Ripple: Private Collaborative Learning Against Model Poisoning via Zero-Knowledge Proofs for Non-Linear Functions

Xiaochang Yu, Meng Shen, *Member, IEEE*, Qi Li, *Senior Member, IEEE*, Jin Dong, Xiangyun Tang, *Member, IEEE*, Gerui Wang, Ke Xu, *Fellow, IEEE*, and Liehuang Zhu, *Senior Member, IEEE*

***Abstract*—Private collaborative learning allows a model owner to train neural networks using data from distributed data owners, while preserving both data and model privacy. Recent frameworks achieve this by leveraging secure multiparty computation (MPC) to enable private training across multiple parties. However, existing approaches assume semi-honest data owners and fail to defend against model poisoning attacks, where malicious data owners can inject forged updates to degrade model performance. In this paper, we propose Ripple, a private collaborative learning framework that defends against model poisoning attacks via zero-knowledge proofs for non-linear functions. We reduce multi-party collaborative learning to a two-party computation paradigm between the model owner and each data owner. To ensure both the verifiability and privacy of inputs to non-linear functions, we design secure protocols that integrate removable noise mechanisms with Paillier-based zero-knowledge proofs. These protocols allow the model owner to verify the correctness of data owners' computations without gaining access to their private inputs. Our theoretical analysis provides the security guarantees of Ripple, and empirical evaluations demonstrate its ability to defend against model poisoning attacks. Specifically, under the Min-Max attack with non-IID data, existing methods suffer from accuracy degradation of up to 59%, whereas Ripple consistently limits the degradation to within 4%. Moreover, the accuracy of the model is not affected by the data distribution or the proportion of malicious data owners.**

***Index Terms*—Collaborative learning, model poisoning attack, privacy protection, zero-knowledge proof.**

I. Introduction

COLLABORATIVE learning has emerged as a prominent paradigm in machine learning (ML), enabling a model owner (MO) to train high-quality models by leveraging data from multiple data owners (DOes) [1]. A canonical application of collaborative learning is disease diagnosis, where medical institutions collaboratively train diagnostic models using data from geographically distributed hospitals. By leveraging large and diverse datasets, collaborative learning can significantly improve model accuracy [2].

With an increasing emphasis on data and model privacy, *private collaborative learning* has been proposed to address these concerns. Specifically, *data privacy* aims to keep the training data of DOes confidential and prevents the MO from reconstructing data from model updates (e.g., by launching gradient matching attacks [3], [4]). *Model privacy* aims to preserve the MO's exclusive ownership of the post-training model [1], [5]. Federated learning (FL) is a widely adopted paradigm where DOes train models locally while a MO aggregates their updates. However, the distribution mechanism of the model compromises model privacy [5], [6]. To address this, recent work has explored MPC-based frameworks that allow DOes to share data with multiple servers [7], [8], [9], [10], [11], [12], [13]. As summarized in Table I, these approaches require a non-colluding assumption, as even two colluding servers can break the protocol. Pencil avoids this assumption but relies on a critical semi-honest assumption for DOes, requiring them to strictly follow the protocol [1]. In practice, however, malicious DOes can launch *model poisoning attacks* by submitting forged updates, significantly degrading model accuracy [5], [6], [14], [15]. Therefore, private collaborative learning frameworks must maintain model accuracy when malicious DOes are involved.

In this paper, we present Ripple *, a private collaborative learning framework that defends against model poisoning*

Received 23 April 2026; revised 24 June 2026; accepted 5 August 2026. Date of publication 11 August 2026; date of current version 10 November 2026. This work was supported part by the National Key R&D Program of China under Grant 2023YFB2703800, in part by the Beijing Natural Science Foundation Project under Grant Z260008, in part by NSFC Projects under Grant U23A20304 and Grant U25A20428, and in part by the Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing. *(Corresponding author: Meng Shen.)*

Xiaochang Yu, Meng Shen, and Liehuang Zhu are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China (e-mail: 3220242048@bit.edu.cn; shenmeng@bit.edu.cn; liehuangz@bit.edu.cn).

Qi Li is with the Institute for Network Sciences and Cyberspace, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing 100084, China, and also with Zhongguancun Laboratory, Beijing 100094, China (e-mail: qli01@tsinghua.edu.cn).

Jin Dong and Gerui Wang are with the Beijing Academy of Blockchain and Edge Computing, Beijing 100086, China (e-mail: dongjin@baec.org.cn; wanggerui@baec.org.cn).

Xiangyun Tang is with the School of Information Engineering and Key Laboratory of Ethnic Language Intelligent Analysis and Security Governance of MOE, Minzu University of China, Beijing 100081, China (e-mail: xiangyunt@muc.edu.cn).

Ke Xu is with the Department of Computer Science and Technology, Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing 100084, China, and also with Zhongguancun Laboratory, Beijing 100094, China (e-mail: xuke@tsinghua.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TDSC.2026.3722647, provided by the authors.

Digital Object Identifier 10.1109/TDSC.2026.3722647

1545-5971 © 2026 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

TABLE I
A COMPARATIVE SUMMARY OF REPRESENTATIVE COLLABORATIVE LEARNING METHODS

Category	Methods	Techniques	Data Privacy	Model Privacy	Against Model Poisoning Attacks	Against collusion
FL	Median [16], Krum [17], FLTrust [6]	ED, CS	×	×	✓	✓
MPC (4 servers)	Trident [18], Fantastic four [19], Tetrad [20]	GC, SS	✓	✓	×	×
MPC (3 servers)	ABY3 [21], BLAZE [11]	GC, SS	✓	✓	×	×
MPC (2 servers)	QUOTIENT [7], PDLM [9], ABY2.0 [22]	GC, SS, PHE	✓	✓	×	×
MPC (1 server)	Pencil [1]	FHE, SS, DP	✓†	✓	×	✓
MPC (1 server)	Ripple	PHE, OT, SS, ZKP	✓	✓	✓	✓

ED is for euclidean distance, CS for cosine similarity, SS for secret sharing, GC for garbled circuits, OT for oblivious transfer, FHE for fully homomorphic encryption, PHE for partially homomorphic encryption, and DP for differential privacy.
† In Pencil, DO adds noise to the gradient for data privacy, balancing privacy and model accuracy. The experimental results are presented in Section VII-D.

attacks via zero-knowledge proofs for non-linear functions. We simplify multi-party collaborative learning into two-party MO–DO sub-protocols; the interaction between the MO and one DO is representative of that with every other DO [1]. A key challenge lies in verifying the correctness of the DO's computations while preserving model privacy during interactive non-linear operations, particularly for complex non-linear functions in model training. To overcome this challenge, we develop *secure and verifiable computation protocols* for non-linear computations, integrating removable noise for model privacy and Paillier's zero-knowledge proofs (ZKPs) to validate the authenticity of DOes' computations.

More specifically, in Ripple, each DO encrypts its local dataset using Paillier [22] before sharing it with the MO. We divide training into linear (e.g., fully connected, convolutional) and non-linear (e.g., activation functions, max pooling) computations. For linear operations, the MO performs computations directly on ciphertexts by utilizing the homomorphic properties of Paillier [22]. Non-linear computations, which cannot be performed directly over encrypted data, require interaction between the MO and the DO holding the private key. We first develop two verifiable and secure two-party protocols (MO/DO) for fundamental gradient operations, including ciphertext comparison and multiplication. The MO first injects carefully designed noise into the input ciphertexts to safeguard model privacy. The DO then decrypts the noisy ciphertexts, performs computations on plaintexts, and generates ZKPs to attest to the correctness of its results. The MO verifies these proofs and employs a tailored mechanism to remove the injected noise, thereby recovering the correct output. Any computations that fail verification are rejected and discarded before aggregation. Building upon these protocols, we further design protocols to extend support for commonly used non-linear layers, such as activation functions and max pooling. Since our ZKP schemes are built on Paillier's specific mathematical structure, other methods (e.g., Pencil [1], which uses the scheme BFV) cannot directly achieve verifiability by integrating our protocols.

As a result, utilizing the above linear and non-linear computations, the MO can obtain encrypted gradients from each DO's dataset. These encrypted gradients are aggregated using the homomorphic properties of the ciphertext. Once aggregation is complete, the MO and DOes jointly decrypt the aggregated gradients via Threshold Paillier [23], a threshold decryption algorithm. This algorithm guarantees that only the MO possesses the plaintext of the final aggregated gradient.

The main contributions are summarized as follows:

- We propose Ripple, a private and verifiability-enhanced collaborative learning framework that safeguards both data and model privacy while supporting up to $K-2$ malicious DOes launching model poisoning under various data distributions.
- We design a suite of secure two-party (MO/DO) computation protocols for non-linear operations. Unlike prior MPC-based frameworks that only ensure privacy, our protocols integrate Paillier's zero-knowledge proofs to detect and reject forged computations submitted by malicious DOes.
- We provide rigorous theoretical security analysis, proving that Ripple protects both data and model privacy against semi-honest MO and DOes. Additionally, it ensures data privacy even in the presence of collusion attacks involving a semi-honest MO and up to $K-2$ semi-honest DOes.
- We evaluate Ripple's performance against model poisoning attacks and the gradient matching attack targeting data privacy. Under model poisoning including Trim [24], Krum [24], FLTrust [6], and Min-Max [25], Ripple limits accuracy degradation to less than 4% compared to plaintext training without attacks. Under gradient matching attack, Ripple reduces the inference success rate to near-random (approximately 13% in ten-class classification tasks).

II. PRELIMINARIES

In this section, we introduce the preliminaries used in this paper.

A. Notations

Vectors and matrices are denoted by boldfaced Latin letters (e.g., $\mathbf{W}$, $\mathbf{x}$), while scalars are denoted by italic Latin letters (e.g., z). We use $[x]$ to denote the encrypted value of x. Notations with a tilde indicate perturbed parameters, e.g., $\widetilde{\mathbf{W}}$. Unless otherwise specified, a function (e.g., ReLU) applied to a tensor is applied element-wise. We denote standard scalar multiplication (e.g., wx) and matrix/vector multiplication (e.g., $\mathbf{AB}$, $\mathbf{Ax}$) using juxtaposition. The element-wise (Hadamard) product is denoted by $\odot$.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

B. Distributed Collaborative Learning

Distributed Collaborative Learning enables multiple DOes to jointly train a global model without sharing their raw datasets. In a typical distributed framework, let K be the total number of DOes, where the i-th DO possesses a local private dataset $\mathcal{D}_i$. The objective is to minimize the global loss function $F(\mathbf{W})$ by optimizing the local objectives:

$$F(\mathbf{W}) = \sum_{i=1}^{K} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} f_i(\mathbf{W}),$$

where $|\mathcal{D}| = \sum |\mathcal{D}_i|$ denotes the total volume of data, and $f_i(\mathbf{W})$ is the local loss function on $\mathcal{D}_i$. The training process typically involves iterative rounds. In each round t, each DO performs local computations to obtain a local update $\Delta \mathbf{w}_i^t$ and uploads it back to the MO for global aggregation, typically using the FedAvg algorithm:

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \eta \sum_{i=1}^{K} \alpha_i \Delta \mathbf{w}_i^t,$$

where η is the learning rate and α_i represents the aggregation weight assigned to the i-th DO [26], [27].

However, the decentralized nature and lack of centralized oversight on local computations introduce a significant vulnerability known as model poisoning attacks. Since the central MO cannot audit the raw data or the execution integrity of local training, a malicious DO m can manipulate the training process by submitting a forged update $\Delta \mathbf{w}_m^*$ instead of the true gradient. The attacker's goal is often to maximize the global loss or steer the model towards a specific incorrect prediction, which can be expressed as:

$$\max_{\Delta \mathbf{w}_m^*} \mathcal{L} \left(\mathbf{W}_t + \alpha_m \Delta \mathbf{w}_m^* + \sum_{j \neq m} \alpha_j \Delta \mathbf{w}_j, \mathcal{D}_{target} \right).$$

where $\mathcal{D}_{target}$ represents the attacker's objective. By injecting such malicious updates, the attacker can substantially degrade the model accuracy, posing a severe threat to the reliability of distributed systems.

C. Paillier and Threshold Paillier

Paillier (Pai) [22] is a PHE scheme that supports additive homomorphism, thereby enabling direct operations on ciphertexts without requiring decryption. However, it does not support non-linear computations on ciphertexts. Pai comprises three main algorithms: Key Generation (Gen), Encryption (Enc), and Decryption (Dec). For any plaintext $m_1, m_2 \in Z_N$, the following properties hold, where $Pai.Enc(m)$ denotes that a message m is encrypted under Paillier.

$$Pai.Enc(m_1 + m_2) = Pai.Enc(m_1) \times Pai.Enc(m_2) \quad (1)$$

$$Pai.Enc(km_1) = Pai.Enc(m_1)^k \quad (2)$$

Zhao, Bowen, et al. [28] proposed a Paillier variant that preserves the same security and additive-homomorphic properties, while replacing expensive modular exponentiations with a single efficient multiplication. We adopt this fast Paillier scheme, along with its precomputation method, in our homomorphic-encryption and zero-knowledge-proof modules to reduce computational cost.

Since Paillier plaintexts lie in $\mathbb{Z}_N$, real-valued inputs are represented with fixed-point encoding using a public scaling factor $s \in \mathbb{N}$. For $x \in \mathbb{R}$, we encode x as $m = \lfloor sx \rceil$ and interpret it in $\mathbb{Z}_N$ (with signed representatives in the interval $(-N/2, N/2)$). Decoding maps m back to an approximate real value as $\tilde{x} = m/s$. In our experiments we set $s = 10^e$ and report the scaling exponent e in Supplemental Section V.

Threshold Paillier (ThPai) [23] is a variant of Paillier that incorporates verifiable threshold decryption. Given a threshold value t and a set of S secret keys, the decryption key is distributed among S participants, each equipped with a secret key GSK_k and a verification key $GVK_k, k \in [1, S]$. Any subset of at least t participants can collaboratively decrypt the ciphertext. Each participant can output a partial decryption along with a proof, and the plaintext can be reconstructed from verified partial decryptions. The key algorithms utilized are as follows. The security of ThPai is proved in [23].

KeyGen: Generates a public key GPK and secret keys $\{GSK_k\}_{k=1}^{S}$, each accompanied by a verification key GVK_k.

Enc: This algorithm selects a random value $r \in [1, N/4]$ to encrypt a message $M \in Z_N$ as $[M] = g^M r^{n^s} mod\ n^{s+1}$.

PDec: Given the ciphertext [M], a private key GSK_k and its verification key GVK_k, outputs the partial decryption c_k and a zero-knowledge proof v_k, convincing the verifier that the decryption was performed correctly using GSK_k.

Comb: After verifying the proofs and combining the S partial decryptions $\{c_k\}_{k=1}^{S}$, the plaintext M is recovered.

D. k-Out-of-n Oblivious Transfer

We incorporate the k-out-of-n Oblivious Transfer (OT) protocol into secure two-party computations to validate the outputs from DOes. The protocol involves two parties: a sender and a receiver. The sender holds n input strings $(x_0, x_1, \ldots, x_n) \in \{0,1\}^l$, while the receiver possesses k indices $\{r_1, r_2, \ldots, r_k\} \subset \{0, 1, \ldots, n-1\}$. Upon completion, the receiver learns $\{x_{r_1}, x_{r_2}, \ldots, x_{r_k}\}$ without gaining any knowledge about unselected inputs x_ψ where $\psi \in \{0, 1, \ldots, n-1\}$ and $\psi \notin \{r_1, r_2, \ldots, r_k\}$), while the sender remains unaware of the chosen indices. This ensures that the receiver's selection is hidden from the sender and prevents unauthorized access to unselected messages. We employ an OT protocol based on the Decisional Diffie-Hellman (DDH) assumption [29], which provides security against semi-honest receivers.

E. Proof Primitives

We introduce several proof primitives that verify basic mathematical operations of Paillier cryptosystem. Based on these, we develop verifiable and secure protocols for non-linear layers in training. $[x]$ denotes the encrypted value of x.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

Ciphertext Origin Proof Primitive (OriginProof) [23]: The prover commits a ciphertext $[a]$ and a plaintext a to the verifier, demonstrating that $[a]$ is indeed the encryption of a.

Ciphertext Equality Proof Primitive (EquProof) [14]: The prover commits $[a], [c]$ to the verifier, proving that $[a]$ and $[c]$ are encrypted from the same plaintext.

Ciphertext Comparison Proof Primitive (ComProof) [14]: The prover commits $[a], [c]$ and a comparison result to the verifier, convincing the verifier of the comparison result.

Ciphertext Addition/Subtraction Proof Primitive (AddProof/SubProof) [14]: The prover commits $[a], [b]$, and $[c]$ to the verifier, proving that $[c]$ is indeed computed from $[a \pm b]$. The verifier can compute $[a \pm b]$ on its own and then use EquProof to verify $[c] = [a \pm b]$.

Ciphertext Multiplication Proof Primitive (MulProof) [23]: The prover commits $[a], [b]$, and $[c]$ to the verifier, proving that $[c]$ is indeed computed from $[ab]$.

These primitives are honest-verifier ZKP algorithms of Paillier [14], [23]. Detailed descriptions of these proof primitives are provided in Section I of our Supplemental File.

III. Problem Statement

A. Threat Model

There are three types of entities in the system: a key generation center (KGC), a model owner (MO), and K data owners (DOes). The KGC is responsible for distributing keys to the MO and DOes. The MO maintains model parameters like weights and biases, performs training over encrypted data from DOes, and interacts with DOes for non-linear computations. Each DO holds private training data and assists in computing non-linear functions during training. Note that one of our goals is to ensure model privacy. In the MPC-based approach, none of the DOes obtains the model parameters. This is different from FL where model privacy is compromised due to the global model distribution mechanism [1].

The KGC is assumed to behave honestly during the one-time initialization in which it generates and distributes cryptographic keys (Section V), and it remains offline during all training rounds. We discuss the rationale and scope of this assumption in Section VIII. Consistent with prior works [1], [2], [7], [30], [31], we assume that the MO is honest-but-curious (i.e., semi-honest), meaning it strictly follows the protocol while attempting to infer sensitive data from DOes. We further discuss the assumption about a malicious MO in Section VIII. We consider two primary attack scenarios depending on whether the DOes engage in model poisoning.

Scenario #1 (Semi-honest DOes): The DOes are semi-honest, strictly following the protocol while attempting to infer private information from other parties [1], [2], [7], [30], [31]. We further distinguish between collusion and non-collusion settings. (i) In the *non-collusion* setting, the MO attempts to infer DOes' datasets by analyzing their model updates, while the DOes attempt to infer both the MO's model parameters and the data of other DOes [1]. (ii) In the *collusion* setting, up to $K-2$ semi-honest DOes may collude with the semi-honest MO to infer the data of the remaining DOes [32].[1] The MO will gain access to the datasets of the colluding DOes [14]. All colluding parties follow the protocol without deviation and combine only the information they legitimately obtain from their protocol transcripts.

Scenario #2 (Malicious DOes): Any DO may be malicious, aiming to degrade model accuracy by submitting forged results during interactions with the MO (e.g., model poisoning attacks) [24]. There are at most $K-2$ malicious DOes in the system.[2] We assume that DOes are aware of the aggregation rule and defense strategies used by MO. Therefore, we consider a full-knowledge setting to demonstrate robustness against strong attacks.

Out of scope attacks/complementary directions: Like all MPC-based systems, our framework cannot guarantee that a DO provides truthful data or prevent data poisoning attacks [33], where malicious DOes may inject corrupted data [34], [35], [36]. However, we ensure that once a DO submits its input to the computation, it is bound to using the same input consistently throughout the entire process. Specifically, this prevents a DO from providing inconsistent inputs at different stages of the computation or improperly mixing inputs from other parties. Since DOes have already submitted their encrypted data to MO, our primary focus is on verifying the correctness of computations based on the provided data. Additionally, such attacks can be mitigated through existing techniques, such as calculating the cosine similarity between local and MO model updates [6]. We will further discuss strategies for defending against data poisoning in Section VIII.

B. Design Goals

Our goal is to ensure that the private collaborative learning framework maintains model accuracy even under model poisoning attacks launched by malicious DOes. Specifically, the framework satisfies:

- *Model Privacy:* Ripple must safeguard model parameters from semi-honest DOes, ensuring that no DO can infer model parameters during training.
- *Data Privacy:* Ripple should ensure that the MO cannot infer the dataset of DOes. Additionally, we consider protecting data even under collusion attacks conducted by semi-honest MO and up to $K-2$ semi-honest DOes. Upon collusion, the MO learns the colluding DOes' datasets [14]; nevertheless, Ripple preserves the privacy of every non-colluding DO.
- *Resistance to Model Poisoning Attacks:* Ripple should maintain model accuracy against model poisoning attacks launched by up to $K-2$ malicious DOes under different data distributions.

[1] We exclude cases where $K-1$ or K semi-honest DOes collude with the MO, as this would lead to the public disclosure of all system information, thereby negating the need for privacy protection measures.

[2] While Ripple can verify all K DOes, a scenario where all DOes are malicious would render collaborative learning meaningless, as their updates would be collectively rejected and discarded. We ensure that there are at least two benign DOes in the system to guarantee the meaningfulness of collaborative learning.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

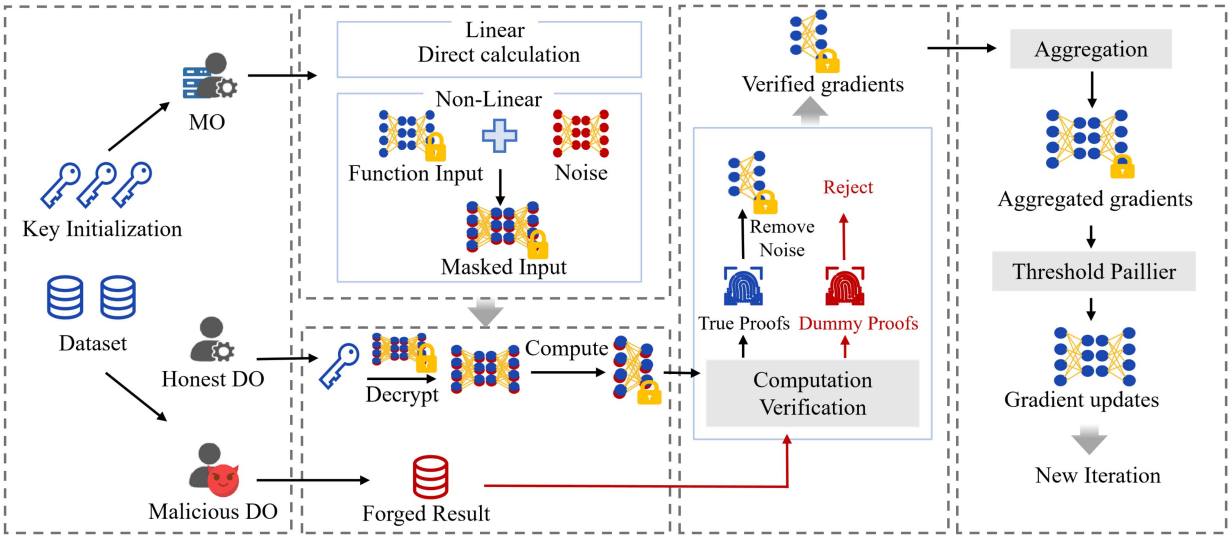


Fig. 1. The workflow of the MO and a DO in Ripple, including an initialization phase and three iterative training phases.

IV. Overview

Ripple addresses two fundamental challenges in private collaborative learning under malicious data owners. The first challenge is privacy leakage during interactive non-linear computations. In encrypted training, non-linear operations such as activation functions, comparisons, and max pooling cannot be executed directly over ciphertexts and therefore require interaction between the MO and a DO. However, such interaction may expose intermediate plaintexts and reveal sensitive model information. The second challenge is model poisoning attacks. Since a malicious DO may return forged intermediate results or partial decryptions, the MO must be able to validate the correctness of each computation before incorporating it into model training.

To address these two challenges, Ripple combines two complementary mechanisms. First, it introduces a removable-noise masking mechanism that preserves privacy during interactive non-linear computations. Before sending intermediate ciphertexts to a DO, the MO injects carefully crafted noise into the ciphertexts so that the DO only observes perturbed plaintexts after decryption. The DO then performs the required computation over the noisy plaintexts and returns the encrypted result to the MO. The MO can deterministically remove the injected noise and recover the correct ciphertext output. This design ensures that the DO cannot infer sensitive model parameters from the decrypted values.

Second, Ripple equips every non-linear computation and partial decryption with lightweight Paillier-based zero-knowledge proofs. The proof attests that the returned result is consistent with the corresponding noisy input and the prescribed computation rule, without revealing the DO's private inputs. By verifying the proof before aggregation, the MO can efficiently detect forged or malicious outputs; if verification fails, the MO rejects and discards the corresponding output, which is excluded from aggregation in that round. As a result, Ripple not only preserves privacy during interaction but also prevents malicious DOes from poisoning the training process with invalid computations.

Ripple organizes collaborative learning into an initialization phase and three iterative training phases (Fig. 1); the interaction between the MO and one DO is representative of all DOes.

Initialization: The KGC distributes personal Paillier keys and global threshold-Paillier (ThPai) keys to all participants and then goes offline. Each DO encrypts its local dataset under its personal key and uploads the ciphertext to the MO, so training proceeds on encrypted data without exposing DO plaintexts to the MO.

Forward propagation: The MO homomorphically evaluates linear layers—convolution, fully connected, and average pooling—on the encrypted inputs. Non-linear layers (ReLU, max pooling, Softmax) and ciphertext multiplications require MO–DO collaboration: the MO sends noise-blinded ciphertexts, the DO computes on perturbed plaintexts and returns encrypted results together with Paillier ZKPs, and the MO removes the mask and verifies each transcript before continuing.

Backward propagation: Encrypted gradients are computed in the same split manner: linear partial derivatives remain at the MO on ciphertexts, while multiplicative and non-linear gradient steps invoke the same verifiable sub-protocols (VSCP, VSMP, and layer-specific proofs in Section V).

Model update: The MO homomorphically aggregates only verified ciphertext gradients from all DOes. DOes then submit verifiable partial ThPai decryptions; the MO combines them to reconstruct the aggregate plaintext gradient, which only the MO learns, and updates the global model for the next iteration. If any verification step fails, the corresponding output is rejected and discarded in that round.

This workflow simultaneously pursues the design goals in Section III-B: ciphertext training and ThPai aggregation protect *data privacy*, removable noise during MO–DO interaction protects *model privacy*, and ZKP verification provides *poisoning resistance* against up to $K-2$ malicious DOes without relying on non-colluding third-party servers.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

V. Design Details

This section delves into the specifics of each phase.

A. Initialization Phase

This phase involves key initialization and data encryption.

Key Initialization: As assumed in Section III-A, the KGC generates (i) a personal Paillier key pair for each DO and (ii) global ThPai keys, distributes them to all participants, and then goes offline [2], [7]. The two key types are as follows:

(i) Personal key pairs $(\{PK_k\}_{k=1}^{K}, \{SK_k\}_{k=1}^{K})$ are used by each DO to encrypt their private data D_k. These key pairs are generated by the KGC according to the Paillier cryptosystem. The KGC distributes a unique key pair to each DO, while the MO retains only the public key of each key pair.

(ii) Global key pairs are jointly used by the MO and DOes to decrypt the aggregated gradients. Since each DO can only obtain a partial decryption and only the MO can obtain the final plaintext aggregated gradient, it safeguards DOes' data privacy because curious participants cannot infer DOes' data from a single gradient. Global key pairs are generated through ThPai, with a threshold set to $K+1$ to ensure that all participants must collaborate for decryption. The KGC generates a global public key GPK and $K+1$ sets of global secret keys $\{GSK_k\}_{k=1}^{K+1}$ and verification keys $\{GVK_k\}_{k=1}^{K+1}$, which are then distributed to each participant.

Encryption of the Dataset: Each DO uses its personal public key PK_k to encrypt its private training data, resulting in the encrypted dataset $[D_k]$. Subsequently, each DO uploads the encrypted dataset to the MO. Not only should we employ encryption algorithms to safeguard data privacy, but also support the verification of the computing results. Paillier's specific mathematical structure supports our ZKP schemes, which other cryptographic algorithms cannot achieve. Therefore, Pencil [1], which adopts the BFV fully homomorphic encryption scheme, cannot directly achieve verifiability by integrating our protocols.

B. Forward Propagation Phase

After receiving encrypted datasets from each DO, the MO performs forward propagation on the ciphertexts. This involves computing the neural network output and evaluating the loss between the outputs and the labels. We divide this process into linear layers and non-linear layers.

1) Linear Layers: Linear layers primarily include convolutional layers, fully connected layers, and average pooling layers. These layers connect neurons via weights $\mathbf{W}$ (a matrix) and biases $\mathbf{b}$ (a vector).

Convolutional and Fully Connected Layers: The core computation in these layers involves calculating $[\mathbf{WX}]$, which can be executed using Paillier's homomorphism property. For gradient computation in later phases, the forward propagation result of the i-th layer $\langle \mathbf{X}_i \rangle$ is split into two parts, $\langle \mathbf{X}_i \rangle_0$ and $\langle \mathbf{X}_i \rangle_1$, such that $\langle \mathbf{X}_i \rangle = \langle \mathbf{X}_i \rangle_0 + \langle \mathbf{X}_i \rangle_1$. The evaluation of these two layers is described in Algorithm 1.

Average Pooling Layers: The MO directly computes the average $\frac{1}{n}\sum_{i=1}^{n}[x_i]$ using Paillier's homomorphic properties.

Algorithm 1: Calculation of Convolutional and Fully Connected Layers.

Input: The MO holds $[\mathbf{X}_{i-1}]$, $\mathbf{W}$, $\mathbf{b}$.
Output: The output shares $[\mathbf{X}_i]$ of $[\mathbf{X}_i] = [\mathbf{W}\mathbf{X}_{i-1} + \mathbf{b}]$.
1: The MO calculates $[\mathbf{W}\mathbf{X}_{i-1}] = [\mathbf{X}_{i-1}]^{\mathbf{W}} = \left([\langle \mathbf{X}_{i-1} \rangle_0] + [\langle \mathbf{X}_{i-1} \rangle_1]\right)^{\mathbf{W}}$;
2: The MO generates a random mask $\mathbf{s} \in \mathbb{Z}_N$ and calculates $[\langle \mathbf{X}_i \rangle_1] = [\mathbf{W}\mathbf{X}_{i-1}] \times [\mathbf{s}]^{-1} = [\mathbf{W}\mathbf{X}_{i-1} - \mathbf{s}]$; The MO sends it to the DO;
3: The DO decrypts $[\langle \mathbf{X}_i \rangle_1]$ and obtains $\langle \mathbf{X}_i \rangle_1$. The MO holds $\langle \mathbf{X}_i \rangle_0 = \mathbf{s} + \mathbf{b}$.

Similarly to convolutional and fully connected layers, the result undergoes secret sharing for further processing.

2) Non-Linear Layers: During neural network training, non-linear layers are utilized to introduce non-linear capabilities, enabling the neural network to fit complex functional relationships. In this phase, the MO must obtain the encrypted output of non-linear layers. It poses challenges since these computations require DO assistance while ensuring that neither the MO nor the DO can infer plaintext input. Additionally, the MO must verify the correctness of results submitted by the DO. To address these, we integrate removable noise for model privacy and Paillier's ZKPs to validate DOes' computations.

We first design two *verifiable and secure sub-protocols* that serve as a foundational toolkit for non-linear layers. These protocols support all basic non-linear computations involved in training, including ciphertext comparison and multiplication for integers. We assume that both the MO and the DO participate, with the ciphertext owned by the MO being encrypted using Paillier. Protocols are designed to be secure and verifiable: neither party can determine the plaintext of the inputs, and the MO should be able to verify the DO calculations, thus mitigating the risk of the DO interfering with the protocol output by submitting forged values. To ensure verifiability, we design protocols that incorporate the proof primitives described in Section II-E.

Verifiable Secure Compare Protocol (VSCP): Given encrypted $[x]$ and $[y]$, the objective of VSCP is to determine the relational comparison between the plaintext values of these two encrypted data (i.e. $x > y$ or $x \leq y$). The protocol consists of an offline phase and an online phase. In the offline phase, the MO prepares $tuple_{MO} = \{r_3, r_4, [r_3 + r_4], [r_4]\}$. The online phase proceeds as follows:

Step-1: The MO calculates:

$$[d_0] = \left([x] \times [y]^{N-1}\right)^{r_3} \times [r_3 + r_4], \tag{3}$$

$$[d_1] = \left([y] \times [x]^{N-1}\right)^{r_3} \times [r_4], \tag{4}$$

The MO flips a coin s randomly. If $s = 0$, the MO sends $[d] = [d_0]$, If $s = 1$, the MO sends $[d] = [d_1]$.

Step-2: The DO decrypts it and obtains the plaintext d. If $d > N/2$ then sets $\mu_0 = 0$; otherwise sets $\mu_0 = 1$. Finally, the DO sends μ_0 back to the MO. It also generates proofs using ComProof to ensure the correctness of the comparison.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

Step-3: The MO verifies the correctness of the result. If $s = 0$ then MO sets $\mu = \mu_0$; otherwise sets $\mu = 1 - \mu_0$. If $\mu = 0, x \geqslant y$, otherwise, $x < y$.

Verifiable Secure Multiplication Protocol (VSMP): Given encrypted $[x]$ and $[y]$, the goal of VSMP protocol is to obtain $[xy]$. In the offline phase, the MO prepares $tuple_{MO} = \{s_1, s_2, [s_1], [s_2], [-s_1 s_2]\}$. The online phase proceeds as follows:

Step-1: The MO sends the blinded values to the DO:

$$[\widetilde{x}] = [x] \times [s_1] = [x + s_1], \tag{5}$$

$$[\widetilde{y}] = [y] \times [s_2] = [y + s_2], \tag{6}$$

Step-2: The DO decrypts $[\widetilde{x}]$ and $[\widetilde{y}]$, multiplies $\widetilde{x}$ and $\widetilde{y}$ to obtain $m = \widetilde{x}\widetilde{y} = (x + s_1)(y + s_2)$ and sends the encrypted $[m]$ to the MO. It also generates proofs using MulProof to ensure the correctness of the multiplication.

Step-3: The MO verifies the correctness of the result $[m]$. For MO with ciphertexts $[x]$, $[y]$ and plaintexts s_1, s_2 , it can calculate $[xs_2] = [x]^{s_2}$ and $[ys_1] = [y]^{s_1}$. These values are then used to remove noise:

$$[xy] = [m] \times [xs_2]^{N-1} \times [ys_1]^{N-1} \times [s_1 s_2]^{N-1}. \tag{7}$$

Upon establishing the aforementioned sub-protocols, we can subsequently calculate the outputs of non-linear layers, such as activation functions and the maximum pooling layer.

ReLU function: The ReLU function, defined as $ReLU(x) = \max\{0, x\}$, is extensively utilized in neural networks. As noted in [14], an implicit requirement is that the inputs and outputs of each layer must belong to $\{R^+, 0\}$. Initially, the inputs to the first layer are a feature matrix with elements normalized to the interval [0, 1]. Subsequently, the inputs to subsequent layers are derived from the non-negative outputs of the activation functions in the preceding layer.

Suppose $\mathbf{w}$ and $\mathbf{x}$ represent vectors, the computation of $z = \mathbf{wx}$ can be decomposed into two components: the product of the positive values of $\mathbf{w}$ with their corresponding elements of $\mathbf{x}$, denoted as z^+, and the product of the negative values of $\mathbf{w}$ with their corresponding elements of $\mathbf{x}$, denoted as z^-. The output of the ReLU function is determined only by comparing the magnitudes of z^+ and z^-.

In Ripple, the MO possesses **[x]** and $\mathbf{w}$. It computes $[z^+]$ and $[z^-]$ using the aforementioned method, then executes VSCP with DO to determine the magnitudes of $[z^+]$ and $[z^-]$. Specifically, if $\mu = 0$, $[relu(z)] = [z]$; otherwise, $[relu(z)] = [0]$. Ripple prevents attackers from inferring $\mathbf{x}$ from DO-to-MO computations, as the value of $\mathbf{x}$ cannot be deduced from the sign of $\mathbf{wx}$, as $\mathbf{wx}$ is indistinguishable from the inner product of $\mathbf{w}$ and a random vector to MO. The calculation of ReLU function is summarized in Algorithm 2.

Softmax function: The softmax function is frequently employed to compute the probabilistic output of a multiclass classification model. The formula is as follows:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \quad i = 1, 2, \ldots, n.$$

Algorithm 2: Calculation of Relu Function.

Input: The MO holds $\mathbf{w}$ and $[\mathbf{x}]$.
Output: The MO holds $[relu(\mathbf{wx})]$.
1: The MO applies function f^+ and f^- to $\mathbf{w}$:

$$f^+(w) = \begin{cases} 1, & w > 0 \\ 0, & w \leq 0 \end{cases} \quad f^-(w) = \begin{cases} 0, & w \geq 0 \\ -1, & w < 0 \end{cases}$$

2: The MO obtains $[z^+] = (f^+(\mathbf{w}) \odot \mathbf{w})[\mathbf{x}]$, $[z^-] = (f^-(\mathbf{w}) \odot \mathbf{w})[\mathbf{x}]$.
3: The MO and DO execute the VSCP on $[z^+]$ and $[z^-]$, if $\mu = 0$, $[relu(\mathbf{wx})] = [\mathbf{wx}]$; otherwise, $[relu(\mathbf{wx})] = [\mathbf{0}]$.

Algorithm 3: Verification of Exponential Function.

Input: The MO holds $[f(\mathbf{x})]$, $f(x) = e^x$, the DO holds $\mathbf{x}$.
Output: Verification results.
1: For each pair $(x_i, x_j), i, j \in [1, l], i \neq j$, the DO calculates $x_i + x_j$ and $[x_i + x_j]$ along with corresponding proofs using AddProof and OriginProof.
2: The MO accepts k (s.t. it is impossible to solve based on these k equations) of them through OT and verifies them. For every accepted $x_a + x_b$, MO calculates $[e^{(x_a + x_b)}]$.
3: The MO announces the pairs it accepted in step 2.
4: The MO and DO jointly execute MulProof, allowing the MO to verify that the product of $[e^{x_a}]$ and $[e^{x_b}]$ submitted by the DO does indeed equal to $[e^{(x_a + x_b)}]$.
5: If the verification is passed, the MO obtains the ordering of $[\mathbf{x}]$ by VSCP. It then checks the ordering of $[e^{\mathbf{x}}]$.

The formula for the softmax function is derived from solving the exponential function $f(x) = e^x$, so we begin by solving the exponential function. Consistent with prior research [1], we do not consider inference attacks based solely on prediction results, allowing the DO to obtain the plaintext prediction of final layer. The DO then computes the output of exponential function and transmits the ciphertext result to the MO.

To verify whether DO submits a forged $[f(\mathbf{x})]$, we propose a method that utilizes ZKP and OT to identify forged calculations. We validate $[e^{\mathbf{x}}]$ by cross-verifying different pairs within $[e^{\mathbf{x}}]$. For any pair $([x_a], [x_b]), a, b \in [1, l]$, the MO can verify $[e^{x_a}]$ and $[e^{x_b}]$ by combining the following two points. (i) The MO can determine the magnitudes of x_a and x_b using VSCP. Since $f(x) = e^x$ is a monotonically increasing function, if $x_a > x_b$, then $f(x_a) > f(x_b)$. This allows the MO to establish the theoretical relationship between $[e^{-x_a}]$ and $[e^{-x_b}]$. (ii) The MO can obtain $x_a + x_b$ using AddProof and OriginProof, then computes $[e^{(x_a + x_b)}]$. Subsequently, the MO verifies whether the product of $[e^{x_a}]$ and $[e^{x_b}]$ submitted by the DO is equal to $[e^{(x_a + x_b)}]$ using MulProof. The verification of exponential function is summarized in Algorithm 3. Refer to Section II of our Supplemental File for a detailed explanation.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

Algorithm 4: Calculation of Softmax Function.

Input: MO holds $[e^{x_i}]$, DO holds $e^{x_i}, i = 1, 2, \ldots, n$.
Output: MO receives $[10^e \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}]$.

1: The DO calculates the denominator $x = \sum_{j=1}^{n} e^{x_j}$ and the numerator $y_i = 10^e e^{x_i}$, which satisfies $y_i = q_i x + r_i, (r_i < x)$. Then q_i is the output of the Softmax. The DO sends $[q_i], [r_i]$ to the MO.
2: The MO performs homomorphic addition to obtain the encrypted denominator $[x] = [\sum_{j=1}^{n} e^{x_j}]$, performs homomorphic scalar multiplication to obtain the numerator $[y_i] = [10^e e^{x_i}]$.
3: The MO and DO execute VSCP to validate if $r_i < x$.
4: If passed, the MO calculates $[y_i - r_i] = [y_i] \times [r_i]^{-1}$. They execute VSMP to validate if $y_i - r_i = q_i x$.

Algorithm 5: Calculation of $\bigtriangledown_{W_i}$

Input: The MO holds $\langle \bigtriangledown_{X_{i+1}} \rangle_0, \langle X_i \rangle_0, [\langle \bigtriangledown_{X_{i+1}} \rangle_1], [\langle X_i \rangle_1]$ and the DO holds $\langle \bigtriangledown_{X_{i+1}} \rangle_1, \langle X_i \rangle_1$.
Output: $[\bigtriangledown_{W_i}]$.

1: The MO calculates $[m] = [\langle \bigtriangledown_{X_{i+1}} \rangle_0 \langle X_i \rangle_0] \times [\langle X_i \rangle_1]^{\langle \bigtriangledown_{X_{i+1}} \rangle_0} \times [\langle \bigtriangledown_{X_{i+1}} \rangle_1]^{\langle X_i \rangle_0}$;
2: The MO selects a random number $s \in Z_N$ and sends $[\widetilde{m}] = [m] \times [s] = [m + s]$ to the DO.
3: The DO computes $\langle \bigtriangledown_{X_{i+1}} \rangle_1 \langle X_i \rangle_1$ and sends $[n] = [\langle \bigtriangledown_{X_{i+1}} \rangle_1 \langle X_i \rangle_1]$ encrypted by GPK. It also generates proofs by **MulProof**; In addition, the DO decrypts $[\widetilde{m}]$ and returns $[\widetilde{m}]'$ encrypted by GPK along with corresponding proofs using **EquProof**.
4: The MO verifies the proofs. If accepted, it computes $[\bigtriangledown_{W_i}] = [\widetilde{m} - s]' \times [n]$.

The Softmax function incorporates a division operation, which is challenging to verify directly. Therefore, we transform the verification of this division into a multiplication. We employ Algorithm 4 to compute the output of the Softmax.

Similarly, the output of the Sigmoid function is computed using an analogous approach. When considering the Tanh function of the last layer, it can be expressed as $2 \times Sigmoid(2x) - 1$, thereby enabling its computation based on the procedure for the Sigmoid function.

Max pooling layers: In contrast to average pooling layers, max pooling layers select the maximum value within the pooling window as the output. The core operation of this involves comparing ciphertexts, a process facilitated by VSCP.

Utilizing these linear and non-linear functions, the MO can obtain the encrypted layer outputs and the encrypted loss $[L]$ with respect to the output of the last layer.

C. Backward Propagation Phase

Upon obtaining the encrypted loss $[L]$, the MO proceeds to compute $\bigtriangledown_X, \bigtriangledown_W$ and $\bigtriangledown_b$. The MO can directly compute $\bigtriangledown_X$ and $\bigtriangledown_b$ because the essence of these computations is *ciphertext-plaintext multiplication*. However, computing $\bigtriangledown_W$ requires ciphertext-ciphertext multiplication, which poses a challenge for the MO to compute directly. To address this issue, we utilize the secret shares generated by the linear layers, thereby eliminating the need for the MO to perform ciphertext-ciphertext multiplication.

Calculation of $\bigtriangledown_X$ and $\bigtriangledown_b$: For $\bigtriangledown_{X_i} = \bigtriangledown_{X_{i+1}} \frac{\partial f(X_i)}{\partial X_i} = \bigtriangledown_{X_{i+1}} W_i$, where $\bigtriangledown_{X_{i+1}}$ is the gradient propagated from the subsequent layer, the MO can compute independently. Similar to the linear layer's forward pass, the result is shared between the MO and DO. For $\bigtriangledown_{b_i} = \bigtriangledown_{X_{i+1}} \frac{\partial f(X_i)}{\partial b_i}$, the MO can compute it directly as $\bigtriangledown_{X_{i+1}}$ is known and $\frac{\partial f(X_i)}{\partial b_i}$ is constant.

Calculation of $\bigtriangledown_W$: The computation of $\bigtriangledown_{W_i} = \bigtriangledown_{X_{i+1}} \frac{\partial f(X_i)}{\partial W_i} = \bigtriangledown_{X_{i+1}} X_i$ poses a challenge to the MO due to the inability to directly perform the multiplication of two ciphertexts. To address this, we leverage the previously established secret shares. Specifically, we expand the multiplication using secret shares of $\bigtriangledown_{X_{i+1}}$ and X_i.

$$\begin{aligned}
[\bigtriangledown_{W_i}] &= [\bigtriangledown_{X_{i+1}} X_i] \\
&= [(\langle \bigtriangledown_{X_{i+1}} \rangle_0 + \langle \bigtriangledown_{X_{i+1}} \rangle_1)(\langle X_i \rangle_0 + \langle X_i \rangle_1)] \\
&= [\langle \bigtriangledown_{X_{i+1}} \rangle_0 \langle X_i \rangle_0] \times [\langle \bigtriangledown_{X_{i+1}} \rangle_0 \langle X_i \rangle_1] \times \\
&\quad [\langle \bigtriangledown_{X_{i+1}} \rangle_1 \langle X_i \rangle_0] \times [\langle \bigtriangledown_{X_{i+1}} \rangle_1 \langle X_i \rangle_1]. \qquad (8)
\end{aligned}$$

This decomposes into four terms, where the MO computes three using its shares. The fourth term, involving the product of the DO's shares $\langle \bigtriangledown_{X_{i+1}} \rangle_1$ and $\langle X_i \rangle_1$ is computed by the DO. The DO sends the result encrypted using GPK along with proofs generated via MulProof and EquProof. The MO verifies these proofs before combining all terms to obtain $[\bigtriangledown_{W_i}]$. As $\langle \bigtriangledown_{X_{i+1}} \rangle_1$ and $\langle X_i \rangle_1$ are obfuscated, the DO cannot extract any sensitive information. Refer to Algorithm 5 for details.

D. Model Update Phase

The MO aggregates verified gradients from all DOes. The MO then initiates a request for the DOes to decrypt the aggregated gradients using the threshold decryption mechanism of ThPai. Each DO computes a partial decryption c_k on the aggregated gradients, along with a decryption proof v_k to convince the MO that it has performed a correct partial decryption.

$$(c_k, v_k) = ThPai.PDec(GSK_k, GVK_k, [\bigtriangledown_W]), \qquad (9)$$

The MO validates the partial decryptions and proofs from all DOes. Upon successful verification, the MO combines partial decryptions to prevent DOes from inferring the plaintext gradient. The aggregated gradients are reconstructed as:

$$\bigtriangledown_W = ThPai.Comb\left((c_k, v_k)_{k=1}^{K+1}\right), \qquad (10)$$

The MO then updates the global model:

$$W_{t+1} = W_t - \eta \bigtriangledown_W . \qquad (11)$$

After updating the global model, all participants proceed to the next iteration. This iterative process continues until the predefined maximum number of iterations is reached.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

VI. Security Analysis

A. Proof Intuition and Simulator Overview

Before stating Definition 1, we summarize the simulator intuition that underlies our privacy guarantees; complete hybrid arguments are deferred to Supplemental Sections III and IV.

Data-privacy simulator Sim_{MO}: The MO's view comprises Paillier ciphertexts, proof messages from MulProof, ComProof, and OriginProof (Section II-E), and threshold-decryption messages. Sim_{MO} outputs random ciphertexts and simulates these proof messages. Because the primitives in Section II-E are honest-verifier zero-knowledge, the simulated transcripts are computationally indistinguishable from real ones even without DO plaintexts.

Model-privacy simulator Sim_{DO}: Each DO decrypts only noise-blinded ciphertexts $[x+s]$ and receives secret shares that are indistinguishable from uniform values in $\mathbb{Z}_N$. Sim_{DO} samples random masks and simulates the corresponding proof messages accordingly.

Collusion-resistance simulator Sim_{MD}: Under passive semi-honest collusion (Section III-A), the MO knows the colluding DOes' datasets, matching the input $\{D_k\}_{k=3}^{K}$ to Sim_{MD} below. Colluders follow the protocol without deviation and combine only their permitted protocol transcripts; Sim_{MD} simulates their joint transcript while replacing non-colluders' sensitive values with random coins. The full hybrid proof is given in Section IV of the Supplemental File.

Definition 1: A protocol Π between an MO with model M as input and K DOes, each with a private dataset D_k, is a private training protocol that defends against model poisoning attacks if it satisfies the following guarantees.

- *Robustness against model poisoning:* The output at the end of the system is the global model indeed trained on K datasets of DOes, i.e., $\Pi(\{D_k\}_{k=1}^{K}, \mathcal{M}) = (\emptyset, \Omega(\mathcal{M}))$, where $\Omega(\mathcal{M})$ denotes the global model trained on $\{D_k\}_{k=1}^{K}$.
- *Data privacy:* A semi-honest MO cannot infer the DOes' training data. Formally, we require the existence of an efficient simulator Sim_{MO} such that $View_{MO}^{\Pi} \approx_c Sim_{MO}(M, out)$, where $View_{MO}^{\Pi}$ denotes the view of the MO in the execution of Π, out denotes the output of the training protocol to the MO, and $\approx_c$ denotes computational indistinguishability between two distributions.
- *Model privacy:* A semi-honest DO cannot infer the MO's model parameters, i.e., W and b. Formally, we require the existence of an efficient simulator Sim_{DO} such that $View_{DO}^{\Pi} \approx_c Sim_{DO}(D)$, where $View_{DO}^{\Pi}$ denotes the view of the DO in the execution of Π.

  Moreover, even if the semi-honest MO colludes with up to $K-2$semi-honest DOes, they cannot learn private training data of other DOes. Formally, there is an efficient simulator Sim_{MD} such that $View_{MD}^{\Pi} \approx_c Sim_{MD}(\{D_k\}_{k=3}^{K}, M, out)$, where $View_{MD}^{\Pi}$ denotes the joint view of the MO and the DOes in the execution of Π.

Theorem 1: In the presence of a semi-honest MO and a malicious DO, assuming the existence of Paillier, VSCP, VSMP, Calculation of Relu, Softmax and $\bigtriangledown_{W_i}$, our framework meets the three conditions of ZKPs: completeness, soundness and zero-knowledge under standard cryptographic assumptions.

Proof sketch: Completeness follows directly from the protocol design. Since VSCP, VSMP, Calculation of Relu, Softmax and $\bigtriangledown_{W_i}$ satisfy the special soundness property, a simulator generating fake proofs succeeds with at most negligible probability. Because Pai is a perfectly binding commitment scheme, the MO's view consists only of proofs; the proofs of VSCP, VSMP, Calculation of Relu, Calculation of Softmax, and Calculation of ∇_{W_i} are computationally indistinguishable from those in an ideal execution, so robustness against model poisoning follows. The zero-knowledge property ensures that proof messages reveal no information about DO private inputs beyond the validity of the stated computation; this is captured by Sim_{MO} in Section VI-A.

Theorem 2: Assuming the existence of Paillier, Threshold Paillier, and secure protocols for non-linearity evaluations, our design is a private protocol as defined in Definition 1.

Theorem 2 is proved under the passive semi-honest collusion model in Section III-A. Full simulator constructions and hybrid proofs appear in Section IV of the Supplemental File.

VII. Performance Evaluation

Our evaluations are designed to validate the following.

- Ripple significantly defends against model poisoning attacks under different data distributions and up to $K-2$ malicious DOes (Section VII-C).
- Ripple effectively protects data privacy against gradient matching attacks, even in scenarios where the semi-honest MO colludes with up to $K-2$ DOes (Section VII-D).
- Ripple achieves competitive efficiency, incurring lower time overhead than the SOTA MPC-based framework. (Section VII-E).

A. Implementation

Our experiments are conducted using PyTorch on a workstation equipped with a 13th Gen Intel Core i7-13700 K processor running at 3.40 GHz and 32 GB of RAM. For cryptographic operations, we employ the batch encryption-accelerated version of the fast Paillier scheme, which accelerates the training convergence by up to 81× [37] .[3] We evaluate two typical network settings: a LAN with 384 MB/s bandwidth and a WAN with 44 MB/s bandwidth, consistent with [1]. The modulus N is set to 2048 bits, providing a 112-bit security level. All experiments are conducted using Python 3.9.

Since Paillier operates on integers, similar to prior works [14], [38], we adapt floating-point numbers for encryption by multiplying them by 10^e, rounding them to fixed-point integers. The original floating-point values can be approximately recovered by dividing the decrypted results by the same constant. This conversion has a negligible impact on model accuracy as demonstrated in [14]. We apply the same fixed-point encoding to all encrypted quantities—features, model parameters, and intermediate values in both linear and non-linear layers—before encryption and verification. Table I in Supplemental Section

[3] https://github.com/marcoszh/BatchCrypt

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

V reports the scaling exponent e for each dataset; all parties and proof checks use identical fixed-point integers end-to-end. Supplemental Section IX provides a sensitivity analysis on CIFAR-10.

B. Evaluation Setup

1) Datasets: We evaluate on three widely used datasets from different domains, including image classification datasets (MNIST, CIFAR-10) and a human activity recognition dataset (HAR).

- *MNIST*[4] contains 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9, each with 28 $\times$ 28 pixels.
- *CIFAR10 [39]* contains 50,000 training images and 10,000 testing images of various objects belonging to ten categories, each with 32 $\times$ 32 pixels.
- *HAR [40]* consists of 7,732 training and 2,567 testing samples of human activities in six classes, with 561 features per sample.

2) Data settings: We construct both independent and identically distributed (IID) and non-IID settings. We adopt the data partitioning method from [14]. Given a dataset with M classes, DOes are divided into M groups. A sample with label l is assigned to group l with probability q, and to other groups with probability $\frac{1-q}{M-1}$. Within each group, data is uniformly distributed among DOes. q controls the degree of non-IID.

- *IID setting:* This setting simulates a collaborative scenario with uniformly distributed data. When $q = \frac{1}{M}$, the setting is IID. By default, we adopt the IID setting.
- *Non-IID setting:* This setting reflects the real-world heterogeneity in data distributions, where each DO holds data biased towards a subset of classes. We set $q = 1$ so that each DO stores data belonging to a single class.

3) Parameter Settings and model architectures: We consider a setting with 30 DOes participating in each training task. The learning rate is set to 0.01, with 500 global training rounds. Each DO performs one local epoch per round using a batch size of 64. We select representative model architectures tailored to the characteristics of each dataset. For HAR, we adopt a Multi-Layer Perceptron (MLP) with hidden layers [256, 64, 6]. For MNIST, we employ a Convolutional Neural Network (CNN) to capture spatial features in handwritten digit classification. For CIFAR-10, we follow the CNN architecture in Pencil [1]. The detailed system parameters and architecture configurations are listed in Section V of Supplemental File.

4) Attacks: We evaluate the performance of Ripple and existing methods under the following attack scenarios:

- *Model poisoning attack* aims to undermine model accuracy. Malicious DOes can send arbitrary values instead of locally updated values to poison the model. To make a comprehensive comparison, we assess four advanced attacks that can significantly reduce the accuracy of the model: *Trim Attack* [24] crafts malicious gradients based on gradient extremes. *Krum Attack* [24] adjusts malicious gradient magnitudes to come closer to the benign update. *FLTrust Attack* [6] maximizes the similarity between malicious updates and MO's seed updates. *Min-Max Attack* [25] optimizes the value range of malicious updates to fall within the benign gradient range. These attacks are presented in FL frameworks, and in MPC frameworks, we apply them to the computations of the DOes instead of the gradient updates.
- *Gradient Matching Attack*[5] [3], [41] reconstructs data by optimizing dummy inputs and labels such that their computed gradients closely match the gradients. We consider a collusion scenario in which the semi-honest MO and $K-2$ semi-honest DOes collaborate to infer labels of the remaining two DOes, based on their aggregated gradients. A system that can resist such a strong collusion attack inherently defends against attacks from a single semi-honest MO.

We do not evaluate DOes as independent inference attackers targeting model privacy, as Ripple does not expose the global model to any DO. Furthermore, during interactions with the MO, each DO only receives carefully perturbed inputs, which are computationally indistinguishable from random values.

5) Metrics: We evaluate the ability to resist model poisoning attacks using global model **accuracy (Acc)**, as these attacks aim to decrease testing accuracy. A method is considered more effective against attacks if its global model predicts labels for a higher fraction of testing samples correctly under those attacks.

$$Acc = \frac{\#\ correctly\ classified\ samples}{\#\ total\ samples}$$

For the gradient matching attack, we employ **Attack Acc**—the proportion of correctly extracted labels to evaluate the effectiveness of data privacy protection.

C. Evaluation Under Model Poisoning Attacks

Resistance to model poisoning: In this subsection, we evaluate Ripple against model poisoning attacks. Specifically, we assess its performance under $K/2-1$ malicious DOes using the MNIST dataset (Table II). The evaluation is conducted under both IID and non-IID settings described in VII-B. To simulate adversarial scenarios, we configure the malicious DOes to launch the four poisoning attacks described above by manipulating their submitted updates in each training round. For MPC-based methods, Ripple is compared with the state-of-the-art private framework Pencil [1], the closest single-server MPC baseline that, like Ripple, does not rely on non-colluding third-party servers, which also eliminates the need for the non-collusion assumption. With the growing concern over model poisoning, several FL methods also introduce techniques to detect malicious gradients. To further assess Ripple's effectiveness, we also compare it with the representative FL frameworks. FedAvg [26], FLTrust [6], and ShieldFL [5] represent plaintext FL baselines with varying defense strength; they benchmark robustness but do not provide the same data and model privacy guarantees as Ripple. FLTrust [6] uses a trusted root dataset to identify malicious updates in plaintext. ShieldFL [5] performs

[4]https://yann.lecun.com/exdb/mnist/

[5]https://github.com/PatrickZH/Improved-Deep-Leakage-from-Gradients

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

TABLE II
EVALUATION UNDER MODEL POISONING ATTACKS ON THE MNIST DATASET ($m = 14$)

Data Distribution	Work	Model Acc Without Attacks	Model Acc under Model Poisoning Attacks			
			Acc Under Trim	Acc Under Krum	Acc Under FLTrust	Acc Under Min-Max
IID	FedAvg [27]	0.9064	0.7827↓12%	0.1861↓72%	0.7613↓14%	0.1351↓77%
	FLTrust [6]	0.8685↓4%	0.8662↓4%	0.8659↓4%	0.8664↓4%	0.8664↓4%
	ShieldFL [5]	0.9060 →	0.8724↓3%	0.7661↓14%	0.8693↓4%	0.8477↓6%
	Pencil [1]	0.8951↓1%	0.8023↓10%	0.3794↓53%	0.8489↓6%	0.3728↓53%
	Ripple	0.9059 →	0.8935↓1%	0.8726↓3%	0.9017 →	0.9033 →
non-IID	FedAvg [27]	0.7506	0.2973↓45%	0.2506↓50%	0.2017↓55%	0.3142↓44%
	FLTrust [6]	0.7104↓4%	0.1635↓59%	0.3353↓42%	0.1635↓59%	0.1633↓59%
	ShieldFL [5]	0.7463 →	0.6973↓5%	0.5578↓19%	0.6528↓10%	0.5176↓23%
	Pencil [1]	0.7294↓2%	0.3789↓37%	0.2563↓49%	0.4199↓33%	0.4354↓32%
	Ripple	0.7497 →	0.7419 →	0.7482 →	0.7163↓3%	0.7301↓2%

We use ↓ to highlight accuracy losses relative to FedAvg without attacks, and → to indicate that accuracy remains within 1% of the no-attack baseline.

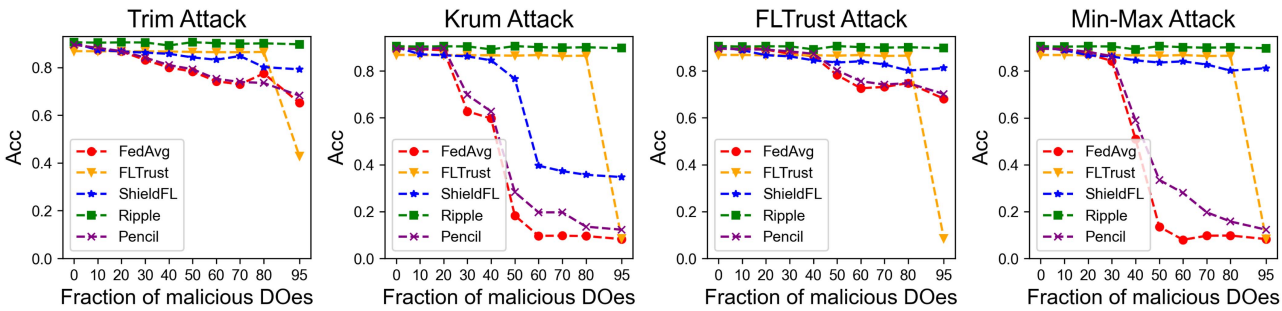


Fig. 2. The impact of varying the fraction of malicious DOes under the IID setting.

ciphertext-level cosine similarity to detect malicious DOes. FedAvg [26] is a standard FL algorithm without any defense that serves as the baseline. Results on additional datasets are provided in Section VI of our Supplemental File.

In the absence of attacks, Ripple consistently achieves high model accuracy as its verifiable secure computation framework does not compromise performance. In contrast, Pencil preserves data privacy by injecting noise into gradients, which inevitably compromises accuracy even in non-adversarial settings. FLTrust experiences an accuracy drop of 4% under both IID and non-IID conditions, while Ripple's accuracy degradation stays within 1%. Unlike heuristic defenses that may discard benign gradients, Ripple retains all verified updates and rejects only outputs that fail ZKP verification, which allows it to preserve valuable information and maintain high accuracy.

When subjected to attacks, Ripple maintains accuracy within 4% of baseline. However, both Pencil and FedAvg exhibit substantial drops in accuracy. This degradation is particularly pronounced under the Krum and Min-Max attacks in the IID setting, where malicious DOes can craft poisoned updates that remain within a plausible range, thereby evading detection. In IID settings, FLTrust stabilizes at 86% accuracy due to its reliance on a root training set for gradient similarity assessment. However, in non-IID scenarios, benign gradients diverge from those derived from the IID root set, causing FLTrust's accuracy to drop by 59%. ShieldFL partially mitigates this issue but remains less effective than Ripple. For example, under a Min-Max attack in the non-IID setting, ShieldFL's accuracy drops by 23%. Ripple overcomes these challenges through cryptographic verification that detects and rejects forged computations before aggregation, ensuring nearly all malicious outputs are excluded. This results in significantly higher model accuracy compared to existing defenses.

Impact of the fraction of malicious DOes: To further assess the resilience of Ripple, we simulate attacks of varying severity by increasing the fraction of malicious DOes from 0% to 95% (i.e., 28 malicious DOes) on the MNIST dataset.

Figs. 2 and 3 demonstrate that Ripple maintains accuracy comparable to FedAvg even with 95% malicious DOes in IID settings. FLTrust maintains a relatively high accuracy with up to 90% of malicious DOes. However, it struggles under non-IID, a common limitation of gradient similarity-based defenses. This is because in non-IID settings, benign DOes produce diverse and inconsistent updates due to heterogeneous data. While ShieldFL mitigates this issue by filtering out unnormalized gradients, this normalization process forces all gradients to converge within a limited range of values, reducing the influence of extreme values or outliers on the global model update. As a result, it limits the effectiveness of partial attacks, such as Trim, FLTrust, and Min-Max. Conversely, Krum aims to reduce the distance between the gradients of malicious and benign DOes, ensuring that malicious updates are not excluded, thus enhancing the effectiveness of the attack. In contrast, Ripple maintains stability and achieves accuracy comparable to FedAvg in the absence of attacks. When the data distribution is non-IID, model accuracy remains stable unless the number of distinct classes held by benign DOes is fewer than the number of classes in the test set, as the model lacks exposure to certain categories during training and thus fails to generalize effectively.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

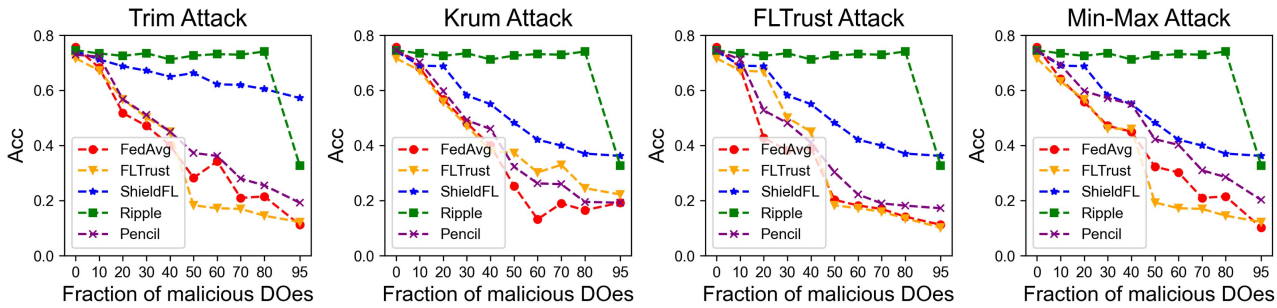


Fig. 3. The impact of varying the fraction of malicious DOes under the non-IID setting.

TABLE III
EVALUATION UNDER GRADIENT MATCHING ATTACK

Task	Model	Work	noise level	Attack Acc	Model Acc
HAR	MLP	FedAvg	—	1.0000	0.9322
		FLTrust	—	1.0000	0.8956
		ShieldFL	—	0.1696	0.9179
		Pencil	0	1.0000	0.9124
			0.005	0.3824	0.9097
			0.01	0.1722	0.9083
		Ripple	—	**0.1693**	**0.9193**
MNIST	CNN	FedAvg	—	1.0000	0.9064
		FLTrust	—	1.0000	0.8685
		ShieldFL	—	0.1481	0.9060
		Pencil	0	1.0000	0.9055
			0.005	0.3607	0.8968
			0.01	0.1482	0.8951
		Ripple	—	**0.1477**	**0.9059**
CIFAR10	CNN	FedAvg	—	1.0000	0.9133
		FLTrust	—	1.0000	0.8719
		ShieldFL	—	0.1326	0.9079
		Pencil	0	1.0000	0.9031
			0.005	0.3920	0.8987
			0.01	0.1425	0.8974
		Ripple	—	**0.1278**	**0.9042**

TABLE IV
TRAINING COSTS FOR DIFFERENT ML TASKS

Task	Model	Work	Communication	Time (hours)
HAR	MLP	Pencil	0.27	2.1
		Ripple	1.78	2.2
		Ripple$^+$	1.96	0.8
MNIST	CNN	Pencil	0.35	4.4
		Ripple	4.29	4.5
		Ripple$^+$	5.91	1.9
CIFAR10	CNN	Pencil	35.5	231
		Ripple	50.17	242
		Ripple$^+$	51.43	49.8

Communication stands for the communication (MB) per sample. Ripple$^+$ denotes the variant of Ripple with accelerated optimizations applied.

D. Evaluation on Protecting Data Privacy

We evaluate Ripple's effectiveness in protecting data privacy against the gradient matching attack and compare it with Pencil, FedAvg, FLTrust, and ShieldFL on the three datasets. We set the batch size to one since this attack requires a very small batch size. We report Attack Acc (attack success rate) and Model Acc (model accuracy under attack) in Table III.

We can observe that Ripple achieves attack accuracy close to random guessing, indicating that the aggregated gradient-based approach fails to effectively perform label inference. In contrast, FedAvg and FLTrust are vulnerable to inference attacks because the MO can access individual plaintext gradients, enabling more precise reconstruction. ShieldFL only exposes aggregated gradients to the MO, making the its attack success rate comparable to that of Ripple. However, it compromises model privacy. Pencil allows the MO to access the plaintext gradients of each DO, enabling MO to directly infer the labels of the DO's training data. To mitigate this risk, Pencil introduces noise to individual gradients, aiming to obscure the sensitive information. However, this comes at the cost of model accuracy. Overall, Ripple consistently outperforms Pencil under all three noise levels. Additionally, we visualize the original data inferred by attacker from gradients in Section VII of our Supplemental File, which clearly demonstrates that Ripple provides strong protection of data privacy.

E. Evaluation on Time and Communication Overhead

We evaluate time and communication overhead of Ripple on the HAR, MNIST, and CIFAR-10 datasets. To ensure fairness, we compare only with MPC-based methods that preserve both data and model privacy. As the most advanced method that does not rely on the non-collusion assumption, Pencil serves as our primary baseline. We also leverage a reduction strategy and preprocessing techniques to enhance the efficiency of Ripple. The optimized approach is referred to as Ripple$^+$. A comprehensive overview of the detailed optimization methods can be found in Section VIII of our Supplemental File. Table IV presents the overhead during training of the MO between a DO, averaged over ten epochs.

In the LAN network setting, Ripple$^+$ can train models for HAR and MNIST from scratch within 0.4 hours and 1.9 hours, respectively. The cost of training CNN on CIFAR10 is higher than that for other tasks, as CIFAR10 consists of 3-channel color images with more features. While Pencil does not defend against model poisoning and thus avoids the associated overhead, its use of FHE, which is more complex than the optimized PHE adopted by Ripple, leads to greater time consumption. We also show a time overhead analysis of Ripple from an overall perspective in Section IX of our Supplemental File.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

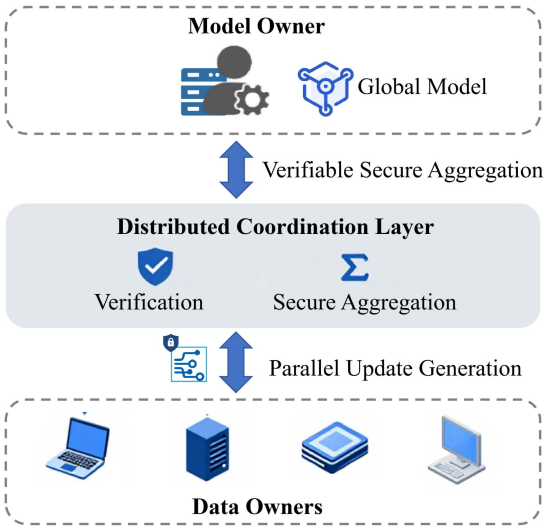


Fig. 4. Distributed verifiable collaborative learning architecture of Ripple.

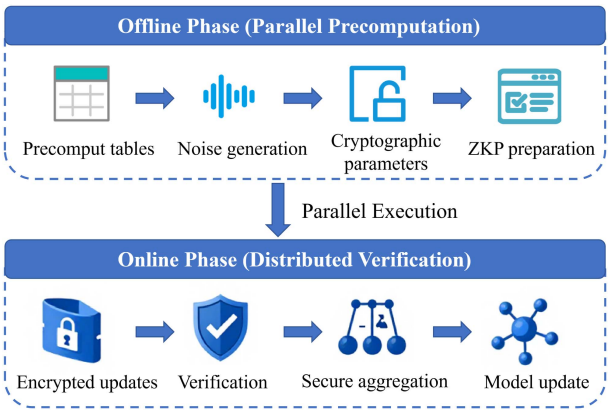


Fig. 5. Parallel execution pipeline of Ripple with offline–online phase decoupling.

F. Distributed and Parallel Verifiable Secure Engineering Applications

Ripple supports scalable distributed MO/DO collaboration with verifiable aggregation and offline–online parallel execution (Figs. 4–5).

1) *Distributed Verifiable Collaborative Learning Architecture:* Ripple can be deployed as a distributed learning infrastructure across geographically distributed data owners and model owners, such as edge–cloud collaborative systems, federated data centers, and distributed AI platforms. As illustrated in Fig. 4, Ripple introduces a distributed coordination layer that manages secure interactions between model owners and multiple data owners. Each data owner performs local training and generates encrypted model updates, while the model owner coordinates distributed verification and aggregation through verifiable secure computation protocols.

The distributed architecture supports parallel update generation and verification across participants; removable noise and ZKPs validate each computation without exposing sensitive data or model parameters, enabling secure training in edge–cloud and cross-organizational settings.

2) *Parallel Verifiable Secure Computation Pipeline:* To further improve efficiency, Ripple supports parallel execution of secure computation tasks through offline–online phase decoupling and precomputation mechanisms. As shown in Fig. 5, the framework separates computation into parallelizable offline precomputation and latency-sensitive online verification phases. During the offline phase, distributed nodes perform parallel generation of cryptographic parameters and precomputation tables, which can be executed asynchronously across computing resources. The online phase then performs lightweight verification and secure aggregation with reduced communication overhead and latency. This pipeline enables high-throughput distributed training and significantly improves system scalability for large models and high-dimensional data.

Parallel verification across heterogeneous nodes further reduces bottlenecks in large-scale deployments.

3) *Engineering Implications for Distributed and Parallel Verifiable Security:* Ripple provides a general engineering framework for distributed and parallel verifiable security in collaborative machine learning systems. Its design enables: a) scalable distributed verification of model updates; b) parallel execution of secure computation protocols; c) verifiable integrity guarantees in decentralized environments, and d) privacy-preserving distributed system coordination.

These properties support scalable, verifiable coordination in adversarial decentralized environments.

VIII. Discussion

In this section, we discuss the assumptions on the KGC and the MO, security against data poisoning attacks, and overhead of Ripple.

Assumption on the KGC: We model the KGC as honest only at setup for two reasons. First, this matches standard PHE-based private collaborative learning [1], [2], [7], where a trusted initializer bootstraps keys before MO–DO training begins. Second, Ripple targets extensible MO/DO collaboration without online non-colluding third-party servers during training; limiting KGC involvement to initialization avoids a continuously trusted party in every round. Our formal analysis therefore focuses on privacy and verifiability during training-phase interaction among the MO and DOes. Setup-time KGC compromise—which could expose DO secret keys or issue malformed keys—lies outside this training-round threat model, consistent with prior works that treat key setup as a separate trust root [1], [2], [7]. Strengthening setup trust via distributed key generation or certified locally generated keys is a direction for future work.

Assumption on the MO: Specifically, a malicious MO may attempt incorrect aggregation [42] to degrade model accuracy. However, in Ripple, the MO, as the model owner, has a primary objective of obtaining a high-quality model and thus has no incentive to perform such attacks. Moreover, during the entire interaction between the MO and DOes, the MO only receives encrypted data. The MO cannot extract any meaningful information from ciphertexts under Paillier.

Defending against data poisoning attacks: While Ripple primarily defends against model poisoning attacks, it can also be integrated with existing defenses to detect data poisoning.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

For instance, if the MO possesses a clean root dataset, it can leverage the homomorphic properties of Paillier to compute cosine similarity between local updates and the MO's own update [6]. For detailed experiments, please refer to Section X of our Supplemental File.

Overhead: Ripple targets high accuracy while ensuring strong security guarantees. To verify malicious DOes, Ripple leverages HE and ZKPs, which inevitably introduce overhead compared to non-cryptographic baselines. Nonetheless, Ripple benefits from the relatively low cost of the optimized PHE compared to FHE. To further reduce overhead, future work will focus on optimizing cryptographic primitives and protocol-level efficiency, without compromising Ripple's security guarantees.

IX. Related Work

Private Collaborative Learning: Private collaborative learning enables both data privacy and model privacy. Representative works are based on MPC, summarized in Table I.

The common setting in MPC-based approaches requires DOes to share encrypted data with (at least two) non-colluding third-party computing MOs [7], [8], [8], [9], [10], [11], [12], [18], [19], [20]. However, these require a critical non-colluding assumption. Pencil proposes a private collaborative learning framework that satisfies *extensibility:* because the MO often seeks to train the model with multiple DOes, it is critical to ensure that the MO can incrementally collaborate with different DOes. The previous methods exhibit a tradeoff between non-collusion and extensibility: (i) Non-collusion sacrifice: If the MO and DOes rely on third-party servers to jointly process data, extensibility is achieved (as multiple DOes can contribute simultaneously), but this requires that the servers do not collude [7], [8], [8], [9], [10], [11], [12], [18], [19], [20]. (ii) Extensibility sacrifice: Alternatively, if the MO and DOes act as their own computation nodes and use n-out-of-n secret sharing, the non-collusion assumption is avoided, but extensibility suffers because adding new DOes drastically increases overhead [43], [44]. Pencil resolves this trade-off by using a 2-party MO/DO protocol to securely train the model (relaxing non-collusion assumptions, and allowing the MO to iteratively aggregate weights after each training step, enabling seamless switching between DOes [1].

Collaborative Learning against malicious DOes: Despite these private techniques, collaborative learning frameworks remain vulnerable to malicious DOes, which can compromise model accuracy through data poisoning (by tampering with data) or model poisoning (by submitting malicious updates). Two primary techniques for defending against these attacks are *Byzantine-robust aggregation* and *verifiable computation*.

- *Byzantine-robust aggregation* aims to defend against both data poisoning and model poisoning attacks in FL. It identifies and filters out malicious updates using statistical analysis and anomaly detection [5], [6], [15], [17]. Representative rules include coordinate-wise trimmed mean, Krum/Multi-Krum, and trust-bootstrapped aggregation (FLTrust), as well as encrypted-similarity filtering (ShieldFL). These methods treat malicious updates as statistical outliers relative to a reference direction or majority vote. However, these methods struggle in non-IID settings, where benign gradients naturally diverge due to heterogeneous data distributions. This divergence makes it difficult to distinguish poisoned updates from legitimate ones [5]. Ripple instead verifies algebraic consistency of each DO's interactive non-linear computation via Paillier ZKPs and rejects forged outputs before aggregation, avoiding similarity thresholds that misfire when benign clients already disagree.
- *Verifiable computation* leverages cryptographic techniques, such as ZKPs and message verification codes (MAC), to ensure updates are correctly computed. PILE [14] studies verifiable perturbations in collaborative learning, and PVD-FL [45] verifies matrix multiplications; verifiable *non-linear* training under a single-server MO/DO setting without non-collusion remains largely open. Verifiable MPC frameworks such as SPDZ [44] realize secure computation and natively provide malicious security for n-party settings, yet this guarantee comes at the expense of extensibility. Specifically, integrating additional DOes leads to a substantial surge in both computational and communication overhead [1]. Theoretically, SPDZ supports a two-party setting ($n = 2$) with resilience against one malicious participant. Nevertheless, recent work has uncovered critical vulnerabilities spanning both its implementation—such as MAC key leakage [46]—and design, including flaws in the Truncpr protocol [47]. We compare VSMP multiplication latency against MP-SPDZ/MASCOT-style secure multiplication in Supplemental Section X-A under matched hardware, network, and security parameters. By contrast, ABY2.0 [21] is tailored for two-party secure computation but relies on the semi-honest adversarial model. Boshi Yuan et al. have formally demonstrated that ABY2.0 cannot be directly extended to malicious-secure MPC with a dishonest majority [48].

Taken together, prior FL robust-aggregation and MPC-based private training schemes either rely on heuristic filtering that often degrades under non-IID data, or sacrifice extensibility and/or non-collusion assumptions. Ripple combines extensible MO/DO collaboration without non-colluding servers with cryptographic verification of non-linear computations.

X. Conclusion

In this paper, we proposed Ripple, a private collaborative learning framework that achieves both data and model privacy while defending against model poisoning attacks. We designed protocols that integrate PHE and SS to ensure comprehensive privacy protection while leveraging ZKPs for non-linear operations to verify the correctness of computations submitted by malicious DOes. We conducted extensive experiments to provide a comprehensive comparison between Ripple and existing methods. The results demonstrated that Ripple effectively achieves both privacy protection and verifiability in various settings. In future work, we will focus on optimizing cryptographic primitives and improving protocol-level efficiency, while preserving Ripple's security guarantees.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] X. Liu, Z. Liu, Q. Li, K. Xu, and M. Xu, "Pencil: Private and extensible collaborative learning without the non-colluding assumption," in *Proc. Netw. Distrib. Syst. Secur.*, 2024.

[2] X. Ma, Q. Jiang, X. Liu, Q. Pei, J. Ma, and W. Lou, "Learning in your 'pocket': Secure collaborative deep learning with membership privacy," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 3, pp. 2641–2656, May/Jun. 2023.

[3] B. Zhao, K. R. Mopuri, and H. Bilen, "iDLG: Improved deep leakage from gradients," 2020, *arXiv:2001.02610*.

[4] M. Shen et al., "Casper: A causality-inspired defense with confounder against label inference attacks in vertical split federated learning," *IEEE Trans. Inf. Forensics Secur.*, vol. 21, pp. 1050–1064, 2026.

[5] Z. Ma, J. Ma, Y. Miao, Y. Li, and R. H. Deng, "ShieldFL: Mitigating model poisoning attacks in privacy-preserving federated learning," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 1639–1654, 2022.

[6] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," in *Proc. 28th Annu. Netw. Distrib. System Secur. Symp.*, 2021.

[7] X. Ma, J. Ma, H. Li, Q. Jiang, and S. Gao, "PDLM: Privacy-preserving deep learning model on cloud with multiple keys," *IEEE Trans. Serv. Comput.*, vol. 14, no. 4, pp. 1251–1263, Jul./Aug. 2021.

[8] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 35–52.

[9] N. Agrawal, A. Shahin, M. J. Shamsabadi Kusner, and A. Gascón, "Quotient: Two-party secure neural network training and prediction," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1231–1247.

[10] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 19–38.

[11] A. Patra and A. Suresh, "Blaze: Blazing fast privacy-preserving machine learning," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, 2020.

[12] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.

[13] M. Shen, X. Tang, W. Wang, and L. Zhu, *Security and Privacy in Web 3.0*. Berlin, Germany: Springer, 2024.

[14] X. Tang, M. Shen, Q. Li, L. Zhu, T. Xue, and Q. Qu, "PILE: Robust privacy-preserving federated learning via verifiable perturbations," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 6, pp. 5005–5023, Nov./Dec. 2023.

[15] M. Shen et al., "Byzantine-robust asynchronous federated learning via feature fingerprinting," *IEEE Trans. Inf. Forensics Secur.*, vol. 21, pp. 3944–3959, 2026.

[16] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5650–5659.

[17] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 119–129.

[18] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4 PC framework for privacy preserving machine learning," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, 2020.

[19] A. Dalskov, D. Escudero, and M. Keller, "Fantastic four: Honest-majority four-party secure computation with malicious security," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 2183–2200.

[20] N. Koti, A. Patra, R. Rachuri, and A. Suresh, "Tetrad: Actively secure 4 pc for secure training and inference," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, 2022.

[21] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved mixed-protocol secure two-party computation," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 2165–2182. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/patra

[22] P. Paillier, "Public-key cryptosystems based on composite residuosity classes," in *Proc. Adv. Cryptology—EUROCRYPT*, 1999, pp. 223–239.

[23] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of Paillier's probabilistic public-key system," in *Proc. Public Key Cryptogr.: 4th Int. Workshop Pract. Theory Public Key Cryptosystems*, Cheju Island, Korea, 2001, pp. 119–136.

[24] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to {Byzantine-Robust} federated learning," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 1605–1622.

[25] V. Shejwalkar and A. Houmansadr, "Manipulating the Byzantine: Optimizing model poisoning attacks and defenses for federated learning," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021.

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

[27] M. Shen et al., "Secure decentralized aggregation to prevent membership privacy leakage in edge-based federated learning," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 3, pp. 3105–3119, May/Jun. 2024.

[28] B. Zhao, W. Deng, X. Li, X. Liu, Q. Pei, and R. H. Deng, "SOCI+ : An enhanced toolkit for secure outsourced computation on integers," *IEEE Trans. Inf. Forensics Secur.*, vol. 19, pp. 5607–5619, 2024.

[29] W.-G. Tzeng, "Efficient 1-out-n oblivious transfer schemes," in *Proc. Public Key Cryptogr.: 5th Int. Workshop Pract. Theory Public Key Cryptosystems*, Paris, France, 2002, pp. 159–171.

[30] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*, vol. 2. Cambridge, U.K.: Cambridge Univ. Press, 2001.

[31] X. Yang et al., "An accuracy-lossless perturbation method for defending privacy attacks in federated learning," in *Proc. ACM Web Conf.*, 2022, pp. 732–742.

[32] D. Yuan, Q. Li, G. Li, Q. Wang, and K. Ren, "PriRadar: A privacy-preserving framework for spatial crowdsourcing," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 299–314, 2020.

[33] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 19–35.

[34] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 724–738.

[35] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2020, doi: 10.1145/3372297.3417872.

[36] B. Yuan, S. Yang, Y. Zhang, N. Ding, D. Gu, and S.-F. Sun, "{MD-ML }: Super fast { Privacy-Preserving} machine learning for malicious security with a dishonest majority," in *Proc. 33 rd USENIX Secur. Symp.*, 2024, pp. 2227–2244.

[37] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "BatchCrypt: Efficient homomorphic encryption for cross-silo federated learning," in *Proc. USENIX Annu. Techn. Conf.*, 2020, pp. 493–506. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/zhang-chengliang

[38] K. Mandal and G. Gong, "Privfl: Practical privacy-preserving federated regressions on high-dimensional data over mobile networks," in *Proc. ACM SIGSAC Conf. Cloud Comput. Secur. Workshop*, 2019, pp. 57–68.

[39] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, ON, Canada, Tech. Rep., 2012. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[40] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," in *Proc. Eur. Symp. Artif. Neural Netw.*, 2013, pp. 437–442. [Online]. Available: https://api.semanticscholar.org/CorpusID:6975432

[41] M. Shen, J. Meng, K. Xu, S. Yu, and L. Zhu, "MemDefense: Defending against membership inference attacks in IoT-based federated learning via pruning perturbations," *IEEE Trans. Big Data*, vol. 11, no. 5, pp. 2135–2147, Oct. 2025.

[42] C. Hu, C. Zhang, D. Lei, T. Wu, X. Liu, and L. Zhu, "Achieving privacy-preserving and verifiable support vector machine training in the cloud," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 3476–3491, 2023.

[43] K.-Y. Lam, X. Lu, L. Zhang, X. Wang, H. Wang, and S. Q. Goh, "Efficient FHE-based privacy-enhanced neural network for trustworthy AI-as-a-service," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 5, pp. 4451–4468, Sep./Oct. 2024.

[44] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "SPDZ2^k: Efficient MPC mod 2^k for dishonest majority," in *Proc. Annu. Int. Cryptology Conf.*, 2018, pp. 769–798.

[45] J. Zhao, H. Zhu, F. Wang, R. Lu, Z. Liu, and H. Li, "PVD-FL: A privacy-preserving and verifiable decentralized federated learning framework," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 2059–2073, 2022.

[46] A. Kyster, F. H. Nielsen, S. Oechsner, and P. Scholl, "Rushing at SPDZ: On the practical security of malicious MPC implementations," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2025, pp. 2491–2508, doi: 10.1109/SP61157.2025.00176.

[47] G. Lin et al., "Is MPC secure? Leveraging neural network classifiers to detect data leakage vulnerabilities in MPC implementations," in *Proc. IEEE Symp. Secur. Privacy*, 2025, pp. 2809–2826.

[48] B. Yuan, S. Yang, Y. Zhang, N. Ding, D. Gu, and S.-F. Sun, "MD-ML: Super fast privacy-preserving machine learning for malicious security with a dishonest majority," in *Proc. 33rd USENIX Secur. Symp.*, Philadelphia, PA, USA, 2024, pp. 2227–2244. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/yuan

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 07:02:54 UTC from IEEE Xplore. Restrictions apply.