models. Adversaries may obtain local data information from model parameters by designing corresponding extractors. For instance, the work by [15] trained the mGANAI model to obtain the original training images of the participants. The main reason for the privacy leakage was that the participants uploaded the gradients in plaintext to the aggregation server. To protect the data privacy security of participants, some researchers have proposed excellent solutions. Wei et al. [16] proposed adding artificial noise to the client's parameters before model aggregation, employing local differential privacy for privacy protection. Kairouz et al. [17] proposed adding discrete Gaussian noise before security aggregation and after discretizing the agents' model updates. Adding disturbing noise to the model can be a good way to protect model privacy. However, there is a trade-off between model convergence and the level of privacy protection. Phong et al. [18] used homomorphic additive encryption for asynchronous stochastic gradient descent training of neural networks. This method can perform arithmetic operations directly on the ciphertext without decryption. Zhang et al. [19] used a homomorphic multiplicative encryption scheme to protect model parameters and achieve faster computational efficiency. While these homomorphic encryption-based methods can effectively protect the private information of each agent, they introduce significant computational overhead.

Secondly, due to security risks such as poisoning attacks, the correctness of server aggregation results in FL systems cannot always be guaranteed. Driven by illegal interests, a malicious cloud server may return incorrect aggregation results to agents. In this case, the global model will be attacked, and the agents will train a bad model. Such a server might also intentionally send designed results to a specific agent to analyze the features of that agent's data. To solve these problems, some solutions have been proposed. Among them, Ren et al. [20] used a linear homomorphic hash and digital signature for traceable verification, ensuring aggregation result accuracy. Xu et al. [21] designed a verifiable method based on homomorphic hash functions and pseudo-randomization techniques to support aggregated result verification. Zhang et al. [22] introduced the bilinear aggregate signature technology into federated learning to verify the correctness of aggregated results. To reduce the validation overhead, Fu et al. [23] used Lagrangian interpolation to set the interpolation points to verify the correctness of the aggregated results. The scheme kept the validation overhead constant regardless of the number of participants. Lin et al. [24] proposed a verification scheme based on discrete logarithms. The verification overhead was reduced by more than half compared to the bilinear set signature approach. Yang et al. [25] exploited keyed homomorphic hash functions and homomorphic signatures to construct verifiable aggregation tags and designed a batch verification method to enhance efficiency. However, the existing research schemes may not be suitable for resource-limited industrial applications due to high communication and computational costs during the verification process. Meanwhile, as important as the verification of the correctness of the aggregation results is, the integrity verification of the parameters uploaded by the participants is crucial, because if model parameters uploaded by unreliable participants are used in model aggregation, the global model may be poisoned, and the gradient aggregation protocol may be broken [19].

In summary, based on the analysis presented above, resource-limited industrial IoT faces three primary challenges in FL: (1) ensuring the privacy of industrial agents during model parameter exchange, (2) balancing training efficiency with the verification of cloud server aggregation results' correctness and distribution consistency, and (3) balancing model training efficiency while validating the integrity of agents' uploaded parameters.

To alleviate these issues, we propose a privacy-conscious and efficient verifiable federated learning scheme called EVFL, which combines adaptive gradient sparsification (AdaGS), BLS signatures, and fully homomorphic encryption schemes. The main contributions of this paper are as follows:

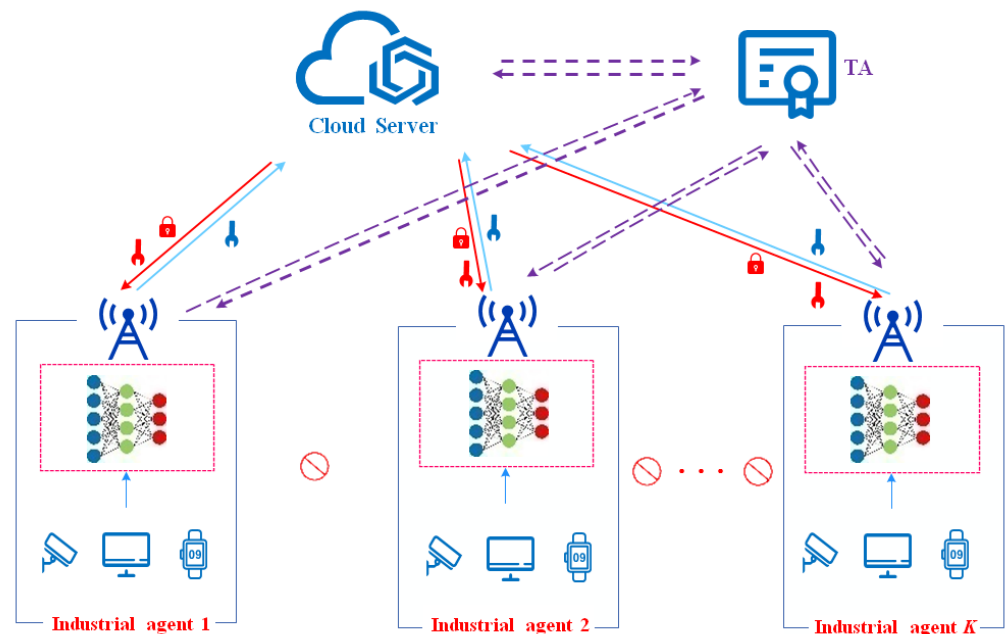
- We design an efficient federated learning mutual attack verification mechanism, which combines BLS signatures and adaptive gradient sparsification (AdaGS). (1) It can verify the integrity of industrial agents' upload parameters to prevent unreliable model parameters from participating. Global model aggregation: (2) It can verify the correctness of cloud server aggregation results and prevent malicious aggregation of cloud servers. (3) It can be extended to verify the consistency of distributed aggregation results, thereby protecting the rights and interests of industrial agents.
- We devise an innovative method, termed AdaGS, which employs adaptive gradient sparsification to minimize the burden of verification processes. By dynamically adjusting the level of sparsification and efficiently reusing parameters, AdaGS achieves high performance while preserving efficiency. In comparison to prevalent approaches, our scheme stands out as an optimal fit for environments constrained by limited resources.
- To protect data privacy during transmission, we employ full homomorphic encryption. A comprehensive security analysis confirms that EVFL effectively protects agents' data privacy and achieves secure verification for federated learning. Extensive experiments demonstrate the system's effectiveness, with only a moderate increase in computational cost.

The rest of the paper is organized as follows. Section 2 presents the system structure, threat model, and design model. Section 3 presents some preliminaries such as the bilinear map, BLS signature, and gradient sparsification (GS) method. Section 4 provides a detailed description of our proposed efficient and verifiable federated learning with privacy preservation. Section 5 analyzes the security, communication, and computation costs of the EVFL method. Section 7 evaluates performance on two datasets. Finally, Section 7 summarizes this study and presents our future work.

## 2. Problem Description

With the rapid development of technologies such as IIoT and cloud–fog–edge computing [26], data privacy protection in related application scenarios has attracted widespread attention. The converged application of IIoT based on federated learning has become a research hotspot. Yang et al. [27] developed the digital twin empowered IIoT (DTEI)-assisted deep reinforcement learning method for the selection process of IIoT devices in FL, especially for selecting IIoT devices with high utility values. Zhang et al. [28] presented an anomaly-based intrusion detection system with FL for privacy-preserving machine learning in future IIoT networks. Wang et al. [29] proposed a reliable anomaly detection strategy for IIoT using federated learning. The proposed strategy achieved high throughput, low latency, and high anomaly detection accuracy for privacy preservation in various IIoT scenarios. In summary, FL systems still face numerous privacy and security challenges. The cloud server responsible for aggregating model parameters may be malicious. It could distribute manipulated aggregation results, potentially launching nefarious attacks. During global model aggregation, industrial agents might submit unreliable model parameters that can adversely impact the overall model performance.

In this section, we describe the system model, threat model, and design goals of EVFL. The system model is illustrated in Figure 1.



**Figure 1.** The FL system model.

### 2.1. System Model

We consider a typical FL system with three types of entities in our scenario: (1) industrial agents, (2) cloud server (CS), and (3) trusted authority (TA). We describe these three types of entities according to their roles in the FL system model.

- **Industrial agents:** Industrial agents participate as distributed training participants under the coordination of a cloud server. Each agent maintains its local dataset, denoted as  $D_i$ , and trains a local model based on this dataset. During the training process, agents are required to transmit their local models to the cloud server. Additionally, they need to send some auxiliary information to both the CS and the TA for verification purposes.
- **Cloud server:** The cloud server is responsible for updating the global model, broadcasting the updated global model parameters, and controlling the task execution phases, such as the start and end of the training. To support integrity verification of the model parameters sent by each industrial agent, the cloud server needs to send proof information to the TA for validation.
- **Trusted authority (TA):** The TA is responsible for initializing the system parameters and publishing some public functions during the system startup phase. Based on the auxiliary information and proofs sent by the CS and the industrial agents, the TA verifies the integrality of the model parameters uploaded by the industrial agents and the correctness of the aggregated parameters of the cloud server.

### 2.2. Threat Model and Design Goal

We assume the cloud server and all industrial agents in the system are semi-honest, i.e., they execute the agreed-upon protocol honestly but may attempt to gain additional knowledge from the data they receive. In the system, the following threats may exist:

- **Cloud server:** A semi-honest cloud server can obtain the model sent from each agent and try to infer from these other entities' private information. A malicious cloud server may conduct dishonest data aggregation and send incorrect aggregation results to all agents. There is also a possibility that a cloud server with ulterior motives may

send correct aggregation results to some agents while providing incorrect results to specific agents.

- Industrial agents: Semi-honest industrial agents may generate unique parameter labels according to the agreed-upon protocol. However, the model parameters they upload may not be consistent with the parameters used to generate the labels.

Once any of the above threats occur, the performance of the global model is affected and even causes the model not to converge, seriously violating the agent's rights.

To solve the above problems, we try to achieve the following three goals.

- Privacy security: the proposed EVFL should guarantee the security of the training information exchanged during the training process to prevent unauthorized access by malicious third parties.
- Protection of right: one of the main goals of the proposed EVFL is to protect the rights and interests of agents, ensuring their access to correct and verifiable aggregation results.
- Model performance: an important goal of the proposed EVFL approach is to ensure model performance while minimizing verification overhead compared to existing schemes.

### 3. Preliminaries

In this section, we introduce some of the background used in the EVFL approach, including the BLS signature and gradient sparsification.

#### 3.1. Bilinear Map and Boneh–Lynn–Shacham (BLS) Signature

- Bilinear map [21]: Assume that there are two cyclic multiplicative groups  $G$  and  $G_1$  with the same prime order  $p$ , where  $G = \langle g \rangle$ .  $e : G_1 \times G_1 \rightarrow G_2$  is a bilinear map satisfying the following conditions:  
 Bilinearity:  $\forall a, b \in G$  and  $\forall x, y \in \mathbb{Z}_p^*$ , we have  $e(a^x, b^y) = e(a, b)^{xy}$ .  
 Computability:  $\forall a, b \in G$ ,  $e(a, b)$  can be computed efficiently.  
 Nondegeneracy:  $e(g, g) \neq 1$  is always true.
- BLS signature [30]: Assume that there are two cyclic multiplicative groups  $G$  and  $G_1$  with the same prime order  $p$ , where  $G = \langle g \rangle$ .  $e : G_1 \times G_1 \rightarrow G_2$  is a bilinear map. Given a hash function  $H: \{0,1\}^* \rightarrow G$ , which can hash directly to the elliptic curve, then a BLS signature can be constructed:  
 Randomly select  $sk$  as the signer's private key, the public key is  $pk = g^{sk} \in G$  and the message being signed is  $M$ . Next, the signer calculates the signature by formula  $Sign = H(M)^{sk} \in G$ . The verifier can take public key  $pk$  and check whether  $e(pk, H(M)) = e(g, Sign)$  holds. If the equation holds, the signature is valid; otherwise, the signature is invalid.

#### 3.2. Gradient Sparsification (GS) Method

Gradient sparsification (GS) is a method designed to enhance communication efficiency by compressing either the gradient or model weight vector. Its main objective is to exchange only a few significant gradients for global model aggregation. Two commonly used GS techniques are periodic averaged GS and top-k GS. In the periodic averaged GS method, a random subset of gradient elements is chosen for uploading and aggregation during each round. After passing a finite number of rounds, all the gradient elements are aggregated at least once. In top-k GS, only the  $k$  gradient elements with the highest importance values are uploaded and aggregated. Recent studies have demonstrated the effectiveness of GS methods in reducing the computational and communication overheads of federated learning systems. Han et al. [31] provide a theoretical analysis demonstrating that local and global models can still converge after applying GS. The key of gradient compression is to identify important gradients (i.e., the setting of the sparsification rate) and encode (quantize) them with low precision. The choice of the sparsification rate requires a trade-off between model performance and resource conservation. A high sparsification rate can significantly reduce the resource overhead but may result in a substantial reduction in

model performance. Conversely, a low sparsification rate guarantees a limited loss in model performance but saves very little computational and communication resources.

### 3.3. Cheon–Kim–Kim–Song (CKKS) [32] Cryptosystem

Homomorphic encryption (HE) [33], first proposed by Rivest, Adleman, and Dertouzos in 1987, is an encryption scheme in which homomorphic encryption operations on plaintexts are equal to the same operations on ciphertexts. Many HE schemes, such as RSA, ElGamal, and Paillier, can be applied to privacy protection, cloud computing, or genomic data [34]. However, these HE schemes can only support homomorphic addition or homomorphic multiplication, but not both. CKKS is a fully homomorphic encryption (FHE) scheme. Compared to Paillier, the CKKS scheme has faster encryption/decryption speeds and allows both additive and multiplicative homomorphic encryption [35,36]. Therefore, the CKKS scheme is used in EVFL. The additively homomorphic encryption is shown in Figure 2, and the multiplicative is shown in Figure 3.

In CKKS, the data (plaintext) are encrypted into ciphertext by the data owner using the public key, and the ciphertext is decrypted using the private key after the data processing is completed. CKKS scheme includes KeyGen, Encrypt, Evaluate, and Decrypt functions.

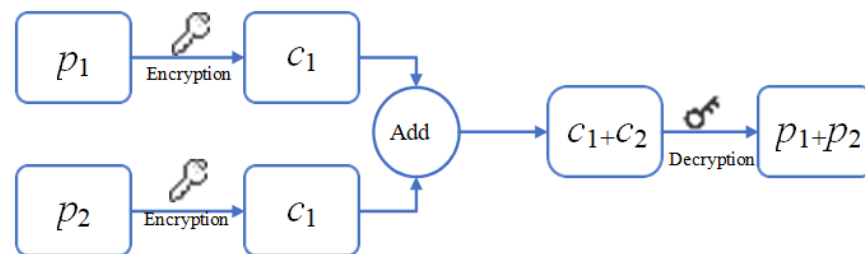


Figure 2. Addictive homomorphism.

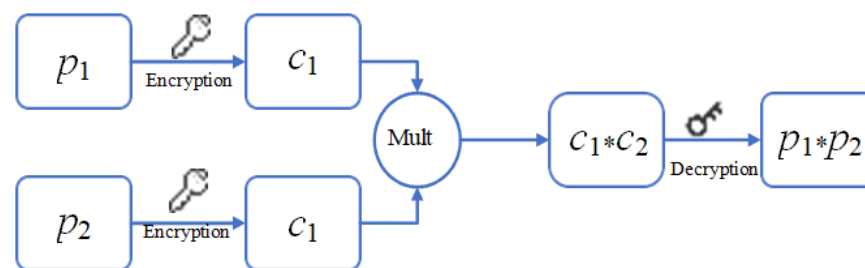


Figure 3. Multiplicative homomorphism.

- KeyGen takes the secure parameter  $\lambda$  as input and outputs the private key  $x$  and public key  $y$ , denoted as  $\text{KeyGen}(1^\lambda) \rightarrow (x,y)$ .
- Encrypt inputs public key  $y$  and plaintext  $p \in P$  ( $P$  is plaintext defined by  $y$ ) and outputs ciphertext  $c \in C$  ( $C$  is ciphertext defined by  $x$ ) obtained by encrypting plaintext  $p$  using public key  $y$ , denoted as  $\text{Encrypt}(p,y) \rightarrow c$ .
- Evaluate inputs the ciphertext  $c$  into function  $F$  to perform the associated addition and multiplication operations.
- Decrypt takes  $x$  and  $c$  as input and outputs plaintext  $p$  decrypted by  $x$ , which is recorded as  $\text{Decrypt}(c,x) \rightarrow p$ .

## 4. Our Approach

In this section, we present our proposed EVFL, which combines our designed adaptive gradient sparsification (AdaGS) method, BLS signatures, and a secure communication protocol based on the CKKS cryptosystem. We first describe the system model and then present the technical details of the scheme.

4.1. The Workflow of the Proposed Method

Our EVFL aims to protect the privacy and rights of industrial agents during the training process. The workflow of EVFL is shown in Figure 4. On the industrial agent side, each agent trains a local model using local data. Then, the local model is sparsified using the AdaGS algorithm, and the sparsified model’s parameters are encrypted using the CKKS encryption method. Before aggregating the model parameters, the TA verifies the integrity of the model parameters sent by the agents. On the server side, the server updates the global model parameters after receiving the parameters from the industrial agents. The TA verifies the correctness of the server aggregated parameters. After the cloud server distributes the parameters to the industrial agents, the industrial agents verify the consistency of the server distribution results. The workflow of EVFL consists of four phases as follows.

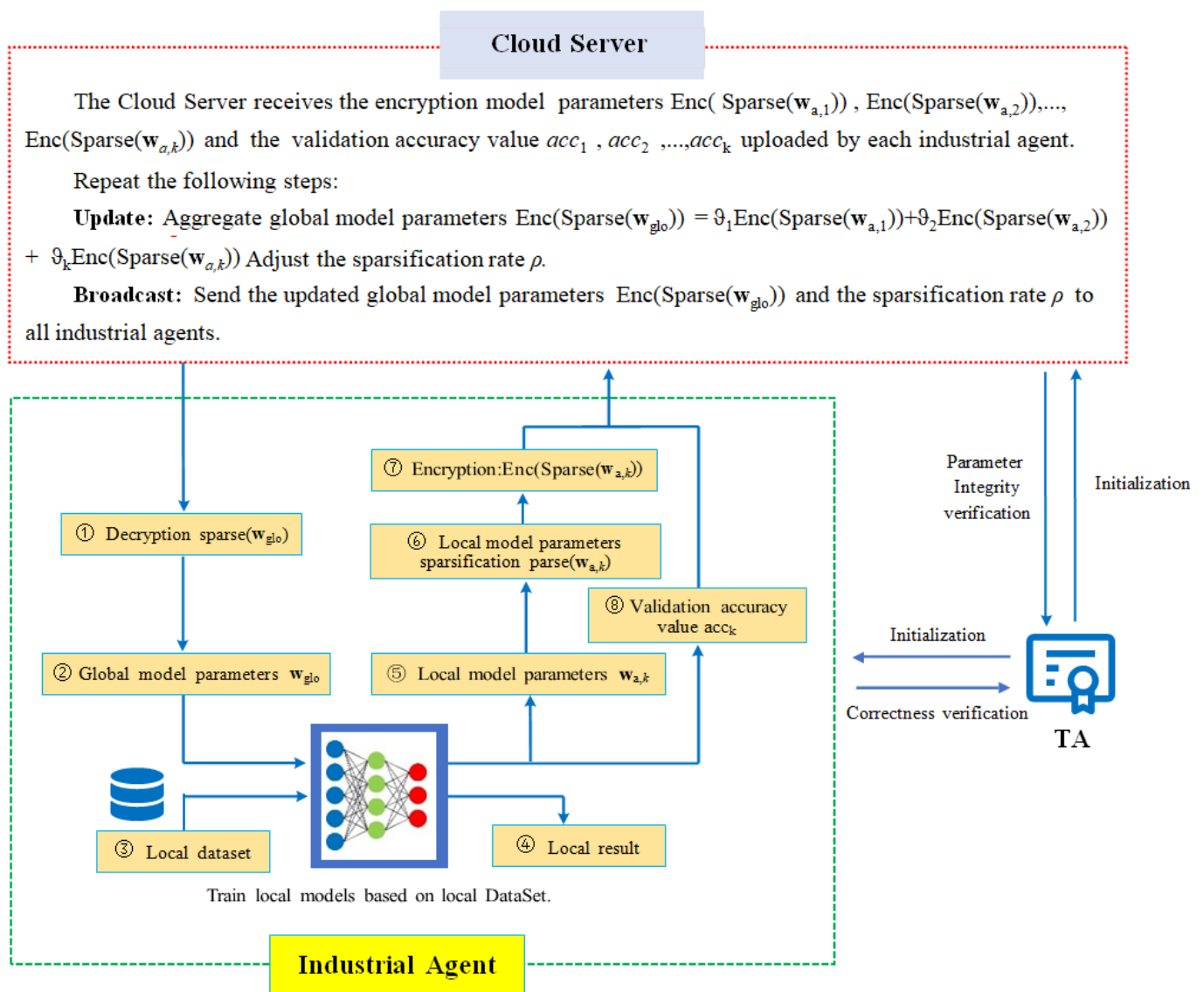


Figure 4. The workflow of the EVFL.

#### 4.1.1. System Setup

In the system setup phase, the TA is responsible for initializing the model parameters and generating various keys used for data transmission and verification. The main tasks in this stage are as follows:

The TA selects cyclic groups  $G$  and  $G_1$  with prime order  $q$  and generator  $g$ , and bilinear pairing maps  $e : G \times G = G_1$ . TA randomly selects a private key  $k$  and uses it to compute the public key  $\delta = g^k$ . One-way hash function  $H: \{0,1\}^* \rightarrow G$  is used for data integrity validation, and random number generators  $\text{rand}(\cdot)$  are used to generate different random numbers each time. The TA publishes parameters  $\{g, q, e, G, G_1, k, \delta\}$ .

The cloud server generates an initial global model and shares this initial global model with all the industrial agents. The cloud server receives the encryption model parameters  $\text{Enc}(\text{Sparse}(w_{a,1})), \text{Enc}(\text{Sparse}(w_{a,2})), \dots, \text{Enc}(\text{Sparse}(w_{a,K}))$  and the validation accuracy value  $\text{acc}_1, \text{acc}_2, \dots, \text{acc}_K$  uploaded by each industrial agent. The cloud server sends the updated global model parameters  $\text{Enc}(\text{Sparse}(w_{glo}))$  and the sparsification rate  $\rho$  to all industrial agents.

Each industrial agent needs to be registered in the system before federated training begins. During registration, an industrial agent randomly selects a private key  $sk$  and calculates the public key  $pk = g^{sk}$ . The industrial agent sends the public key  $pk$  to the TA, which returns a unique identifier ID to the industrial agent. In addition, the TA also publishes sufficient public parameters  $\lambda_k, k = 1, 2, \dots$  to all the industrial agents.

#### 4.1.2. Each Industrial Agent Trains the Local Model and Uploads the Model Parameters

After receiving the encrypted global model parameters  $\text{Enc}(w_{glo})$  and the sparsification rate  $\rho$  from the cloud server, each industrial agent decrypts  $\text{Enc}(w_{glo})$ . Then, each agent trains the model using its local data, which can be expressed as:

$$w_{a,k} = \text{MGD}(w_{a,glo}, D_{batch,k}) \tag{1}$$

where momentum gradient descent (MGD) denotes the momentum gradient descent algorithm,  $D_{batch,k}$  denotes the mini-batch data of the agent  $k$ .

After training the local model, each industrial agent performs sparsification on the local model to obtain  $\text{Sparse}(w_{a,k})$  and  $\text{Ind}_{a,k}$ . The sparse model can be reconstructed to obtain  $w_{a,k}$  using the cloud server's recovery sparsification model algorithm (detailed in Section 3.3). Next,  $\text{Sparse}(w_{a,k})$  and  $w_{a,k}$  are encrypted using the CKKS encryption algorithm to obtain  $\text{Enc}(\text{Sparse}(w_{a,k}))$  and  $\text{Enc}(w_{a,k})$ . Then,  $\text{Enc}(\text{Sparse}(w_{a,k}))$  and  $\text{Enc}(w_{a,k})$  are divided into  $n$  and  $(1 - \rho)n$  blocks, respectively, with each block further subdivided into  $s$  sectors. Here, we use Equations (2) and (3) to represent  $\text{Enc}(\text{Sparse}(w_{a,k}))$  and  $\text{Enc}(w_{a,k})$ , respectively.

$$spm^{(k)} = \text{Enc}(\text{Sparse}(w_{a,k})) = \{spm_{i,j}^k, i = 1, 2, \dots, (1 - \rho)n, j = 1, 2, \dots, s \in Z_p^{n \times s} \tag{2}$$

$$m^{(k)} = \text{Enc}(w_{a,k}) = \{m_{i,j}^k, i = 1, 2, \dots, n, j = 1, 2, \dots, s \} \in Z_p^{n \times s} \tag{3}$$

where  $spm_{i,j}^{(k)}$  denotes the data in the  $i^{th}$  block,  $j^{th}$  sector of  $\text{Enc}(\text{Sparse}(w_{a,k}))$ . Then agent  $k$  calculates the flags  $sp\varphi_i^{(k)}$  and  $\varphi_i^{(k)}$  and the signatures  $sp\sigma_i^{(k)}$  and  $\sigma_i^{(k)}$  of the  $i^{th}$  block, respectively.

$$sp\varphi_i^{(k)} = \sum_{j=1}^s H(sp m_{i,j}^{(k)}), i = 1, 2, \dots, (1 - \rho)n \tag{4}$$

$$sp\sigma_i^{(k)} = (sp\varphi_i^{(k)})^{sk_k} \cdot g^{\sum_{j=1}^s sp m_{i,j}^{(k)} \lambda_i}, i = 1, 2, \dots, (1 - \rho)n \tag{5}$$

$$\varphi_i^{(k)} = \sum_{j=1}^s H(m_{i,j}^{(k)}), i = 1, 2, \dots, n \tag{6}$$

$$\sigma_i^{(k)} = (\varphi_i^{(k)})^{sk_k} \cdot g^{\sum_{j=1}^s m_{i,j}^{(k)} \lambda_i}, i = 1, 2, \dots, n \tag{7}$$



The industrial agent sends the processed parameters  $spm^{(k)}$ ,  $Enc(Sparse(w_{a,k}))$ ,  $Ind_{a,k}$ ,  $sp\varphi_i^{(k)}$ , and  $sp\sigma_i^{(k)}$  to the cloud server and sends  $\varphi_i^{(k)}$  and  $\sigma_i^{(k)}$  to the TA. Algorithm 1 depicts the pseudo-code for training the local model and uploading the parameters.

---

**Algorithm 1** Training the local model and uploading parameters

---

**Input:** Model, training parameters

**Output:**  $spm^{(k)}$ ,  $Enc(Sparse(w_{a,k}))$ ,  $Ind_{a,k}$ ,  $sp\varphi_i^{(k)}$ ,  $sp\sigma_i^{(k)}$ ,  $\varphi_i^{(k)}$ ,  $\sigma_i^{(k)}$

- 1: **Industrial agent side:**
  - 2: **For industrial agent**  $k = 1, \dots, K$  **do:** // **Parallel running**
  - 3:     Receive  $Enc(w_{glo})$  and  $\rho$ .
  - 4:     Decrypt  $Enc(w_{glo})$  to obtain  $w_{glo}$ .
  - 5:     Train the model locally and obtain the model parameters  $w_{a,k}$ .
  - 6:     Perform sparse operation on model parameters to obtain  $Sparse(w_{a,k})$ ,  $Ind_{a,k}$ , and  $w_{a,k}$ .
  - 7:     Encrypt  $Sparse(w_{a,k})$  and  $w_{a,k}$  to obtain  $Enc(Sparse(w_{a,k}))$  and  $Enc(w_{a,k})$ .
  - 8:     Divide  $Enc(Sparse(w_{a,k}))$  into  $(1-\rho)n$  blocks, each of which is divided into  $s$  sectors, to obtain  $spm_{i,j}^{(k)}$ .
  - 9:     Divide  $Enc(w_{a,k})$  into  $n$  blocks, each of which is divided into  $s$  sectors, to obtain  $m_{i,j}^{(k)}$ .
  - 10:     Compute  $sp\varphi_i^{(k)}$ ,  $sp\sigma_i^{(k)}$ ,  $\varphi_i^{(k)}$ ,  $\sigma_i^{(k)}$ .
  - 11:     Send  $Enc(Sparse(w_{a,k}))$ ,  $Ind_{a,k}$ ,  $spm^k$ ,  $sp\varphi_i^k$ ,  $sp\sigma_i^k$ , and  $acc_k$  to the server.
  - 12:     Send  $\sigma_i^{(k)}$  and  $\varphi_i^{(k)}$  to the TA.
  - 13: **End**
  - 14: **Return**  $Enc(Sparse(w_{a,k}))$ ,  $Ind_{a,k}$ ,  $sp\varphi_i^{(k)}$ ,  $sp\sigma_i^{(k)}$ ,  $\varphi_i^{(k)}$ ,  $\sigma_i^{(k)}$ .
- 

4.1.3. Integrity Verification for Parameters Uploaded to the CS

In order to ensure the authenticity of the aggregated parameters, the integrity of the uploaded parameters of each agent is verified before the cloud server aggregates the global model. The TA first runs the random number generator to generate  $c$  integer random numbers  $\{\beta_1, \beta_2, \dots, \beta_c\}$  in the range between 1 and  $(1 - \rho)n$ . Then, the TA sends  $\{\beta_1, \beta_2, \dots, \beta_c\}$  to the cloud server. After receiving the challenge information from TA, the cloud server generates the proof information for each agent according to Equations (8)–(10) and sends  $sp\varphi^{(k)}$ ,  $sp\sigma^{(k)}$  and  $\eta_i^{(k)}$ ,  $i = \beta_1, \beta_2, \dots, \beta_c$  to TA.

$$sp\varphi^{(k)} = \prod_{i=\beta_1}^{\beta_c} (sp\varphi_i^{(k)})^c \tag{8}$$

$$sp\sigma^{(k)} = \prod_{i=\beta_1}^{\beta_c} (sp\sigma_i^{(k)})^c \tag{9}$$

$$sp\eta^{(k)} = g^{\lambda_i \sum_{j=1}^s spm_{i,j}^{(k)}}, i = \beta_1, \beta_2, \dots, \beta_c \tag{10}$$

After receiving the proof information from the cloud server, the TA verifies whether Equation (11) holds. If Equation (11) holds, the parameters uploaded by agent  $k$  can be used for global model aggregation. Otherwise, they are rejected. The whole process is shown in Algorithm 2

$$e(sp\sigma^{(k)}, g) = e(sp\varphi^{(k)}, pk_k).e(\prod_{i=\beta_1}^{\beta_c} sp\eta_i^{(k)}, g^c) \tag{11}$$

---

**Algorithm 2** Integrity verification for parameters uploaded to the CS

---

**Input:**  $spm^{(k)}, sp\varphi_i^{(k)}, sp\sigma_i^{(k)}$   
**Output:** 1: Integrity verification passed, 0: Integrity verification not passed  
 1: **TA side:**  
 2: Run random number generator to generate  $\{\beta_1, \beta_2, \dots, \beta_c\}$ .  
 3: Send challenge information  $\{\beta_1, \beta_2, \dots, \beta_c\}$  to CS.  
 4: **for industrial agent  $k = 1, \dots, K$  do:**  
 5:     **CS side:**  
 6:     Compute  $sp\varphi^{(k)}, sp\sigma^{(k)}$  and  $sp\eta_i^{(k)}$   
 7:     Send  $sp\varphi^{(k)}, sp\sigma^{(k)}$  and  $sp\eta_i^{(k)}$  to TA;  
 8:     **TA side:**  
 9:     Compute  $e1 = e(sp\sigma^{(k)}, g), e2 = e(sp\varphi^{(k)}, pk_k) \cdot e(\prod_{i=\beta_1}^{\beta_c} sp\eta_i^{(k)}, g^c)$   
 10:     If  $e1=e2$   
 11:     Return 1;  
 12:     else  
 13:     Return();  
 14: **End**

---

4.1.4. Parameter Aggregation by the CS and Correctness Verification for Aggregated Results

Before the cloud server performs model parameter aggregation, we assume that the cloud server has received the complete set of local model parameters from the agents. Agents whose parameters do not pass the integrity verification are rejected. The cloud server begins by recovering the model parameters,  $Enc(w_{a,k})$ , based on  $spm^{(k)}$  and  $Ind_{a,k}$  uploaded by agent  $k$ . After this recovery process, the global model is obtained by aggregating the model parameters using the following formula:

$$Enc(w_{glo}) = \prod_{k=1}^K (\vartheta_k \cdot Enc(w_{a,k})) \tag{12}$$

where  $\vartheta_k$  denotes data contribution ratios calculated by  $\vartheta_k = |Dk| / |Dall|$ .

After aggregating the model parameters, the cloud server distributes the aggregated model parameters to each agent. The correctness of the cloud server’s aggregation result is verified by each agent with the assistance of the TA. In this section, correctness means that the cloud server correctly aggregated all the validation parameters uploaded by the agents. According to Section 4.1.3, the TA receives  $\varphi^{(k)}$  and  $\sigma^{(k)}$  from each agent. The TA runs a random number generator to generate  $c$  random integers  $\{\beta_1, \beta_2, \dots, \beta_c\}$  in the range between 1 and  $n$ . Then, the TA computes  $\varphi^{(k)}, \sigma^{(k)}$ , and  $\eta_i^{(k)}, i = \beta_1, \beta_2, \dots, \beta_c$ , according to Equations (13)–(15).

$$\varphi^{(k)} = \prod_{i=\beta_1}^{\beta_c} (\varphi_i^{(k)})^c \tag{13}$$

$$\sigma^{(k)} = \prod_{i=\beta_1}^{\beta_c} (\sigma_i^{(k)})^c \tag{14}$$

$$\eta^{(k)} = g^{\lambda_i \sum_{j=1}^s m_{i,j}^{(k)}}, i = \beta_1, \beta_2, \dots, \beta_c \tag{15}$$

In this scenario, let us assume that an agent accepts the global model  $Enc(w_{glo})$  and divides the global model parameters into  $n$  blocks, with each block further divided into  $s$  sectors, denoted as  $m(glo)$ . The TA sends the challenge information  $\{\beta_1, \beta_2, \dots, \beta_c\}$  to agent  $k$ . Agent  $k$  generates proof information, which includes  $\varphi^{(glo)}, \sigma^{(glo)}$ , and  $\eta_i^{(glo)}, i = \beta_1, \beta_2, \dots, \beta_c$ , based on the provided challenge information. The TA can verify the correctness of the cloud server’s aggregation parameters by assessing whether Equation (16) holds. For a detailed proof, please refer to Section 4.1. Algorithm 3 presents the pseudo-code for Correctness verification for aggregated results.

$$\prod_{k=1}^K e(\sigma^{(k)}, g) = \prod_{k=1}^K e(\varphi, pk_k) \cdot e(\prod_{i=\beta_1}^{\beta_c} \eta_i^{(glo)}, g^c) \tag{16}$$

---

**Algorithm 3** Correctness verification for aggregated results

---

**Input:**  $Enc(Sparse(wglo))$

**Output:** 1: Correctness verification passed, 0: Correctness verification not passed

- 1: **TA side:**
  - 2: Run random number generator to generate  $\{\beta_1, \beta_2, \dots, \beta_c\}$ .
  - 3: Send challenge information  $\{\beta_1, \beta_2, \dots, \beta_c\}$  to some agent.
  - 4: **Agent side:**
  - 5: Generates proof information  $\varphi^{(glo)}, \sigma^{(glo)}$  and  $\eta_i^{(glo)}, i = \beta_1, \beta_2, \dots, \beta_c$
  - 6: Send  $\varphi^{(glo)}, \sigma^{(glo)}$  and  $\eta_i^{(glo)}$  to the TA.
  - 7: **TA side:**
  - 8: Compute  $e1 = \prod_{k=1}^K e(\sigma^{(k)}, g), e2 = \prod_{k=1}^K e(\varphi^{(k)}, pk_k) \cdot e(\prod_{i=\beta_1}^{\beta_c} \eta_i^{(glo)}, g^c)$
  - 9: If  $e1=e2$
  - 10:     Return 1;
  - 11: else
  - 12:     Return();
  - 13: **End**
- 

4.1.5. Consistency Checking of Aggregated Results

To protect the agents and prevent the cloud server from sending incorrect aggregation results to some agents, the TA performs verification to ensure the consistency of the agents' received parameters. In this context, let us assume that two different agents  $\alpha$  and  $\varepsilon$  receive the aggregated model parameters  $Enc(w_{a,\alpha}^{(glo)})$  and  $Enc(w_{a,\varepsilon}^{(glo)})$ , respectively, sent by the cloud server. Agent  $\alpha$  receives the correct model parameters. It is evident that for the random integers  $\{\beta_1, \beta_2, \dots, \beta_c\}$  generated by the TA within the range between 1 and  $n$ ,  $\varphi_i^{(\alpha)}$  and  $\varphi_i^{(\varepsilon)}$  can be obtained by Equations (17) and (18).

$$\varphi_i^{(\alpha)} = \sum_{j=1}^s H(m_{i,j}^{(\alpha)}), i = 1, 2, \dots, n \tag{17}$$

$$\varphi_i^{(\varepsilon)} = \sum_{j=1}^s H(m_{i,j}^{(\varepsilon)}), i = 1, 2, \dots, n \tag{18}$$

Since agents  $\alpha$  and  $\varepsilon$  receive the same aggregation result,  $\varphi_i^{(\alpha)}$  is equal to  $\varphi_i^{(\varepsilon)}$ . This verification process helps ensure that agents receive consistent and correct model parameters, enhancing the overall security and trustworthiness of the federated learning system.

4.2. Adaptive Gradient Sparsification

To reduce the resource consumption overhead of verification, we introduce an adaptive gradient sparsification method aimed at reducing the number of transmitted parameters. Drawing inspiration from the concept of feedback tuning commonly used in control systems, we devise an adaptive gradient sparsification (AdaGS) method. This method adaptively adjusts the sparsification rate based on the model's performance, thereby striking a better balance between model performance and resource consumption.

In our approach, we employ accuracy values as the metrics for assessing model performance. The industrial agent  $k$  calculates the local accuracy value,  $acc_k$ , and sends it to the cloud server. The cloud server then calculates the average accuracy  $acc_{glo}$  according to the formula:

$$acc_{glo} = \sum_{k=1}^K (\vartheta_k \cdot acc_k) \tag{19}$$

When the average accuracy value  $acc_{glo}$  falls below the highest accuracy for  $\psi$  consecutive times, the system adjusts the sparsification rate  $\rho$  using the following adjustment formula:

$$\rho = \max(\rho - d, 0) \tag{20}$$

where  $d$  represents the decay rate, which controls the extent of sparsification rate adjustment, and  $\psi$  influences the sensitivity of the algorithm. Based on the experiment in this paper, we set  $d$  to 0.001 and  $\psi$  to 2.

The cloud server sends the updated sparsification rate to all agents for use in the next round of model sparsification. After training their local models, each agent applies sparsification based on the received sparsification rate. The importance values of local model parameters at each iteration  $t$  are represented by gradient parameters. These gradient parameters are ranked into a sequence based on their absolute values, sorted from smallest to largest. Selecting the value at the  $\rho\%$  position as the sparsity threshold  $\zeta$ . The parameters with gradient absolute values exceeding  $\zeta$  are chosen to compose the sparse model parameters  $Sparse(w_{a,k})$ . The position of these parameter is recorded to create an index sequence  $Ind_{a,k}^{(t)}$ . Agents upload only the parameters whose importance values exceed the threshold  $\zeta$ , rather than transmitting all model parameters to the cloud server.

In each iteration, the local model parameter recovery algorithm is executed to obtain  $Enc(w_{a,k})$  when the cloud server receives the sparsified model parameter  $Enc(Sparse(w_{a,k}))$  and indexing sequence  $Ind_{a,k}^{(t)}$  from agent  $k$ . For the local model parameter at position  $i$ , the following logic is applied:

If index  $i$  exists in the sequence index  $Ind_{a,k}^{(t)}$ , the model parameter uploaded by agent  $k$  for that indexed position is used.

If index  $i$  is not found in the sequence index  $Ind_{a,k}^{(t)}$ , the parameter at the same position in the global model from the previous iteration is used to fill in the missing value.

Finally, the cloud server computes the global model parameters according to Equation (12). The whole process is shown in Algorithm 4. The AdaGS consists of two communication transfer processes, uploading and downloading. Each local model has  $M$  parameters, and after sparsification,  $(1-\rho)M$  parameters and  $(1-\rho)M$  parameter indexes are uploaded to the server. The cloud server aggregates the global model and returns the  $M$  parameters to each agent. Ranking at each agent  $k$  to obtain the importance threshold  $\zeta$  takes  $O(T \log T)$  time. The computational complexity of Algorithm 4 is  $O(T \log T)$ .

---

#### Algorithm 4 AdaGS

---

**Input:**  $d, \psi$

**Initialize**  $acc_0 \leftarrow 0, \mu \leftarrow 0$

- 1: For  $t = 1, \dots, N$  do:
  - 2:   Each industrial agent  $k = 1, \dots, K$ :
  - 3:   Train the model locally and obtain the model gradient parameters;
  - 4:   Rank the gradient parameters according to their absolute values from smallest to largest;
  - 5:   Select the value at the  $\rho\%$  position as the sparsity threshold  $\zeta$ ;
  - 6:   Select the parameters whose gradient absolute values exceed  $\zeta$  to obtain  $Sparse(w_{a,k}^{(t)})$ ;
  - 7:   Record the position of the parameter whose gradient absolute value exceeds  $\zeta$  to create an index sequence  $Ind_{a,k}^{(t)}$ ;
  - 8:   Encrypt  $Sparse(w_{a,k}^{(t)})$  to obtain  $Enc(Sparse(w_{a,k}^{(t)}))$ ;
  - 9:   Compute the validation accuracy value  $acc_k$ ;
  - 10:   Send  $acc_k^{(t)}, Ind_k^{(t)}$  and  $Enc(Sparse(w_{a,k}^{(t)}))$  to cloud server.
  - 11:   The cloud server:
  - 12:   For  $i = 1, 2, \dots, Np$
  - 13:    If  $i \in Ind_{a,k}^{(t)}$
  - 14:      $w_{a,k}^{(t)} = Enc(Sparse(w_{a,k}^{(t)}))$ .
  - 15:    Else
  - 16:      $w_{a,k}^{(t)} = Enc(w_{glo,i}^{(t-1)})$ .
  - 17:     $Enc(w_{glo}^{(t)}) = \frac{1}{K} \sum_{k=1}^K Enc(w_{a,k}^{(t)})$ .
  - 18:     $acc_{glo}^{(t)} \leftarrow \sum_{k=1}^K (\theta_k \cdot acc_k^{(t)})$ .
  - 19:    If  $acc_{glo}^{(t)} > acc_0$
  - 20:      $acc_0 \leftarrow acc_{glo}^{(t)}$
  - 21:    Else
  - 22:      $\mu \leftarrow \mu + 1$ .
  - 23:     If  $\mu \geq \psi$
  - 24:       $\rho \leftarrow \max(\rho - d, 0)$
  - 25:       $\mu \leftarrow 0$
  - 26:    Send  $\rho$  and  $Enc(w_{glo}^{(t)})$  to agents.
  - 27: **End**
-

### 5. Analysis

In this section, we provide the security and correctness analysis of the proposed scheme.

#### 5.1. Correctness Analysis

The correctness guarantees of our EVFL are: (1) if the parameters uploaded by an agent are intact, the proof of integrity always passes the validation of the TA; and (2) if the cloud server aggregates the parameters correctly, the proof of correctness always passes the validation of the TA.

**Proof.** (1) The integrity of the parameters uploaded by agents depends on (11), which can be derived as follows:

$$\begin{aligned}
 e(sp\sigma^{(k)}, g) &= e\left(\prod_{i=\beta_1}^{\beta_c} (sp\sigma_i^{(k)})^c, g\right) \tag{21} \\
 &= e\left(\prod_{i=\beta_1}^{\beta_c} (\varphi_i^{(k)})^{c \cdot sk_k}, g\right) \cdot e\left(\prod_{i=\beta_1}^{\beta_c} g^{c\lambda_j \sum_{j=1}^s spm_{i,j}^{(k)}}, g\right) \\
 &= e\left(\prod_{i=\beta_1}^{\beta_c} (\varphi_i^{(k)})^c, g^{sk_k}\right) \cdot e\left(\prod_{i=\beta_1}^{\beta_c} g^{\lambda_j \sum_{j=1}^s spm_{i,j}^{(k)}}, g^c\right) \\
 &= e(sp\varphi^{(k)}, pk_k) \cdot e\left(\prod_{i=\beta_1}^{\beta_c} sp\eta^{(k)}, g^c\right)
 \end{aligned}$$

(2) The correctness of the parameters aggregated by the cloud server relies on a checking equation, as shown in (16), which can be proved as follows:

$$\begin{aligned}
 \prod_{k=1}^K e(\sigma^{(k)}, g) &= \prod_{k=1}^K e(\varphi^{(k)}, pk_k) \cdot e\left(\prod_{i=\beta_1}^{\beta} \eta_i^{(k)}, g^c\right) \tag{22} \\
 &= \prod_{k=1}^K e(\varphi^{(k)}, pk_k) \cdot \prod_{k=1}^K e\left(\prod_{i=\beta_1}^{\beta} g^{\lambda_i \sum_{j=1}^s m_{i,j}^{(k)}}, g^c\right) \\
 &= \prod_{k=1}^K e(\varphi^{(k)}, pk_k) \cdot e\left(\prod_{i=\beta_1}^{\beta} g^{\lambda_i \sum_{j=1}^s \sum_{k=1}^K m_{i,j}^{(k)}}, g^c\right) \\
 &= \prod_{k=1}^K e(\varphi^{(k)}, pk_k) \cdot e\left(\prod_{i=\beta_1}^{\beta} \eta_i^{(glo)}, g^c\right)
 \end{aligned}$$

□

#### 5.2. Security Analysis

**Definition 1 (CPA Security).** The private key encryption scheme  $\Pi = (Gen, Enc, Dec)$  satisfies that if for any probabilistic polynomial adversary  $A$ , there exists a negligible function  $negl$  such that

$$\Pr\left[PrivK_{A,\Pi}^{cpa}(n) = 1\right] \leq \frac{1}{2} + negl(n) \tag{23}$$

then indistinguishable encryption under the choice of plaintext attack (CPA Security) is satisfied.

Secure communication protocol based on CKKS encryption is proved to be indistinguishable under chosen plaintext attack. In other words, in the CKKS encryption algorithm, assuming the existence of an adversary  $A$ , even though  $A$  knows the chosen plaintext and the encrypted ciphertext, they still cannot directly deduce the secret key. Next, we demonstrate that our scheme protects the agents' privacy.

**Theorem 1.** In the absence of collusion in the system, our scheme can protect the agent's data privacy and model privacy security if the CKKS encryption algorithm satisfies CPA security.

**Proof.** Assume that an adversary in our system intercepts the ciphertexts of the model parameters, but since the CKKS algorithm satisfies CPA security and there is no collusion between the agents and the adversary in the system, the adversary is unable to obtain the private key to decrypt the parameters. For eavesdroppers, we use TLS/SSL (Transport Layer Security/Secure Sockets Layer) to establish a separate communication channel

between the cloud server and each agent to prevent eavesdropping during parameter transmission. Since each agent uses a separate channel to communicate with the cloud server, each agent cannot access the model parameters of other agents. Therefore, in our system, we can ensure the privacy and security of all industrial agents' data and models. □

### 5.3. Analysis of Communication Cost and Computation Cost

We introduce some new notations to describe the operations in our protocol. Let  $P$  denote a bilinear pairing,  $A$  denote an addition in  $G$ ,  $M$  denote a multiplication in  $G$ ,  $E$  denote an exponentiation in  $G$ ,  $A_z$  denote an addition in  $Z^*$ ,  $M_z$  denote a multiplication in  $Z^*$ ,  $H$  denote a hash computation for different hash functions, and  $U$  denote the number of agents. In addition, the length of an element in  $Z^*$  and  $G$  is represented as  $|Z|$  and  $|g|$ , respectively. The total number of model parameters is denoted by  $NP$ .

- **Communication cost:** According to our protocol, communication costs are mainly incurred in data transmission. During each training round, the industrial agent sends sparsified local model parameters and auxiliary information to the cloud server. This incurs a communication cost of  $U_A \cdot [3(1 - \rho)N_p \cdot |Z_n| + 2(1 - \rho)n \cdot |g_1| + |Z_n|]$ . For integrity verification, the TPA sends a challenge sequence  $\beta = \{\beta_1, \beta_2, \dots, \beta_c\}$  to the cloud server, where the size is  $c \cdot |Z_n|$  bits. The cloud server then sends back a proof of integrity  $\{sp\varphi^{(k)}, sp\sigma^{(k)}, sp\mu_i^{(k)}\}$ , with a size of  $(c + 2) \cdot |g_1|$  bits. After aggregating the local models, the cloud server sends the global model parameters to the agents, which requires  $U_A \cdot N_p \cdot |Z_n|$  bits. When verifying the correctness of the aggregation result, each industrial agent sends auxiliary validation message  $\{\varphi_i^{(k)}, \sigma_i^{(k)}\}$  to the TA, incurring a communication cost of  $2U_A \cdot n \cdot |g_1|$  bits. The TA then transmits a challenge sequence  $\beta = \{\beta_1, \beta_2, \dots, \beta_c\}$  to some agents, where the size is  $c \cdot |Z_n|$  bits. Agents send back a proof of correctness  $\{\varphi^{(glo)}, \sigma^{(glo)}, \eta_i^{(glo)}\}$ , with a size of  $(c + 2) \cdot |g_1|$  bits. Thus, the communication cost per round of training is approximately  $[(4 - 3\rho)U_A \cdot N_p + 2c + 1] \cdot |Z_n| + [2U_A \cdot 2n + 2c + 4] \cdot |g_1|$  bits.
- **Computational cost:** Before uploading local parameters, the industrial agent is required to generate auxiliary validation information to be sent to both the cloud server and the TA. The computational overhead of the auxiliary information generation is  $(1 + \rho)n \cdot s \cdot H + (s - 1) \cdot (1 - \rho)n \cdot A$ . After receiving the challenge message from the TA, the cloud server generates proof information, incurring a computational overhead of  $c[3E + (s - 1) \cdot A + M_Z] + 2(c - 1)M$ . The TA then validates the integrity, and the computation overhead for this validation is approximately  $n(1 - \rho) \cdot [(s - 1) \cdot A + s \cdot m + 2E + M]$ . When performing the aggregation results' correctness validation, the industrial agent generates proof information, incurring computation costs approximately equal to  $n \cdot s \cdot H + (s - 1) \cdot n \cdot A$ . Then, the TA validates the correctness, and the computation overhead is close to  $3P + (c - 1) \cdot M_Z + E$ .

From Table 1, it is evident that the computational cost encompasses various operations, including power, hash, addition, and multiplication operations. Among these, the hash operation is the most time-consuming, followed by power, multiplication, and pairwise operations, with the remaining operations being less time-consuming, as noted in reference [37]. Due to the use of fewer parameters, the communication and computation overhead of our scheme costs less.

**Table 1.** Lists the resource overhead of some existing schemes.

Cost Type	Quantity	EVFL	The State of the Art [38]
Communication cost	Data transmission	$[(4 - 3\rho)U_A \cdot N_p + 2c + 1] \cdot  Z_n  + [2U_A \cdot n + 2(1 - \rho) + 2c + 4] \cdot  g_1 $	$(2U_A \cdot N_p + 2c) \cdot  Z_n  + 2(c + 2) \cdot  g_1 $

Table 1. Cont.

Cost Type	Quantity	EVFL	The State of the Art [38]
Computation cost	Computation cost on auxiliary information generation	$(1 - \rho)nsH + (s - 1) \cdot (1 - \rho)nA$	$nsH + 2nE + nM_z + 2sA + sM_z$
Computation cost	Computation cost on proof information generation	$c[3E + (s - 1)A + M_z] + 2(c - 1)M$	$cs(A + M_z) + 2(c - 1)M + 3cE + 2cM$
Computation cost	Computation cost of validation	$3P + (c - 1)M_z + E$	$3P + (c + 1)M_z$

## 6. Implementation and Evaluation

### 6.1. Experiment Settings and Evaluation Standards

We conducted our experiments on a computer equipped with an Intel i7-6550 CPU, NVIDIA 1080-Ti GPU, and 16 GB of RAM. Our implementation used Python for the EVFL model and CKKS-Python for implementing CKKS homomorphic encryption.

To evaluate the performance of the EVFL model, we employed two datasets: Fashion-MNIST (F-MNIST) and KDD CUP 99. F-MNIST is a widely recognized benchmark dataset commonly used in various machine learning tasks. On the other hand, the KDD CUP 99 dataset originates from packet traces collected in military network environments and is a prevalent dataset in the field of Industrial Internet of Things (IIoT) intrusion detection. Thus, we chose these two datasets to thoroughly assess the benchmark performance of our approach.

Here are the details regarding the number of data records and characteristics in the different datasets:

- Fashion-MNIST(F-MNIST) [39]: there are 60,000 training samples and 10,000 test samples in this dataset, and the samples can be categorized into 10 classes (sandals, shirts, sneakers, bags, etc.), each with grayscale images with an input size of  $28 \times 28 \times 1$ .
- KDD CUP 99 [40]: This dataset consists of 5 million records, and we used a 10% training subset and a test set for our experiments. The training set had 494,021 training samples, and the test set had 311,092 test samples. The dataset included 41 features, and there were 24 types of attacks in the training set and 38 types of attacks in the test set. The types of attacks can be classified as denial-of-service (DoS), attack from remote to local machine (R2L), unauthorized access to local administrator user (U2R), and four types of probing attacks.

We used the CNN model as the local model of industrial agents. The structure of the CNN model trained on the F-MNIST dataset was  $28 \times 28$  (input)– $5 \times 5 \times 16$  (conv, one stride, one padding)–(relu)– $2 \times 2$  (max\_pooling, one stride)– $5 \times 5 \times 32$  (conv, one stride, one padding)–(relu)– $2 \times 2$  (max\_pooling, one stride)–(flatten)–64 (dense)–(relu)–10 (output). The structure of the CNN model trained on KDD CUP 99 dataset was  $5 \times 5$  (input)– $3 \times 3 \times 16$  (conv, one stride, two padding)–(batchnorm)–(relu)– $2 \times 2$  (max\_pooling,1stride)– $3 \times 3 \times 32$  (conv, one stride, two padding)–(batchnorm)–(relu)– $2 \times 2$  (max\_pooling,1stride)–(flatten)–(output). We trained the model using an MGD optimizer with a momentum rate of 0.5. The loss function was a cross-entropy cost function. The mini-batch size, the aggregation round, the learning rate, and the decay rate  $d$  used to train the network were 512, 1500, 0.05, and 0.001, respectively.

The accuracy, precision, recall, F1-score, and compression rate (CR) were used as the evaluation standards. The CR is the proportion of the transmission parameter, which is calculated according to  $CR = \frac{\sum_{i=1}^N \sum_{k=1}^K |sparse(w_i^{a,k})|}{\sum_{i=1}^N \sum_{k=1}^K |w_i^{a,k}|}$ , where  $N$  represents the aggregation rounds.

### 6.2. The Model Performance of the EVFL

In this experiment, we chose the centralized learning (CL) method and the privacy-preserving federated learning (PFL) [41] method to compare with our EVFL method.

- Accuracy: Early-stage observations in Figure 5a,d reveal that EVFL exhibits lower accuracy compared to CL and FL. This initial discrepancy can be attributed to EVFL’s utilization of a model sparsification algorithm. As the number of aggregation rounds increases, the accuracy curve of EVFL steadily improves. By the end of the training process, its accuracy approaches that of PFL. Detailed experimental results are presented in Table 2, highlighting the similarity in model performance between EVFL and PFL. These findings emphasize that EVFL’s introduction does not adversely impact model performance across diverse datasets.
- Resource cost: Figure 5c,f show transmission parameter curves for different datasets, while Table 2 provides comprehensive experimental data. The quantity of transmission parameters plays a pivotal role in determining computation and communication overhead. As evident from Figure 5c,f, the number of transmission parameters increases notably with the growing number of iteration rounds, varying across different datasets. From Table 2, we can see that compared with PFL, EVFL demonstrates an impressive reduction in transmission parameters. Specifically, on the Fashion-MNIST and KDD CUP 99 datasets, EVFL reduces transmission parameters by a factor of 8.94 and 8.92, respectively. This outcome serves as a testament to the effectiveness of our AdaGS method in curtailing resource consumption.

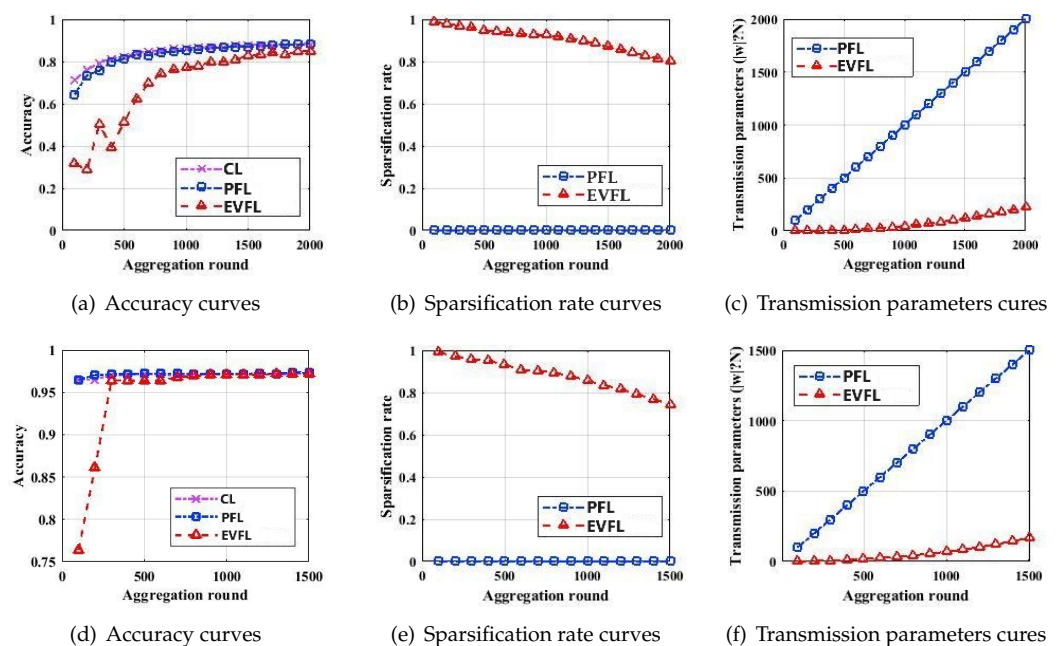


Figure 5. Curves of the different models; (a–c) were trained on Fashion-MNIST; (d–f) were trained on KDD CUP 99.

Table 2. Classification results of the different models.

Dataset	Fashion-MNIST				KDD CUP 99			
	Accuracy	Precision	Recall	CR	Accuracy	Precision	Recall	CR
CL	88.49%	88.49%	88.44%	–	97.36%	97.29%	97.36%	–
PFL	88.36%	88.46%	88.36%	1x	97.32%	96.02%	97.32%	1x
EVFL	84.92%	84.73%	84.92%	8.94x	97.13%	97.34%	97.13%	8.92x

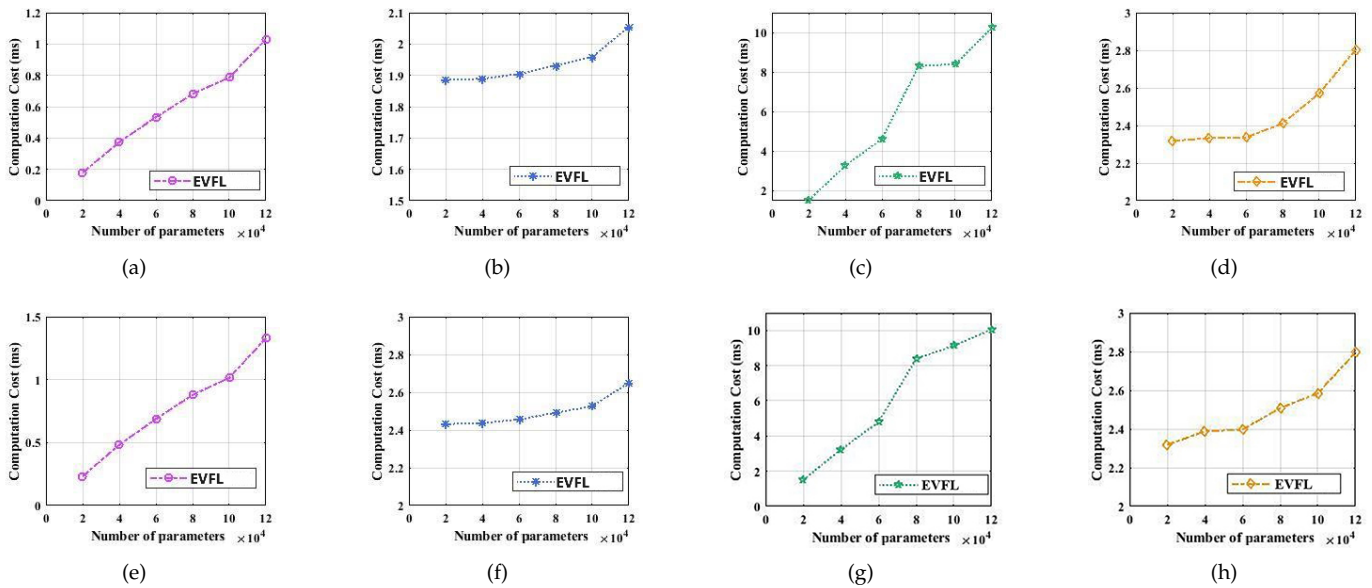


### 6.3. The Computation Cost of Proof Generation and Verification

In this section, we conducted experiments to assess the computational cost associated with proof generation and verification. We explored two specific aspects.

#### 6.3.1. Varying the Number of Parameters

First, we tested the time cost of proof generation and validation with varying parameter quantities. In this experiment, 10 industrial agents were employed, and they utilized 60% of the available parameters. Testing was conducted on both the F-MNIST and KDD CUP 99 datasets. Figure 6 provides a graphical representation of the results. As shown in Figure 6, the time cost for proof generation and validation increased as the number of parameters grew. However, there was a slight variation in computational overhead for integrity and correctness verification. To illustrate, using the F-MNIST dataset as an example, when the parameter count ranged from  $2 \times 10^4$  to  $12 \times 10^4$ , the time required for parameter integrity validation increased from 1.885 ms to 1.904 ms, while the time for correctness validation of aggregation results increased from 2.318 ms to 2.795 ms. These results show that the time cost for integrity and correctness verification remains relatively stable with a fixed number of agents and varying numbers of parameters.



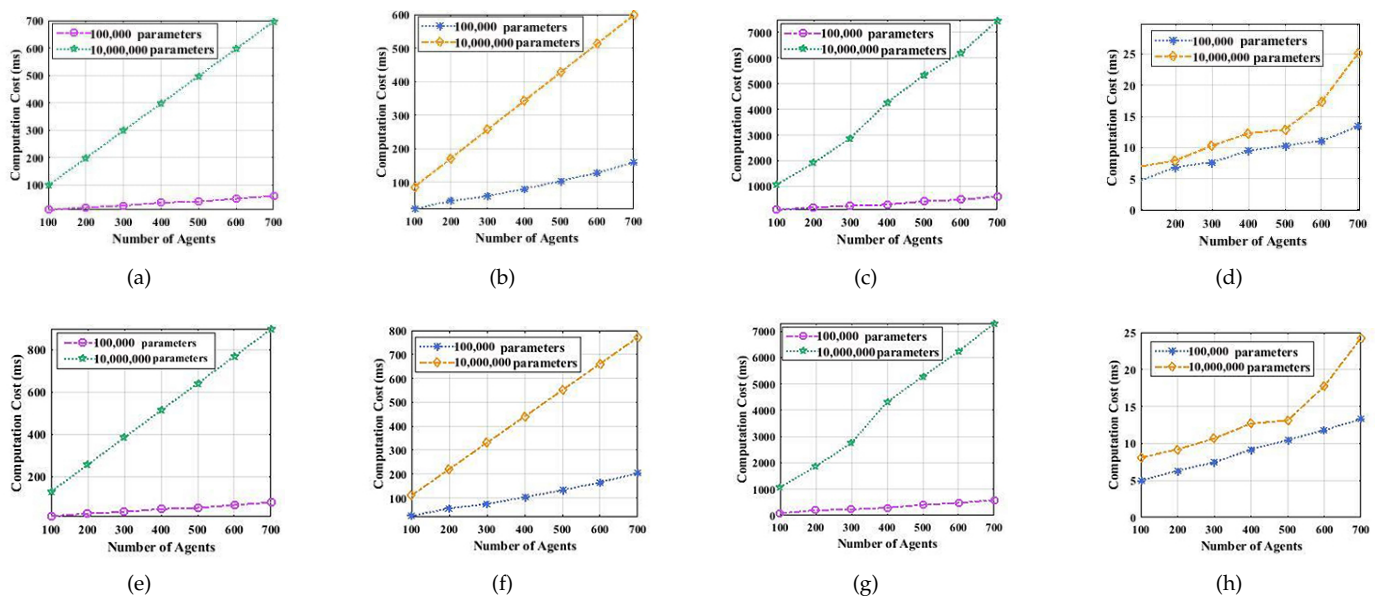
**Figure 6.** Time required to generate proofs and validations with different numbers of parameters, with 10 agents and 60% of the parameters used; (a,e) Time required for the parameter integrity proof generation; (b,f) Time required for the parameter integrity verification; (c,g) Time required for the aggregated results' correctness proof generation; (d,h) Time required for the aggregated results' correctness verification; (a–d) were tested on Fashion- MNIST; (e–h) were tested on KDD CUP 99.

#### 6.3.2. Varying the Number of Agents

Additionally, we explored the time cost variations for proof generation and verification as the number of industrial agents changed. These experiments were conducted using 60% of the parameters, involving 104 and 106 parameters on the Fashion-MNIST and KDD CUP 99 datasets, respectively. Figure 7 illustrates the outcomes. As shown in Figure 7, the time cost of proof generation and validation increased with the increase in the number of industrial agents. Moreover, the time cost was more sensitive to changes in the number of industrial agents when dealing with 106 parameters compared to cases with 104 parameters. This sensitivity aligned with theoretical expectations, as proof generation and verification were performed independently by each industrial agent.

In summary, our experimental results are in line with prior research, as reported in reference [23,38]. Notably, our approach, which integrates the AdaGS algorithm to

reduce the number of uploaded parameters, exhibits a lower time cost for proof generation and validation when compared to certain existing schemes. This highlights the practical advantages of our method in terms of computational efficiency.



**Figure 7.** Time required to generate proofs and validations with different numbers of agents, with 60% of the parameters used; (a,e) Time required for the parameter integrity proof generation; (b,f) Time required for the parameter integrity verification; (c,g) Time required for the aggregated results’ correctness proof generation; (d,h) Time required for the aggregated results’ correctness verification; (a–d) were tested on Fashion-MNIST; (e–h) were tested on KDD CUP 99.

### 7. Conclusions

In this paper, an efficient and verifiable federated learning framework with privacy protection (EVFL) for the industrial internet was introduced. EVFL integrates three key capabilities: privacy protection, verification mechanism, and resource optimization. Firstly, in terms of privacy protection, it employs full homomorphic encryption technology. A comprehensive security analysis confirmed that EVFL effectively safeguarded the privacy of agent data and enabled secure verification in federated learning. Secondly, regarding the verification mechanism, by incorporating BLS signatures, EVFL supports verifying the integrity of uploaded parameters from industrial agents, the correctness of aggregation results from cloud servers, and the consistency of distributed results from cloud servers. Thirdly, in terms of resource optimization, through the integration of our meticulously designed adaptive gradient sparsification method, EVFL significantly reduces resource consumption. Both theoretical analysis and experimental results demonstrated the effectiveness of EVFL.

In the future, our goal is to leverage new technologies such as large language models (LLMs) to delve deeper into identifying untrusted participants within FL systems, promptly detecting data poisoning and model poisoning attacks, thereby further enhancing the security and reliability of this approach.

**Author Contributions:** Methodology, J.W., C.W., C.C. and C.Z.; formal analysis, J.J. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by National Science and Technology Major Project of China (No. 2021YFB0300104), the 2024 Key R&D Program of Zhejiang Province, China (No. 2024C01212).

**Data Availability Statement:** Data are contained within the article.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

Symbol	Quantity
$G, G_1$	Cyclic multiplicative groups with prime order $p$
$g$	The generator of $G$
$e$	A bilinear map
$H$	A hash function
$Enc(w)$	Encrypted model parameters
$\rho$	Sparsification rate
$D_{batch,k}$	The mini-batch data of the agent $k$
$sk$	The privacy key of the industrial agent
$pk$	The public key of the industrial agent
$ID$	The unique identifier of the industrial agent
$Sparse(w)$	Sparsified model parameters
$Enc(Sparse(w))$	Encrypted sparsified model parameters
$acc_{a,k}$	Accuracy value
$d$	The decay rate
$t$	Number of iterations
$CR$	Compression rate

## References

- Zhu, H.X.; Wu, C.K. Smart Healthcare in the Era of Internet-of-Things. *IEEE Consum. Electron. Mag.* **2019**, *8*, 26–30. [\[CrossRef\]](#)
- Ghahramani, M.; Qiao, Y.; Zhou, M.C.; O'Hagan, A.; Sweeney, J. AI-based modeling and data-driven evaluation for smart manufacturing processes. *IEEE/CAA J. Autom. Sin.* **2020**, *7*, 1026–1037. [\[CrossRef\]](#)
- Dileep, G. AI-based modeling and data-driven evaluation for smart manufacturing processes. *Renew Energy* **2020**, *146*, 2589–2625. [\[CrossRef\]](#)
- Feng, X.Z.; Wu, J.; Wu, Y.L.; Li, J.H.; Yang, W. Blockchain and digital twin empowered trustworthy self-healing for edge-AI enabled industrial Internet of things. *Inf. Sci.* **2023**, *642*. [\[CrossRef\]](#)
- Li, H.; Li, C.C.; Wang, J.; Yang, A.; Ma, Z.Z.; Zhang, Z.Q.; Hua, D.B. Review on security of federated learning and its application in healthcare. *Future Gener. Comput. Syst.* **2023**, *144*, 271–290. [\[CrossRef\]](#)
- Djenouri, Y.; Michalak, T.P.; Lin, J. Federated deep learning for smart city edge-based applicationst. *Future Gener. Comput. Syst.* **2023**, *147*, 350–359. [\[CrossRef\]](#)
- Tong, Z.; Wang, J.K.; Mei, J.; Li, K.L.; Li, W.B.; Li, K.Q. Multi-type task offloading for wireless Internet of Things by federated deep reinforcement learning. *Future Gener. Comput. Syst.* **2023**, *145*, 536–549. [\[CrossRef\]](#)
- Lin, Z.L.; Li, J.G. FedEVCP: Federated Learning-Based Anomalies Detection for Electric Vehicle Charging Pile. *Comput. J.* **2023**, *67*, 1521–1530. [\[CrossRef\]](#)
- Al-Huthaifi, R.; Li, T.R.; Huang, W.; Gu, J.; Li, C.S. Federated learning in smart cities: Privacy and security survey. *Comput. J.* **2023**, *632*, 833–857. [\[CrossRef\]](#)
- Zeng, X.J.; Yu, Z.P.; Zhang, W.S.; Wang, X.; Lu, Q.H.; Wang, T.; Gu, M.; Tian, Y.L.; Wang, F.Y. Homophily Learning-Based Federated Intelligence: A Case Study on Industrial IoT Equipment Failure Prediction. *IEEE Internet Things* **2023**, *10*, 7356–7365. [\[CrossRef\]](#)
- Zhang, P.Y.; Chen, N.; Li, S.B.; Choo, K.R.; Jiang, C.X.; Wu, S. Multi-Domain Virtual Network Embedding Algorithm Based on Horizontal Federated Learning. *IEEE Trans. Inf. Forensics Secur.* **2023**, *18*, 3363–3375. [\[CrossRef\]](#)
- Wang, K.I.; Zhou, X.K.; Liang, W.; Yan, Z.; She, J.H. Federated Transfer Learning Based Cross-Domain Prediction for Smart Manufacturing. *IEEE Trans. Ind. Informatics* **2022**, *18*, 4088–4096. [\[CrossRef\]](#)
- Fisichella, M.; Lax, G.L.; Russo, A. Partially-federated learning: A new approach to achieving privacy and effectiveness. *Inform. Sci.* **2022**, *614*, 534–547. [\[CrossRef\]](#)
- Zhang, J.M.; Ye, A.; Chen, J.W.; Zhang, Y.X.; Yang, W.J. CSFL: Cooperative Security Aware Federated Learning Model Using The Blockchain. *Comput. J.* **2023**, *67*, 1298–1308. [\[CrossRef\]](#)
- Wang, Z.B.; Song, M.K.; Zhang, Z.F.; Song, Y.; Wang, Q.; Qi, H.R. Beyond Inferring Class Representatives: User-Level Privacy Leakage from Federated Learning. In Proceedings of the IEEE Conference on Computer Communications (IEEE INFOCOM 2019), Paris, France, 29 April–2 May 2019; pp. 2512–2520.
- Wei, K.; Li, J.; Ding, M.; Ma, C.; Yang, H.H.; Farhad, F.; Jin, S.; Poor, H.V. Federated Learning with Differential Privacy: Algorithms and Performance Analysis. *IEEE Trans. Inf. Forensics Secur.* **2019**, *15*, 3454–3469. [\[CrossRef\]](#)
- Kairouz, P.; Liu, Z.Y.; Steinke, T. The Distributed Discrete Gaussian Mechanism for Federated Learning with Secure Aggregation. *Int. Conf. Mach. Learn.* **2021**, *139*, 5201–5212.
- Moriai, S. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. In Proceedings of the 2019 IEEE 26th Symposium on Computer Arithmetic (ARITH), Kyoto, Japan, 10–12 June 2019; p. 198.

19. Zhang, Z.H.; He, N.X.; Li, Q.D.; Wang, K.S.; Gao, H.; Gao, T.G. DetectPMFL: Privacy-Preserving Momentum Federated Learning Considering Unreliable Industrial Agents. *IEEE Trans. Ind. Informatics* **2022**, *18*, 7696–7706. [[CrossRef](#)]
20. Ren, Y.L.; Li, Y.R.; Feng, G.R.; Zhang, X.P. Privacy-Enhanced and Verification-Traceable Aggregation for Federated Learning. *IEEE Internet Things J.* **2022**, *9*, 24933–24948. [[CrossRef](#)]
21. Xu, G.W.; Li, H.W.; Liu, S.; Yang, K.; Lin, X.D. VerifyNet: Secure and Verifiable Federated Learning. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 911–926. [[CrossRef](#)]
22. Zhang, X.L.; Fu, A.; Wang, H.Q.; Zhou, C.Y.; Chen, Z.Z. A Privacy-Preserving and Verifiable Federated Learning Scheme. In Proceedings of the ICC 2020–2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020.
23. Fu, A.; Zhang, X.L.; Xiong, N.X.; Gao, Y.S.; Wang, H.Q.; Zhang, J. VFL: A Verifiable Federated Learning With Privacy-Preserving for Big Data in Industrial IoT. *IEEE Trans. Ind. Informatics* **2022**, *18*, 3316–3326. [[CrossRef](#)]
24. Lin, L.; Zhang, X.Y. PPVerifier: A Privacy-Preserving and Verifiable Federated Learning Method in Cloud-Edge Collaborative Computing Environment. *IEEE Internet Things J.* **2023**, *10*, 8878–8892. [[CrossRef](#)]
25. Yang, Z.; Zhou, M.; Yu, H.Y.; Sinnott, R.; Liu, H.P. Efficient and Secure Federated Learning With Verifiable Weighted Average Aggregation. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 205–222. [[CrossRef](#)]
26. Li, Z.Y.; Mei, X.S.; Sun, Z.; Xu, J.; Zhang, J.C.; Zhang, D.W.; Zhu, J.Y. A reference framework for the digital twin smart factory based on cloud-fog-edge computing collaboration. *J. Intell. Manuf.* **2024**. [[CrossRef](#)]
27. Yang, W.; Xiang, W.; Yang, Y.; Cheng, P. Optimizing Federated Learning With Deep Reinforcement Learning for Digital Twin Empowered Industrial IoT. *IEEE Trans. Ind. Informatics* **2023**, *19*, 1884–1893. [[CrossRef](#)]
28. Zhang, J.Z.; Luo, C.B.; Carpenter, M.; Min, G.Y. Federated Learning for Distributed IIoT Intrusion Detection Using Transfer Approaches. *IEEE Trans. Ind. Informatics* **2023**, *19*, 8159–8169. [[CrossRef](#)]
29. Wang, X.; Garg, S.; Lin, H.; Hu, J.; Kaddoum, G.; Piran, M.J.; Hossain, M.S. Toward Accurate Anomaly Detection in Industrial Internet of Things Using Hierarchical Federated Learning. *IEEE Internet Things J.* **2022**, *9*, 7110–7119. [[CrossRef](#)]
30. Han, G.; Zhang, T.T.; Zhang, Y.H.; Xu, G.W.; Sun, J.F.; Cao, J. Verifiable and privacy preserving federated learning without fully trusted centers. *J. Ambient. Intell. Humaniz. Comput.* **2022**, *13*, 1431–1441. [[CrossRef](#)]
31. Han, P.C.; Wang, S.Q.; Leung, K. Adaptive Gradient Sparsification for Efficient Federated Learning: An Online Learning Approach. In Proceedings of the 2020 IEEE 40TH International Conference on Distributed Computing Systems (ICDCS), Singapore, 29 November–1 December 2020; pp. 300–310.
32. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y.S. *Homomorphic Encryption for Arithmetic of Approximate Numbers*; Springer: Cham, Switzerland, 2017.
33. Rivest, R. On databanks and privacy homomorphism. *Found. Secur. Comput.* **1978**, *4*, 169–180.
34. Mohammed, S.J.; Taha, D.B. Performance Evaluation of RSA, ElGamal, and Paillier Partial Homomorphic Encryption Algorithms. In Proceeding of the 2nd 2022 International Conference on Computer Science and Software Engineering (CSASE 2022), Duhok, Iraq, 15–17 March 2022; pp. 89–94.
35. Ou, W.; Zeng, J.H.; Guo, Z.J.; Yan, W.G.; Liu, D.W.; Fuentes, S. A Homomorphic-encryption-based Vertical Federated Learning Scheme for Risk Management. *Comput. Sci. Inf. Syst.* **2020**, *17*, 819–834. [[CrossRef](#)]
36. Fan, Y.K.; Bai, J.R.; Lei, X.; Zhang, Y.Q.; Zhang, B.; Li, K.C.; Tan, G. Privacy preserving based logistic regression on big data. *J. Netw. Comput. Appl.* **2020**, *171*, 102769. [[CrossRef](#)]
37. Li, X.; Liu, S.P.; Lu, R.X.; Khan, M.K.; Gu, K.; Zhang, X.S. An Efficient Privacy-Preserving Public Auditing Protocol for Cloud-Based Medical Storage System. *IEEE J. Biomed. Health Inform.* **2022**, *26*, 2020–2031. [[CrossRef](#)] [[PubMed](#)]
38. Gao, H.; He, N.X.; Gao, T.G. VeriFL: Successive verifiable federated learning with privacy-preserving. *Inf. Sci.* **2023**, *622*, 98–114. [[CrossRef](#)]
39. Seo, Y.; Shin, K.S. Hierarchical convolutional neural networks for fashion image classification. *Expert Syst. Appl.* **2019**, *116*, 328–339. [[CrossRef](#)]
40. Sahu, S.K.; Sarangi, S.; Jena, S.K.. A detail analysis on intrusion detection datasets. In Proceedings of the 2014 IEEE international advance computing conference (IACC), Gurgaon, India, 21–22 February 2014.
41. Lohana, A.; Rupani, A.; Rai, S.; Kumar, A. Efficient Privacy-Aware Federated Learning by Elimination of Downstream Redundancy. *IEEE Des. Test* **2021**, *39*, 73–81. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.