Secure and Verifiable Data Collaboration with Low-Cost Zero-Knowledge Proofs

Yizheng Zhu
National University of Singapore
zhuyizheng@gmail.com

Yuncheng Wu[†*]
Renmin University of China
wuyuncheng@ruc.edu.cn

Zhaojing Luo[†]
Beijing Institute of Technology
zjluo@bit.edu.cn

Beng Chin Ooi
National University of Singapore
ooibc@comp.nus.edu.sg

Xiaokui Xiao
National University of Singapore
xkxiao@nus.edu.sg

ABSTRACT

Federated Learning (FL) emerges as a viable solution to facilitate data collaboration, enabling multiple clients to collaboratively train a machine learning (ML) model under the supervision of a central server while ensuring the confidentiality of their raw data. However, existing studies have unveiled two main risks: (i) the potential for the server to infer sensitive information from the client's uploaded updates (i.e., model gradients), compromising client input privacy, and (ii) the risk of malicious clients uploading malformed updates to poison the FL model, compromising input integrity. Recent works utilize secure aggregation with zero-knowledge proofs (ZKP) to guarantee input privacy and integrity in FL. Nevertheless, they suffer from extremely low efficiency and, thus, are impractical for real deployment. In this paper, we propose a novel and highly efficient approach RiseFL for secure and verifiable data collaboration, ensuring input privacy and integrity simultaneously. Firstly, we devise a probabilistic integrity check method that transforms strict checks into a hypothesis test problem, offering great optimization opportunities. Secondly, we introduce a hybrid commitment scheme to satisfy Byzantine robustness with improved performance. Thirdly, we present an optimized ZKP generation and verification technique that significantly reduces the ZKP cost based on probabilistic integrity checks. Furthermore, we theoretically prove the security guarantee of RiseFL and provide a cost analysis compared to state-of-the-art baselines. Extensive experiments on synthetic and real-world datasets suggest that our approach is effective and highly efficient in both client computation and communication. For instance, RiseFL is up to 28x, 53x, and 164x faster than baselines ACORN, RoFL, and EIFFeL for the client computation.

PVLDB Reference Format:
Yizheng Zhu, Yuncheng Wu, Zhaojing Luo, Beng Chin Ooi, Xiaokui Xiao. Secure and Verifiable Data Collaboration with Low-Cost Zero-Knowledge Proofs. PVLDB, 17(9): 2321 - 2334, 2024.
doi:10.14778/3665844.3665860

†This work was done when the authors were at National University of Singapore.
*Yuncheng Wu is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 9 ISSN 2150-8097.
doi:10.14778/3665844.3665860

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/nusdbsystem/risefl.

1 INTRODUCTION

Organizations and companies are progressively embracing digitization and digitalization as pathways to transformation, targeting enhancements in profitability, efficiency, or sustainability. Data plays a crucial role in such transformations. For example, financial analysts may use users' historical data to adjudicate on credit card applications, and clinicians may use patients' electronic health records for disease diagnosis. While many organizations may lack expansive or pertinent datasets, there is a growing inclination towards data collaboration [5, 64, 68] for analytics purposes. Nevertheless, the ascent of rigorous data protection legislation, such as GDPR [1], deters direct raw data exchange. For example, in a healthcare data collaboration scenario shown in Figure 1, three hospitals aim to share their respective organ image databases to build a more accurate diagnosis machine learning (ML) model [40, 47]. However, due to the highly sensitive nature of patients' data, directly sharing those organ images among hospitals is not permissible.

In this landscape, federated learning [4, 19, 21–23, 35, 38, 39, 43, 63, 67, 73, 80] emerges as a viable solution to facilitate data collaboration. It enables multiple data owners (i.e., clients) to collaboratively train an ML model without necessitating the direct sharing of raw data, thereby complying with data protection acts. Typically, there is a centralized server that coordinates the FL training process. Take Figure 1 as an example. The hospitals train an FL model under the coordination of a healthcare center (i.e., server). The healthcare center first initializes the model parameter and broadcasts it to all the hospitals. Subsequently, in each iteration, each hospital computes a local update, i.e., model gradients, on its own patients' data and uploads it to the healthcare center. The healthcare center then aggregates all hospitals' updates to generate a global update and sends it back to the hospitals for iterative training [43]. Finally, the hospitals obtain a federated model trained on the three hospitals' organ images. After that, doctors in each hospital can use it to assist in diagnosing new patients.

Despite the fact that FL could facilitate data collaboration among multiple clients, two main risks remain. The first is the client's *input privacy*. Even without disclosing the client's raw data to the server, recent studies [44, 46] have shown that the server can recover the client's sensitive data through the uploaded update with a high probability. The second is the client's *input integrity*.

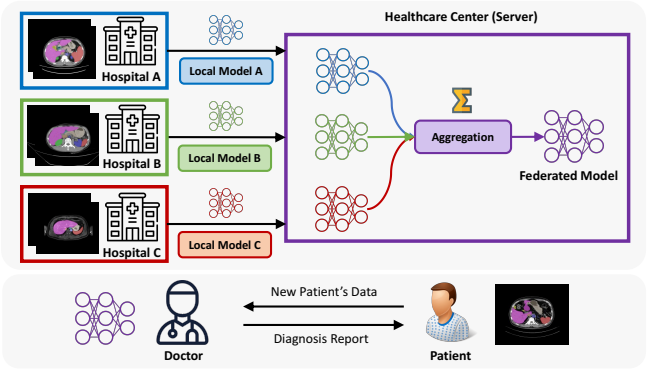


Figure 1: An example for healthcare data collaboration.

In FL, there may exist a set of malicious clients that aim to poison the collaboratively trained model via Byzantine attacks, such as imposing backdoors so that the model is susceptible to specific input data [3, 71], contaminating the training process with malformed updates to degrade model accuracy [9, 24, 30], and so on.

A number of solutions [7, 15, 32, 42, 48, 74, 78, 79, 81] have been proposed to protect input privacy and ensure input integrity in FL. On the one hand, instead of uploading the plaintext local updates to the server, the clients can utilize secure aggregation techniques [7, 12, 32], such as secret sharing [33, 57] and homomorphic encryption [18, 25], to mask or encrypt the local updates so that the server can aggregate the clients' updates correctly without knowing each update. In this way, the client's input privacy is preserved. However, these solutions do not ensure input integrity because it is difficult to distinguish malicious encrypted updates from benign ones. On the other hand, [15, 42, 48, 74, 78, 79] present various Byzantine-robust aggregation algorithms, allowing the server to identify malformed updates and eliminate them from being aggregated into the global update. Nevertheless, these algorithms require the clients to send plaintext updates to the server for integrity verification, compromising the client's input privacy.

In order to ensure input integrity while satisfying input privacy, [6, 41, 55] use secure aggregation to protect each client's update and allow the server to check the encrypted update's integrity using zero-knowledge proof (ZKP) [36] protocols. The general idea is to let each client compute a commitment of its local update and generate a proof that the update satisfies a publicly-known predicate, for example, the L_2-norm is within a specific range [59]; then, the server can verify the correctness of the proofs based on the commitments and securely aggregate the valid updates without the need of knowing the plaintext values. Unfortunately, these solutions suffer from extremely low efficiency in proof generation and verification, making them impractical for real deployments. For example, under the experiment settings in Section 6.2, EIFFeL [55] takes 161 seconds to generate and verify proofs on 10K model parameters, compared to 7.6 seconds for client local training.

To introduce a practical FL system that ensures both input privacy and input integrity, we propose a secure and verifiable federated learning approach, called RiseFL, with high efficiency. RiseFL has been included in our federated learning system Falcon[1] so as to enhance the trustworthiness of the clients' model updates while preserving their privacy [68]. In this paper, we focus on the L_2-norm integrity check, i.e., the L_2-norm of a client's local update is less than a threshold, which is widely adopted in existing works [16, 41, 55, 61]. Our key observation of the low-efficiency in [6, 41, 55] is that their proof generation and verification costs for each client are linearly dependent on the number of parameters d in the FL model. Therefore, we aim to reduce the proof cost to render system scalability and practicality for handling large FL models. To this end, we propose a novel approach with the following three key features. First, we propose a probabilistic L_2-norm check method that transforms the strict L_2-norm check into a Chi-square hypothesis test. The rationale is to sample a set of public vectors and let each client prove that the summation of the inner products between its update and the public vectors is bounded, instead of directly checking the L_2-norm of the update. This transformation facilitates the development of optimization techniques to reduce the complexity. Second, we devise a hybrid commitment scheme based on Pedersen commitment [50] and verifiable Shamir secret sharing commitment [20, 57], tailored for the probabilistic check method. It not only ensures Byzantine robustness but also achieves notable performance improvement on the client computation. Third, we design an optimized ZKP generation and verification technique by merging the common group exponentiations across the clients for the probabilistic check method, significantly reducing the cryptographic operation cost of proof generation and verification from $O(d)$ to $O(d/\log d)$. Although we consider the L_2-norm check, our approach can be easily extended to various Byzantine-robust integrity checks [15, 60, 78] based on different L_2-norm variants, such as cosine similarity [3, 15], sphere defense [60], and Zeno++ [72].

In summary, we make the following contributions.

- We propose a novel and highly efficient solution RiseFL for FL-based data collaboration, which simultaneously ensures each client's input privacy and input integrity.
- We introduce a probabilistic L_2-norm integrity check method coupled with two novel techniques for commitment generation and ZKP generation/verification, which significantly reduces the overall cost for the check.
- We provide a formal security analysis of RiseFL and theoretically compare its computational and communication costs with three state-of-the-art solutions.
- We implement RiseFL and evaluate its performance with a set of micro-benchmark experiments as well as FL tasks on three real-world datasets. The results demonstrate that RiseFL is effective in detecting various attacks and is up to 28x, 53x and 164x faster than ACORN [6], RoFL [41] and EIFFeL [55] for the client computation, respectively.

The rest of the paper is organized as follows. Section 2 introduces preliminaries, and Section 3 presents an overview of our approach. We detail the system design in Section 4 and analyze the security and cost in Section 5. The evaluation is given in Section 6. We review related works in Section 7 and conclude the paper in Section 8.

2 PRELIMINARIES

We first introduce the notations used in this paper. Let $\mathbb{G}$ denote a cyclic group with prime order p, where the discrete logarithm

[1] https://www.comp.nus.edu.sg/~dbsystem/fintech-Falcon/

problem [52] is hard. Let $\mathbb{Z}_p$ denote the set of integers modulo the prime p. We use x, $\mathbf{x}$, and $\mathbf{X}$ to denote a scalar, a vector, and a matrix, respectively. We use $\mathsf{Enc}_{\mathsf{K}}(x)$ to denote an encrypted value of x under an encryption key K, and $\mathsf{Dec}_{\mathsf{K}}(y)$ to denote a decrypted value of y under the same key K. Since the data used in ML are often floating-point values, we use fixed-point integer representation to encode floating-point values. In the following, we introduce the cryptographic building blocks used in this paper.

Pedersen Commitment. A commitment scheme is a cryptographic primitive that allows one to commit a chosen value without revealing the value to others while still allowing the ability to disclose it later [28]. Commitment schemes are widely used in various zero-knowledge proofs. In this paper, we adopt Pedersen commitment [50] for a party to commit its secret value $x \in \mathbb{Z}_p$. Given independent group elements (g, h), the party generates a random number $r \in \mathbb{Z}_p$ and uses Pedersen commitment to encrypt the secret value x by a commitment algorithm $C(x, r) = g^x h^r$. Later, the party can reveal its x and r such that a verifier can compute $C'(x, r) = g^x h^r$ and check if $C'(x, r) = C(x, r)$. If matched, the verifier has proven that the party's committed value in $C(x, r)$ is indeed x. Notice that if provided the commitment $C(x, r)$ and the random number r, the value of x can be computed based on $g^x = C(x, r) \cdot h^{-r}$. Another property of Pedersen commitment is that it is additively homomorphic. Given two values x_1, x_2 and two random numbers r_1, r_2, the commitment follows $C(x_1, r_1) \cdot C(x_2, r_2) = C(x_1 + x_2, r_1 + r_2)$. These properties enable us to design a novel scheme built on Pedersen commitment, which securely aggregates the parties' secret values without revealing them.

Verifiable Shamir's Secret Sharing Scheme. Shamir's t-out-of-n secret sharing (SSS) scheme [57] allows a party to distribute a secret among a group of n parties via shares so that the secret can be reconstructed given any t shares but cannot be revealed given less than t shares. The SSS scheme is verifiable (aka. VSSS) if auxiliary information is provided to verify the validity of the secret shares. We use the VSSS scheme [20, 57] to share a number $r \in \mathbb{Z}_p$. Specifically, the VSSS scheme consists of three algorithms, namely, SS.Share, SS.Verify, and SS.Recover.

- $((1, r_1), \ldots, (n, r_n), \Psi) \leftarrow \mathsf{SS.Share}(r, n, t, g)$. Given a secret $r \in \mathbb{Z}_p$, $g \in \mathbb{G}$, and $0 < t \leq n$, this algorithm outputs a set of n shares (i, r_i) for $i \in [n]$ and a check string Ψ as the auxiliary information to verify the shares.
- $r \leftarrow \mathsf{SS.Recover}(\{(i, r_i)\} : i \in A)$. For any subset $A \subset [n]$ with size at least t, this algorithm recovers the secret r.
- $\mathsf{True/False} \leftarrow \mathsf{SS.Verify}(\Psi, i, r_i, n, t, g)$. Given a share (i, r_i) and the check string Ψ, it verifies the validity of this share so that it outputs True if (i, r_i) was indeed generated by $\mathsf{SS.Share}(r, n, t, g)$ and False otherwise.

This scheme is additively homomorphic in both the shares and the check string. If $((1, r_1), \ldots, (n, r_n), \Psi_r) \leftarrow \mathsf{SS.Share}(r, n, t, g)$ and $((1, s_1), \ldots, (n, s_n), \Psi_s) \leftarrow \mathsf{SS.Share}(s, n, t, g)$, then:

- $r + s \leftarrow \mathsf{SS.Recover}(\{(i, r_i + s_i)\} : i \in A)$ for any subset $A \subset [n]$ with size at least t,
- $\mathsf{True} \leftarrow \mathsf{SS.Verify}(\Psi_r \cdot \Psi_s, i, r_i + s_i, n, t, g)$.

Zero-Knowledge Proofs. A zero-knowledge proof (ZKP) allows a prover to prove to a verifier that a given statement is true, such as a value is within a range, without disclosing any additional information to the verifier [11]. We utilize two additively homomorphic ZKP protocols based on the Pedersen commitment as building blocks. Note that the zkSNARK protocols [8, 26, 49] are not additively homomorphic, thus they cannot support the secure aggregation required in federated learning.

The first is the Σ-protocol [14] for proof of square and proof of relation. For proof of square $((x, r_1, r_2), (y_1, y_2))$, denote (g, h) as the independent group elements, and let $y_1 = C(x, r_1)$ and $y_2 = C(x^2, r_2)$ be the Pedersen commitments, where $x, r_1, r_2 \in \mathbb{Z}_p$ are the secrets. The function $\mathsf{GenPrfSq}((x, r_1, r_2), (y_1, y_2))$ generates a proof π that the secret value in y_2 is the square of the secret value in y_1. Accordingly, the function $\mathsf{VerPrfSq}(\pi, y_1, y_2)$ verifies the correctness of this proof. Similarly, for proof of a relation $((r, v, s), (z, e, o))$, denote (g, q, h) as the independent group elements, and let $z = g^r, e = g^v h^r, o = g^v q^s$ be the Pedersen commitments, where $r, v, s \in \mathbb{Z}_p$ are the secrets. The function $\mathsf{GenPrfWf}((r, v, s), (z, e, o))$ generates a proof π that the secrets in e and o are equal, and the secret in z is equal to the blind in e. The function $\mathsf{VerPrfWf}(\pi, z, e, o)$ verifies the proof. The second ZKP protocol used is the Bulletproofs protocol [13] for checking the bound of a value x with its Pedersen commitment $y = C(x, r)$. We denote $\mathsf{GenPrfBd}(x, y, r, b)$ as the function that generates a proof π that $x \in [0, 2^b)$, where 2^b is the bound to be ensured. The corresponding function $\mathsf{VerPrfBd}(\pi, y, b)$ verifies the proof. We refer the interested readers to [13] for more details.

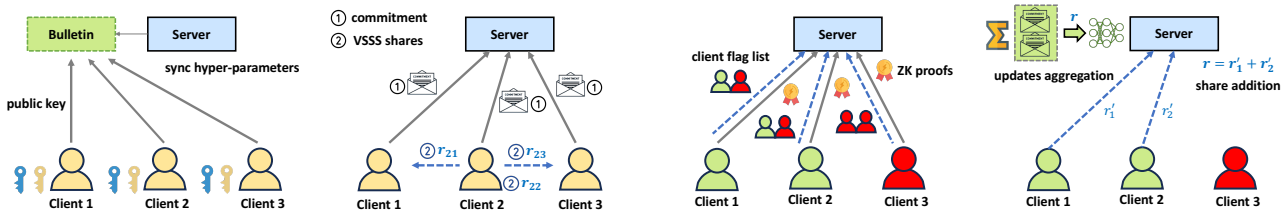
3 SYSTEM OVERVIEW

3.1 System Model

There are n clients $\{\mathcal{C}_1, \cdots, \mathcal{C}_n\}$ and a centralized server in the system. Each client $\mathcal{C}_i (i \in [n])$ holds a private dataset $\mathcal{D}_i$ to participate in data collaboration for training a federated learning (FL) model $\mathcal{M}$. Let d be the number of parameters in $\mathcal{M}$. In each iteration, the FL training process consists of three phases. Firstly, the server broadcasts the current model parameters to all the clients. Secondly, each client $\mathcal{C}_i$ computes a local update $\mathbf{u}_i$ (i.e., model gradients) given the model parameters and its dataset $\mathcal{D}_i$, and submits $\mathbf{u}_i$ to the server. Thirdly, the server aggregates the clients' gradients to a global update $\mathcal{U} = \sum_{i \in [n]} \mathbf{u}_i$ and updates the model parameters of $\mathcal{M}$ for the next round of training until convergence.

3.2 Threat Model

We consider a malicious threat model in two aspects. First, regarding input privacy, we consider a malicious server (i.e., the adversary) that can deviate arbitrarily from the specified protocol to infer each client's uploaded model update. Also, the server may collude with some of the malicious clients to compromise the honest clients' privacy. Similar to [55], we do not consider the scenario that the server is malicious against the input integrity because its primary goal is to ensure the well-formedness of each client's uploaded update. Second, regarding input integrity, we assume that there are at most m malicious clients in the system, where $m < n/2$. The malicious clients can also deviate from the specified protocol arbitrarily, such as sending malformed updates to the server to poison the aggregation of the global update, or intentionally marking an



(a) System initialization (b) Commitment generation (c) Proof generation and verification (d) Secure aggregation

Figure 2: An overview of the proposed RiseFL system.

honest client as malicious to interfere with the server's decision on the list of malicious clients.

3.3 Problem Formulation

We aim to ensure both input privacy (for the clients) and input integrity (for the server) under the threat model described in Section 3.2. We formulate our problem as a relaxed variant of the secure aggregation with verified inputs (SAVI) problem in [55], namely (D, F)-relaxed SAVI, as defined in Definition 1. It achieves the same level of input privacy while relaxing the input integrity for significant efficiency improvement.

Definition 1. *Given a security parameter κ, a function $D : \mathbb{R}^d \to \mathbb{R}$ satisfying that $\boldsymbol{u}$ is malicious if and only if $D(\boldsymbol{u}) > 1$, a function $F : (1, +\infty) \to [0, 1]$, a set of inputs $\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n\}$ from clients $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ respectively, and a list of honest clients $\mathcal{C}_H$, a protocol Π is a (D, F)-relaxed SAVI protocol for $\mathcal{C}_H$ if:*

- *Input Privacy. The protocol Π realizes the ideal functionality $\mathcal{F}$ such that for an adversary $\mathcal{A}$ that consists of the malicious server and the malicious clients $\mathcal{C} \setminus \mathcal{C}_H$ attacking the real interaction, there exist a simulator $\mathcal{S}$ attacking the ideal interaction, and*

$$|\Pr[\text{Real}_{\Pi,\mathcal{A}}(\{\boldsymbol{u}_{\mathcal{C}_H}\}) = 1] - \Pr[\text{Ideal}_{\mathcal{F},\mathcal{S}}(\mathcal{U}_{\mathcal{C}_H}) = 1]| \leq \text{negl}(\kappa),$$

  where $\mathcal{U}_{\mathcal{C}_H} = \sum_{\mathcal{C}_i \in \mathcal{C}_H} \boldsymbol{u}_i$.
- *Input Integrity. The protocol Π outputs $\sum_{\mathcal{C}_i \in \mathcal{C}_{\text{Valid}}} \boldsymbol{u}_i$ with probability of at least $1 - \text{negl}(\kappa)$, where $\mathcal{C}_{\text{Valid}}$ is the set of clients that pass the integrity check and $\mathcal{C}_H \subseteq \mathcal{C}_{\text{Valid}}$. For any malformed input $\boldsymbol{u}_j$ from a malicious client $\mathcal{C}_j$, the probability that it passes the integrity check satisfies:*

$$\Pr[\mathcal{C}_j \in \mathcal{C}_{\text{Valid}}] \leq F(D(\boldsymbol{u}_j)).$$

For input privacy in Definition 1, it ensures that the server can only learn the aggregation of honest clients' updates. For input integrity, Definition 1 relaxes the integrity verification by introducing a malicious pass rate function F. F is a function that maps the degree of maliciousness of an input to the pass rate of the input, and the degree of maliciousness is measured by the function D. For instance, with an L_2-norm bound B, a natural choice of D is $D(\mathbf{u}) = \|\mathbf{u}\|_2/B$, and the pass rate of a malicious $\mathbf{u}$ is $F(D(\mathbf{u}))$. Intuitively, the higher the degree of maliciousness, the lower the pass rate. So, F is usually decreasing. Our system also satisfies $\limsup_{x \to +\infty} F(x) \leq \text{negl}(\kappa)$. When $F \equiv \text{negl}(\kappa)$, a protocol that satisfies (D, F)-relaxed SAVI will also satisfy SAVI.

3.4 Solution Overview

To solve the problem in Definition 1, we propose a secure and verifiable federated learning system RiseFL with high efficiency. It tolerates $m < n/2$ malicious clients for input integrity, which means that the server can securely aggregate the clients' inputs as long as a majority of the clients are honest. Figure 2 gives an overview of RiseFL, which is composed of a system initialization stage and three iterative rounds: commitment generation, proof generation and verification, and secure aggregation. In the initialization stage (Figure 2a), all the parties agree on some hyper-parameters, such as the number of clients n, the maximum number of malicious clients m, the security parameters, e.g., key size, and so on.

In each iteration of the FL training process, each client $\mathcal{C}_i (i \in [n])$ commits its model update $\mathbf{u}_i$ using the hybrid commitment scheme based on Pedersen commitment and verifiable Shamir's secret sharing (VSSS) in Section 4.3, and then sends the commitment to the server and the secret shares to the corresponding clients (see Figure 2b). In the proof generation and verification round (Figure 2c), there are two steps. In the first step, each client verifies the authenticity of other clients' secret shares. For the secret shares that are verified to be invalid, the client marks the respective clients as malicious. With the marks from all clients, the server can then identify a subset of malicious clients. In the second step, the server uses a probabilistic integrity check method presented in Section 4.4 to check each client's update $\mathbf{u}_i$. Next, the server filters out the malicious client list $\mathcal{C}^*$ and broadcasts it to all the clients. In the secure aggregation round (Figure 2d), each client aggregates the secret shares from clients $\mathcal{C}_j (j \notin \mathcal{C}^*)$ and sends the result to the server. The server reconstructs the sum of secret shares and securely aggregates the updates $\mathbf{u}_j (j \notin \mathcal{C}^*)$ based on the Pedersen commitments.

4 RISEFL DESIGN

In this section, we introduce our system design. We first present the rationale of the protocol in Section 4.1. Then, we introduce the initialization stage in Section 4.2 and the three steps in each iteration of the training stage in Sections 4.3-4.5, respectively. Finally, we discuss the extension of our system in Section 4.6.

4.1 Rationale

The most relevant work to our problem is EIFFeL [55], which also ensures input privacy and integrity in FL training. In EIFFeL, each

Algorithm 1: Probabilistic L_2-norm bound check

Input: $\mathbf{u} \in \mathbb{R}^d$, B, k, ϵ
Output: "Pass" or "Fail"
Sample $\mathbf{a}_1, \ldots, \mathbf{a}_k \in \mathbb{R}^d$ i.i.d. from $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$
Compute $\gamma_{k,\epsilon}$ which satisfies that $\Pr_{t \sim \chi^2_k}[t < \gamma_{k,\epsilon}] = 1 - \epsilon$
if $\sum_{i=t}^{k} \langle \boldsymbol{a}_i, \boldsymbol{u} \rangle^2 \leq B^2 \gamma_{k,\epsilon}$ **then**
    return *"Pass"*
else
    return *"Fail"*
end

client first shares every coordinate of its model update with other clients via verifiable Shamir's t-out-of-n secret sharing (VSSS) [57] as commitments. Then, each client generates a secret-shared non-interactive proof (SNIP) [16] and sends it to the other clients via VSSS. Next, the clients, together with the server, act as the verifiers to check the correctness of clients' secret-shared proofs. Finally, the server and the honest clients who pass the verification can securely aggregate the valid model updates with VSSS. However, its efficiency is low, and thus is impractical to be deployed in real-world systems. For example, under the experiment settings in Section 6.2, given 100 clients and 1K model parameters, EIFFeL takes around 16.2 seconds for proof generation and verification on each client. More severely, the cost is increased to 161 seconds when the number of model parameters d is 10K. The underlying reason is two-fold. First, it requires the client to generate the check strings w.r.t. d coordinates for the commitments, which involves $O(md)$ cryptographic group exponentiations (g.e.) to tolerate m malicious clients. Second, the complexity of its proof generation and verification is almost linearly dependent on the number of coordinates d in the model update, making EIFFeL inefficient and less scalable.

The rationale behind our idea is to reduce the complexity of expensive group exponentiations. To do so, we first design a probabilistic L_2-norm bound check method, as shown in Algorithm 1. Instead of generating and verifying proofs for the L_2-norm of an update $||\mathbf{u}||_2$, where $\mathbf{u} \in \mathbb{R}^d$, we randomly sample k points $\mathbf{a}_1, \ldots, \mathbf{a}_k$ from the normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$. Then, the random variable

$$\frac{1}{||\mathbf{u}||_2^2} \sum_{t=1}^{k} \langle \mathbf{a}_t, \mathbf{u} \rangle^2, \tag{1}$$

follows a Chi-square distribution χ^2_k with k degrees of freedom. In Algorithm 1, if $||\mathbf{u}||_2 \leq B$, then the probability that $\mathbf{u}$ passes the check is at least $1 - \epsilon$, where ϵ is chosen to be cryptographically small, e.g., 2^{-128}. In this way, the probability that the client fails the check is of the same order as the probability that the client's encryption is broken. Figure 3 shows an example of this method. Assume that $||\mathbf{u}||_2 = 1$. We sample two random normal samples $\mathbf{a}_1, \mathbf{a}_2$; then, the inner products $\langle \mathbf{a}_1, \mathbf{u} \rangle, \langle \mathbf{a}_2, \mathbf{u} \rangle$ are the projections of $\mathbf{a}_1, \mathbf{a}_2$ onto $\mathbf{u}$. Figure 3b gives the probability density function of χ^2_k. With the specific bound $\gamma_{k,\epsilon}$ as computed in Algorithm 1, there is an overwhelming probability such that $\sum_{t=1}^{k} \langle \mathbf{a}_t, \mathbf{u} \rangle^2 \leq \gamma_{k,\epsilon}$.

We note that the probabilistic check method does not directly reduce complexity, as the ZKP proof w.r.t. $\sum_{t=1}^{k} \langle \mathbf{a}_t, \mathbf{u} \rangle^2$ is still dependent on d. Nevertheless, this method allows us to design optimization techniques to reduce the complexity. Specifically, we introduce a hybrid commitment scheme, which commits $\mathbf{u}$ by Pedersen commitment with $O(d)$ g.e. and commits the random secret used in the Pedersen commitment by VSSS with $O(m)$ g.e., thereby reducing commitment generation complexity by a factor of $O(m)$ compared to EIFFeL. Moreover, we propose an optimized ZKP generation and verification technique by carefully merging common group elements across clients required for each sample vector in the probabilistic test, achieving a cost reduction by a factor of $O(\log d)$ compared to EIFFeL. We shall detail these techniques in the following subsections and theoretically analyze the cost in Section 5.2.

4.2 System Initialization

In this stage, all parties (the server and the clients) are given the system parameters, including the number of clients n, the maximum number of malicious clients m, the bound on the number of bits b_{ip} of each inner product, the maximum number of bits $b_{\max} > b_{\text{ip}}$ of the sum of squares of inner product, the bound of the sum of inner products $B_0 < 2^{b_{\max}}$, the number of samples k for the probabilistic check, a set of independent group elements $g, q \in \mathbb{G}, \mathbf{w} = (w_1, \cdots, w_d) \in \mathbb{G}^d$, the factor $M > 0$ used in discretizing the normal distribution samples, and a cryptographic hash function $H(\cdot)$. Note that b_{ip} and $b_{\max}$ sets the bounds of $\langle \mathbf{a}_i, \mathbf{u} \rangle$ and $\sum_{i=1}^{k} \langle \mathbf{a}_i, \mathbf{u} \rangle^2$ in Eqn 1, respectively.

Since there is no direct channel between any two clients in the FL setting considered in this paper, we let the server forward some of the messages. To prevent the server from accessing the secret information, each client $\mathcal{C}_i (i \in [n])$ generates a public/private key pair (pk_i, sk_i) and sends the public key pk_i to a public bulletin. Subsequently, each client fetches the other clients' public keys such that each pair of clients can establish a secure channel via the Diffie-Hellman protocol [45] for exchanging messages securely.

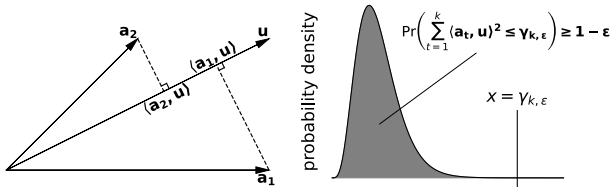
4.3 Commitment Generation

Recall that in each iteration of the FL training process, each client $\mathcal{C}_i (i \in [n])$ obtains a model update $\mathbf{u}_i$ via local training on its dataset $\mathcal{D}_i$. In order to prove to the server that the L_2-norm of $\mathbf{u}_i$ is within a bound B_0, the client $\mathcal{C}_i$ needs to commit its update $\mathbf{u}_i$ before generating the proofs. In RiseFL, the server is expected to not only identify malicious clients, but also aggregate well-formed model updates so that the training process is not affected by malicious clients. Therefore, we propose a novel *hybrid commitment scheme* based on Pedersen commitment and VSSS, where VSSS is used to protect the random secret in the Pedersen commitment.

Hybrid commitment scheme. Note that the clients and server agree on independent group elements $g, w_1, \ldots, w_d \in \mathbb{G}$, where $w_j (j \in [d])$ is used for committing the j-th coordinate in $\mathbf{u}_i$. Then, $\mathcal{C}_i$ generates a random secret $r_i \in \mathbb{Z}_p$ and encrypts $\mathbf{u}_i$ with Pedersen commitment as follows:

$$\begin{aligned} \mathbf{y}_i = C(\mathbf{u}_i, r_i) &= (C(u_{i1}, r_i), \ldots, C(u_{id}, r_i)) \\ &= (g^{u_{i1}} w_1^{r_i}, \ldots, g^{u_{id}} w_d^{r_i}), \end{aligned} \tag{2}$$

where u_{ij} is the j-th coordinate in $\mathbf{u}_i$. Each client $\mathcal{C}_i$ sends $\mathbf{y}_i = C(\mathbf{u}_i, r_i)$ and $z_i = g^{r_i}$ to the server as commitments. As r_i is held by each client $\mathcal{C}_i$, the server knows nothing regarding each update $\mathbf{u}_i$.



(a) Sample random vectors and compute inner products

(b) Check the bound of the sum of inner products

Figure 3: The probabilistic check when $||\mathbf{u}||_2 = 1$.

To facilitate the server to aggregate well-formed updates, we also require each client $\mathcal{C}_i$ to share its secret r_i with other clients using VSSS. Specifically, $\mathcal{C}_i$ computes $((1, r_{i1}), \ldots, (n, r_{in}), \Psi_{r_i}) \leftarrow \mathsf{SS.Share}(r_i, n, m+1, g)$ and sends $((j, r_{ij}), \Psi_{r_i})$ to $\mathcal{C}_j$. Note that the equality $\Psi_{r_i}(0) = z_i$ holds. Since the clients do not have communication channels, $\mathcal{C}_i$ encrypts each share r_{ij} with the corresponding symmetric key established by (pk_j, sk_i) for each client $\mathcal{C}_j (j \in [n])$, obtaining $\mathsf{Enc}(r_{ij})$. The encrypted $\mathsf{Enc}(r_{ij})$ ensures that only $\mathcal{C}_j$ can decrypt it. Finally, $\mathcal{C}_i$ sends the commitment $\mathbf{y}_i$, the encrypted share $\mathsf{Enc}(r_{ij})(j \in [n])$, and the check string Ψ_{r_i} to the server.

Example 1. Suppose in Figure 2b, Client 2 has a model update $\mathbf{u}_2 = (5, 1)$ and its generates a random secret $r_2 = 10$. Let $w_1 = g^3, w_2 = g^4$. Client 2 can commit $\mathbf{y}_2 = (g^5 \cdot w_1^{10}, g^1 \cdot w_2^{10}) = (g^{35}, g^{41})$ and $z_2 = g^{r_2} = g^{10}$. Then, it can generate secret shares of r_2 as $(r_{21}, r_{22}, r_{23}) = (3, -4, -11)$ using the polynomial $-7x + 10$ and the check string $\Psi_{r_2} = (g^{10}, g^{-7})$, where $r_{21} = 3$ and $r_{23} = -11$ will be encrypted using keys shared with Client 1 and Client 3 respectively before they are sent to the server together with the check string. For the equation $\Psi_{r_2}(0) = z_2$, both sides equal g^{10}.

After receiving the messages from all clients, the server forwards the encrypted shares $\mathsf{Enc}(r_{ij})(i \in [n])$ to client $\mathcal{C}_j$, and broadcasts the check strings $\Psi_{r_i}(i \in [n])$ to all clients. We will introduce how the server aggregates well-formed updates from honest clients based on these commitments in Section 4.5.

4.4 Proof Generation and Verification

In this round, the clients and the server jointly flag the malicious clients. Specifically, the server initializes a list $\mathcal{C}^* = \varnothing$ to record the malicious clients that will be identified in the current iteration. This round consists of two steps: verifying the authenticity of secret shares and verifying the integrity of each client's update.

4.4.1 Verify the authenticity of secret shares. After receiving the encrypted shares $\mathsf{Enc}(r_{ji})$ and check strings Ψ_{r_j} for $j \in [n]$, each client $\mathcal{C}_i$ decrypts r_{ji} and checks against Ψ_{r_j}. If a check fails, $\mathcal{C}_i$ marks the corresponding client $\mathcal{C}_j$ as malicious because the share is not authenticated. Subsequently, $\mathcal{C}_i$ posts the list of clients that do not pass the check on the public bulletin board.

The server follows two rules to flag malicious clients [55]. First, if a client $\mathcal{C}_i$ flags more than m clients as malicious or is flagged as malicious by more than m clients, the server puts $\mathcal{C}_i$ to $\mathcal{C}^*$. This is because there are at most m malicious clients in the system. An honest client cannot mark more than m clients as malicious clients and will not be marked as malicious by more than m clients. For example, in Figure 2c, Client 3 marks the other two clients as malicious, then the server can ensure that this client is malicious.

Second, if a client $\mathcal{C}_i$ is flagged as malicious by $[1, m]$ clients, it sends the shares r_{ij} to the server in the clear for all $\mathcal{C}_j$ that flags $\mathcal{C}_i$. The server then checks them against Ψ_{r_i}. If any of the clear r_{ij} fails the check, the server puts $\mathcal{C}_i$ to $\mathcal{C}^*$. Note that any client that marks an honest $\mathcal{C}_i$ as malicious must be malicious. If client $\mathcal{C}_i$ reads from the public bulletin board that more than m clients mark $\mathcal{C}_i$ as malicious, it quits the protocol as this should not happen.

4.4.2 Verify the integrity of each client's update. Next, we present how the server and clients jointly execute the probabilistic integrity check described in Section 4.1. Without loss of generality, we describe the check for one client $\mathcal{C}_i$.

Sampling k random vectors. At the beginning, the server and clients need to agree on k random samples $\mathbf{a}_1, \cdots, \mathbf{a}_k$ with dimensionality d. Intuitively, we can let the server sample the k vectors and broadcast them to all clients. However, there are two main issues. First, the communication cost can be very high. As will be discussed in Section 5.1, the choice of k typically requires to be several thousand. Therefore, it is communication inefficient for the server to broadcast k vectors (each vector contains d elements). Second, the server may select special vectors that are not random, i.e., the vectors may allow the server to infer the client's update easily. For example, if the server selects $\mathbf{a}_t = (1, 0, \ldots, 0)$ for all $t \in [k]$, it may infer the range of the first element in a client's update.

To solve these issues, we let the server select a random value s and broadcast it to all clients. Based on s, the server and each client $\mathcal{C}_i (i \in [n])$ can compute a seed $H(s, \{pk_i\}_{1 \le i \le n})$ using s and all clients' public keys. Hence, the clients and the server can generate the same set of random samples because the seed is the same. Note it is computationally infeasible to find a seed that produces specific vectors since $H(\cdot)$ is a cryptographic hash function.

Merging group elements. Given the random vectors $\mathbf{a}_1, \cdots, \mathbf{a}_k$, a naive way of realizing the probabilistic bound check in Algorithm 1 is to let client $\mathcal{C}_i$ commit to $\langle \mathbf{a}_t, \mathbf{u}_i \rangle$ with $z_t = g^{\langle \mathbf{a}_t, \mathbf{u}_i \rangle} w_{d+t}^{r_i}$ for $t \in [k]$ based on additional independent group elements $w_{d+1}, \ldots, w_{d+k}$. It can then generate a Σ-protocol [14] proof that the secret in z_t is indeed the inner product of the secrets in $\mathbf{y}_i$ and $\mathbf{a}_t$ for $t \in [k]$ and a proof that the sum of the squares of the secrets in z_t is bounded. However, its computational cost will be $O(d)$ in terms of group exponentiations. We thus propose an optimization by merging group elements to reduce the cost of proof generation to $O(d/\log d)$ group exponentiations.

We observe that the following term $e_t (t \in [k])$ is used in the inner product proof generation and verification:

$$e_t := y_{i1}^{a_{t1}} \ldots y_{id}^{a_{td}} = (g^{u_{i1}} w_1^{r_i})^{a_{t1}} \ldots (g^{u_{id}} w_d^{r_i})^{a_{td}} \tag{3}$$

$$= g^{\langle \mathbf{a}_t, \mathbf{u}_i \rangle} (w_1^{a_{t1}} \ldots w_d^{a_{td}})^{r_i}, \tag{4}$$

where $y_{i1}, \cdots, y_{id}$ are the commitments of elements in $\mathbf{u}_i$ (see Eqn 2). We merge the group elements $w_1, \ldots, w_d$ into $h_t = w_1^{a_{t1}} \ldots w_d^{a_{td}}$, which is the same for all the clients as the random vector $\mathbf{a}_t$ and the group elements $w_1, \cdots, w_d$ are shared by the server and clients. Thus, we let the server precompute the merged h_t for $t \in [k]$ and broadcast them to all clients before they generate proofs. Given h_t, the cost of computing the values e_t in Eqn 3 for $t \in [k]$ is $O(k)$ group exponentiations and $O(kd)$ finite field operations. Still, the client

Algorithm 2: VerCrt$(\mathbf{w}, \mathbf{h}, \mathbf{A})$

Input: $\mathbf{w} = (w_1, \ldots, w_d) \in \mathbb{G}^d$, $\mathbf{h} = (h_0, \ldots, h_k) \in \mathbb{Z}_p^{(k+1)}$, $\mathbf{A} \in \mathbb{M}_{(k+1)\times d}(\mathbb{Z}_p)$.

1 Randomly Sample $\mathbf{b} = (b_0, \ldots, b_k) \in \mathbb{Z}_p^{k+1}$.

2 Compute $\mathbf{c} = (c_1, \ldots, c_d) = \mathbf{b} \cdot \mathbf{A} \in \mathbb{Z}_p^d$.

3 **return** $h_0^{b_0} \ldots h_k^{b_k} == w_1^{c_1} \ldots w_d^{c_d}$.

cannot blindly compute the value in Eqn 3 by $e_t = g^{\langle \mathbf{a}_t, \mathbf{u}_i \rangle} h_t^{r_i}$ as h_t is received from the server, which could be malicious. Therefore, we let the client use batch verification[2] to check whether the merged elements h_t for $t \in [k]$ are computed correctly. The cost of batch verification is $O(d/\log d)$ group exponentiations and $O(kd)$ finite field operations. Note that the group exponentiations are the major cost as they are much more complex than finite field operations. The reduction of the number of group exponentiations from $O(d)$ to $O(d/\log d)$ significantly reduces client cost, while the additional cost of $O(kd)$ finite field operations is not large.

To make sure that e_t is indeed the commitment of the inner product of $\mathbf{a}_t$ and the secret in $\mathbf{y}_i$, the server uses batch verification to check that the values e_t, $t \in [k]$, submitted by client $\mathcal{C}_i$ satisfy Eqn 3: $e_t \stackrel{?}{=} y_{i1}^{a_{t1}} \ldots y_{id}^{a_{td}}$. The only issue at this step is that even if all of the equations $e_t = y_{i1}^{a_{t1}} \ldots y_{id}^{a_{td}}$ are satisfied, the server is still not sure that client $\mathcal{C}_i$ possesses a value $\mathbf{u}_i$ that is used to produce $\mathbf{y}_i$. To address this issue, the server additionally samples $\mathbf{a}_0 \in \mathbb{Z}_p^d$ from the uniform distribution on $\mathbb{Z}_p$ with cryptographically secure pseudo-random number generator (PRNG), computes the corresponding $h_0 = \prod_l w_l^{a_{il}}$ and broadcasts it together with $h_1, \ldots, h_k$. The client needs to additionally verify the correctness of h_0 together with $h_1, \ldots, h_k$ by batch verification. The server then uses batch verification in Algorithm 2 to check the correctness of $e_t \stackrel{?}{=} y_{i1}^{a_{t1}} \ldots y_{id}^{a_{td}}$ for $t \in \{0, \cdots, k\}$. With the additional commitment e_0, the server can acquire an additional Σ-protocol proof that client $\mathcal{C}_i$ possesses a value γ that satisfies $e_0 = g^{\gamma} h_0^{r_i}$. Once this additional proof is satisfied, the server is sure that client $\mathcal{C}_i$ possesses $\mathbf{u}_i$ that satisfies $y_{il} = g^{u_{il}} w_l^{r_i}$, and that $\gamma = \langle \mathbf{a}_0, \mathbf{u}_i \rangle$. Therefore, the secrets in e_t are indeed the inner product between $\mathbf{a}_t$ and $\mathbf{u}_i$, $t \in \{0, \cdots, k\}$.

Finally, the commitment e_t uses group element h_t, which is different for different t. In order to verify the bound of the sum of the squares of the secrets in e_t, we need to fix another independent group element q and convert e_t to another commitment $o_t = g^{\langle \mathbf{a}_t, \mathbf{u}_i \rangle} q^{s_t}$, where s_t is another blind chosen by client $\mathcal{C}_i$. o_t, $t \in [k]$ are all based on q. We then proceed to check the bound of the squares of the secrets in o_t.

Probabilistic input integrity verification. Now we detail the probabilistic input integrity verification in Algorithm 1. Assume the server and clients generate $k+1$ random samples $\mathbf{A} = (\mathbf{a}_0, \mathbf{a}_1, \cdots, \mathbf{a}_k) \in \mathbb{Z}_p^d$ using the aforementioned techniques. After that, the server computes $h_t = \prod_l w_l^{a_{tl}}$ for $t \in \{0, \cdots, k\}$. Let $\mathbf{h} = (h_0, h_1, \cdots, h_k)$. The server sends $\mathbf{h}$ to the client. Upon receiving the information, the client first verifies the correctness of $\mathbf{h}$ using VerCrt$(\mathbf{w}, \mathbf{h}, \mathbf{A})$ in Algorithm 2. If it is correct, the client computes the following items for generating the proof that Eqn 1 is less than the bound B_0.

- The client computes the inner products between $\mathbf{u}_i$ and each row of $\mathbf{A}$, obtaining $\mathbf{v}^* = (v_0, v_1, \cdots, v_k)$, where $v_t = \langle \mathbf{a}_t, \mathbf{u}_i \rangle$ for $t \in \{0, \cdots, k\}$. The client commits $e_t = g^{v_t} h_t'^{r_i}$ using its secret r_i for $t \in \{0, \cdots, k\}$. Let $\mathbf{e}^* = (e_0, e_1, \cdots, e_k)$ and $\mathbf{e} = (e_1, \cdots, e_k)$. The commitment e_0 is used for integrity check of $\mathbf{y}_i$. The commitments $\mathbf{e}$ are used for bound check of $\mathbf{v} = (v_1, \cdots, v_k)$.
- The client commits v_t using $o_t = g^{v_t} q^{s_t}$ for $t \in [k]$, where s_t is a random number. Let $\mathbf{o} = (o_1, \cdots, o_k)$ be the resulted commitment. Note that e_t and o_t commit to the same secret v_t using different group elements h_t and q.
- The client generates a proof ρ to prove that $(z, \mathbf{e}^*, \mathbf{o})$ is well-formed, which means that the secret in z is used as the blind in e_t, $t \in \{0, \cdots, k\}$, and that the secrets in e_t and o_t are equal, $t \in [k]$. Note that $z = g^{r_i} = \Psi_{r_i}(0)$ is the 0-th coordinate of the check string of Shamir's share of r_i.
- The client generates a proof σ that the secret in o_t is in the interval $[-2^{b_{\text{ip}}}, 2^{b_{\text{ip}}})$ for $t \in [k]$. This ensures the inner product of $\mathbf{a}_t$ and $\mathbf{u}_i$ does not cause overflow when squared.
- The client commits $o_t' = g^{v_t^2} q^{s_t'}$ for $t \in [k]$, where s_t' is a random number. Let $\mathbf{o}' = (o_1', \cdots, o_k')$ be the resulted commitment. This commitment will be used in the proof generation and verification for proof of square.
- The client generates a proof τ to prove that the secret in o_t' is the square of the secret in o_t for $t \in [k]$, using the building block described in Section 2.
- The client generates a proof μ that $B_0 - \sum_t v_t^2$ is in the interval $[0, 2^{b_{\max}})$. This proof is to guarantee that Eqn 1 is less than the bound of the probabilistic check.

Finally, the client sends the proof $\pi = (\mathbf{e}^*, \mathbf{o}, \mathbf{o}', \rho, \tau, \sigma, \mu)$ to the server for verification.

After receiving the proof, the server verifies it accordingly, including checking the correctness of $\mathbf{e}^*$ using Algorithm 2, checking the well-formedness proof ρ, checking the square proofs of $(\mathbf{o}', \mathbf{o})$, and checking the two bound proofs. If all the checks are passed, the server guarantees that the client's update passes the probabilistic check in Algorithm 1. Consequently, the server can verify the proof of each client $\mathcal{C}_i$ and put it to the malicious client list $\mathcal{C}^*$ if the verification fails. The list $\mathcal{C}^*$ is broadcast to all the clients.

4.5 Secure Aggregation

Let $\mathcal{C}_H = \mathcal{C} \setminus \mathcal{C}^*$ be the set of honest clients. In this round, each client $\mathcal{C}_i \in \mathcal{C}_H$ selects the corresponding secret shares from the honest clients $\mathcal{C}_j \in \mathcal{C}_H$, aggregates the shares $r_i' = \sum_{\mathcal{C}_j \in \mathcal{C}_H} r_{ji}$, and sends r_i' to the server. The server uses SS.Verify$(\prod_{j \in \mathcal{C}_H} \Psi_{r_j}, i, r_i', n, m+1, g)$ to verify the integrity of each r_i', obtains the list of clients $\mathcal{C}_{\text{Valid}} \supseteq \mathcal{C}_H$ that pass the integrity check, and computes $r' \leftarrow$ SS.Recover$(\{(i, r_i') : i \in \mathcal{C}_{\text{Valid}}\})$. According to the homomorphic property of VSSS, $r' = r = \sum_{\mathcal{C}_i \in \mathcal{C}_H} r_i$ is the summation of the honest clients' secrets. The summation will be used for calculating the aggregation of honest clients' model updates $\mathcal{U} = \sum_{\mathcal{C}_i \in \mathcal{C}_H} \mathbf{u}_i$.

Specifically, the server can multiply the commitments from honest clients $\mathcal{C}_j \in \mathcal{C}_H$ and obtain:

$$C(\mathcal{U}, r) = (\textstyle\prod_{\mathcal{C}_i \in \mathcal{C}_H} C(u_{i1}, r_i), \ldots, \prod_{\mathcal{C}_i \in \mathcal{C}_H} C(u_{id}, r_i)). \quad (5)$$

[2] To check whether $x_1 = \cdots = x_n = 1$, it is sufficient to randomly sample $\alpha_1, \ldots, \alpha_n$ from the uniform distribution on $\mathbb{Z}_p$ and check whether $x_1^{\alpha_1} \ldots x_n^{\alpha_n} = 1$.

Table 1: Cost comparison (g.e. = group exponentiation, f.a. = field arithmetic, b = bit length)

		EIFFeL		RoFL	ACORN	RiseFL	
		g.e.	f.a.			g.e.	f.a.
Client Comp.	commit.	$O(md)$	$O(nmd)$	$O(d)$	$O(d)$	$O(d)$	small
	proof gen.	0	$O(bnmd)$	$O(db)$	$O(d)$	$O(d/\log d)$	$O(kd)$
	proof ver.	$O(nmd/\log(md))$	$O(bnmd)$	0	0	0	0
	total	$O(nmd/\log(md))$	$O(bnmd)$	$O(db)$	$O(d)$	$O(d)$	$O(kd)$
Server Comp.	prep.	0	0	0	0	$O(kd\log M/\log d\log p)$	small
	proof ver.	0	small	$O(ndb/\log(db))$	$O(nd/\log(d))$	$O(nd/\log d)$	small
	agg.	0	$O(nmd)$	$O(nd/\log p)$	$O(nd/\log(p))$	$O(nd/\log p)$	small
	total	0	$O(nmd)$	$O(ndb/\log(db))$	$O(nd/\log(d))$	$O(d(n+k\log M/\log p)/\log d)$	small
Comm. Per Client		$\approx 2dnb$		$\approx 12d$	$\approx (b+\log n)/\log(p)$	$\approx d$	

Without loss of generality, we consider the l-th ($l \in [d]$) dimension and derive the calculation as follows.

$$\prod_{\mathcal{C}_i \in \mathcal{C}_H} C(u_{il}, r_i) = g^{\sum_{\mathcal{C}_i \in \mathcal{C}_H} u_{il}} w_l^{\sum_{\mathcal{C}_i \in \mathcal{C}_H} r_i} \tag{6}$$

$$= g^{u_l} w_l^{r}, \tag{7}$$

where u_l is the aggregation of the l-th elements in honest clients' updates, w_l is the common group element used for the commitment in Eqn 2, and r is the summation of honest clients' secrets. Note that the derivation of Eqn 6 is due to clients using the same group elements g and w_l. Since the server has already calculated $r = r'$, it can thus compute g^{u_l} for $l \in [d]$ and solve u_l according to Eqn 7.

4.6 Extensions and Discussions

Although we focus on the L_2-norm bound check in this paper, our approach can be extended to support a wide range of defense methods. For instance, it can support sphere defense [60], cosine similarity defense [3, 15], and Zeno++ defense [72], which are designed based on variants of L_2-norm bound check. Moreover, it can be extended to L_∞-norm bound check, i.e., $||\mathbf{u}||_\infty = \max(|u_1|, |u_2|, \cdots, |u_d|)$, which is to ensure that the maximum absolute value of the vector's coordinate is less than a threshold. We shall discuss the extensions as follows.

For sphere defense, the server broadcasts a public vector $\mathbf{v}$ and a bound B, and then checks whether the model update $\mathbf{u}$ satisfies $||\mathbf{u} - \mathbf{v}||_2 \le B$. We can change our protocol in which the i-th client commits to $\mathbf{u}_i - \mathbf{v}$ instead of $\mathbf{u}_i$. The server then recovers $\sum_{i \in \mathcal{C}_H}(\mathbf{u}_i - \mathbf{v})$ and computes $\sum_{i \in \mathcal{C}_H} \mathbf{u}_i = \sum_{i \in \mathcal{C}_H}(\mathbf{u}_i - \mathbf{v}) + \mathbf{v} \cdot |\mathcal{C}_H|$, where $\mathcal{C}_H$ is the set of honest clients. For cosine similarity defense, the server broadcasts $\mathbf{v}$, B and a public hyperparameter α, and then checks if the model update $\mathbf{u}$ satisfies both $||\mathbf{u}||_2 \le B$ and $\langle \mathbf{u}, \mathbf{v} \rangle \ge \alpha ||\mathbf{u}||_2 ||\mathbf{v}||_2$. We can add a predicate to the protocol that checks $||\mathbf{u}||_2 \le \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\alpha ||\mathbf{v}||_2}$ based on Algorithm 1. For the Zeno++ defense $\gamma \langle \mathbf{v}, \mathbf{u} \rangle - \rho ||\mathbf{u}||_2^2 \ge \gamma\epsilon$, where ρ, γ, ϵ are public hyperparameters, it can be converted into sphere defense by $||\mathbf{u} - \frac{\gamma}{2\rho}\mathbf{v}||_2 \le \sqrt{\frac{\gamma}{\rho}\epsilon + \frac{\gamma^2}{4\rho^2}||\mathbf{v}||_2^2}$ for the check.

For L_∞-norm bound check [41], we can adapt our approach to using the idea of approximate proofs of L_∞ bound [27]. To verify that $||\mathbf{u}||_\infty \le B$, the client first commits $\mathbf{u}$ with the hybrid commitment scheme introduced in Section 4.3 and commits an additional random vector $\mathbf{z}$. The server then randomly samples $\mathbf{a}_1, \ldots, \mathbf{a}_k$ from the discrete distribution $P(-1) = P(1) = 1/4$, $P(0) = 1/2$ and broadcasts these vectors to the client. Next, the client proves to the verifier that $|\langle \mathbf{a}_i, \mathbf{u} \rangle + z_i| \le B/\gamma$ for every $i \in [1, k]$. The parameter γ can be computed from k, d and the security parameter using the technique introduced in [27]. If passed, the verifier can guarantee that $||\mathbf{u}||_\infty$ is bounded with overwhelming probability. Note that we can still use the proposed technique in Section 4.4 to generate and verify the proofs of $\langle \mathbf{a}_i, \mathbf{u} \rangle$ with high efficiency.

Norm-bound checks, despite their wide application in defense mechanisms, are not a silver bullet. For example, backdoor attacks on tail targets are still effective even with norm constraints, as shown in [62]. Yet, they are still useful components in newer defenses such as [70], which combine norm-bound checks with parameter smoothing to mitigate the above attack [62]. Besides, although our approach suits various norm-bound defenses, it faces challenges when requiring strict checks on each individual gradient due to the difficulty of dimensionality reduction for ZKP computations.

5 ANALYSIS

5.1 Security Analysis

We present the formal security guarantee of RiseFL in Theorem 2.

Theorem 2. *By choosing $\epsilon = \mathrm{negl}(\kappa)$ and $B_0 = B^2M^2(\sqrt{\gamma_{k,\epsilon}} + \frac{\sqrt{kd}}{2M})^2$, for any list of honest clients $\mathcal{C}_H$ of size at least $n - m$, RiseFL satisfies $(D, F_{k,\epsilon,d,M})$-SAVI, where $D(\mathbf{u}) = ||\mathbf{u}||_2/B$ and*

$$F_{k,\epsilon,d,M}(c) = \Pr_{x \sim \chi_k^2}\left[x < \frac{1}{c^2}\left(\sqrt{\gamma_{k,\epsilon}} + \frac{3\sqrt{kd}}{2M}\right)^2\right] + \mathrm{negl}(\kappa). \tag{8}$$

Proof. We only give a proof sketch due to the space limitation. The proof consists of three parts. First, we show that the server computes an aggregate $\mathcal{U}_{\mathcal{C}_H} = \sum_{i \in \mathcal{C}_{\text{Valid}}} \mathbf{u}_i$ for some $\mathcal{C}_{\text{Valid}} \supseteq \mathcal{C}_H$ with probability at least $1 - \mathrm{negl}(\kappa)$. Note that the aggregation step in Section 4.5 ensures the server computes an aggregate $\mathcal{U}_{\mathcal{C}_H} = \sum_{\mathcal{C}_i \in \mathcal{C}_{\text{Valid}}} \mathbf{u}_i$ for a list $\mathcal{C}_{\text{Valid}}$ that is marked as honest clients. $\mathcal{C}_{\text{Valid}}$ must contain $\mathcal{C}_H$ because given any non-zero $\mathbf{u}$, if $\mathbf{b}_1, \ldots, \mathbf{b}_k$ are sampled i.i.d. from the normal distribution $\mathcal{N}(\mathbf{0}, M \cdot \mathbf{I}_d)$, then $\frac{\langle \mathbf{b}_t, \mathbf{u} \rangle}{||\mathbf{u}||_2 M}$ follows the normal distribution $\mathcal{N}(0, 1)$ for $t \in [k]$ and thus $\frac{1}{||\mathbf{u}||_2^2 M^2} \sum_{t=1}^k \langle \mathbf{b}_t, \mathbf{u} \rangle^2$ follows the Chi-squared distribution χ_k^2.

Table 2: Breakdown cost comparison w.r.t. the number of model parameters d, where $k = 1000$

#Param.	Approach	Client Computation (seconds)				Server Computation (seconds)				Comm. Cost per Client (MB)
		commit.	proof gen.	proof ver.	total	prep.	proof ver.	agg.	total	
d = 1K	EIFFeL	0.865	3.63	11.7	16.2	-	-	0.182	**0.182**	125
	RoFL	0.051	4.43	-	4.5	-	91.2	0.040	91.3	0.37
	ACORN	0.076	2.49	-	2.6	-	58.9	0.040	58.9	**0.004**
	RiseFL (ours)	0.054	1.48	0.08	**1.6**	1.17	75.6	0.071	76.8	0.44
d = 10K	EIFFeL	8.38	36.8	115	161	-	-	1.81	**1.81**	1250
	RoFL	0.51	46.4	-	46.9	-	860	0.41	860	3.66
	ACORN	0.75	24.5	-	25.3	-	522	0.41	523	**0.03**
	RiseFL (ours)	0.49	1.8	0.08	**2.3**	8.61	82.5	0.71	91.8	0.71
d = 100K	EIFFeL	84.7	382	1070	1536	-	-	18.8	**18.8**	12500
	RoFL	5.1	496	-	502	-	8559	4.1	8563	36.6
	ACORN	7.6	253	-	261	-	5087	4.1	5091	**0.3**
	RiseFL (ours)	4.8	4.5	0.08	**9.3**	73.3	139	7.2	219	3.5
d = 1M	EIFFeL	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	RoFL	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	ACORN	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	RiseFL (ours)	48.0	31.2	0.08	**79.3**	653	612	72.1	**1338**	**30.9**

Second, we prove the security of honest clients such that with probability at least $1 - \text{negl}(\kappa)$, nothing else except $\sum_{\mathcal{C}_i \in \mathcal{C}_H} \mathbf{u}_i$ is known for any $\mathcal{C}_i \in \mathcal{C}_H$. Cryptographically, the values of Ψ_{r_i}, $C(\mathbf{u}_i, r_i)$, and π_i do not reveal any information about $\mathbf{u}_i$ or r_i. VSSS ensures that nothing is revealed from the $\leq m$ shares $\{r_{ij}\}_{j \notin \mathcal{C}_H}$ of the secret r_i. For secure aggregation, VSSS ensures that only $\sum_{\mathcal{C}_i \in \mathcal{C}_H} r_i$ is revealed. From $\sum_{\mathcal{C}_i \in \mathcal{C}_H} r_i$, the only value that can be computed is $\sum_{\mathcal{C}_i \in \mathcal{C}_H} \mathbf{u}_i$. Therefore, the claim holds.

Third, we prove that if $\mathcal{C}_i$ is malicious with $||\mathbf{u}||_2/B = c$, the probability that $i \in \mathcal{C}_{\text{Valid}}$ is bounded by $F_{k,\epsilon,d,M}(c)$. Assume that $\mathcal{C}_i$ is malicious and $D(\mathbf{u}_i) = ||\mathbf{u}_i||_2/B = c > 1$. Since $\frac{1}{||\mathbf{u}||_2^2 M^2} \sum_{t=1}^{k} \langle \mathbf{b}_t, \mathbf{u} \rangle^2$ follows χ_k^2, the probability that $\sum_{t=1}^{k} \langle \mathbf{b}_t, \mathbf{u} \rangle^2 \leq B^2 M^2 \gamma_{k,\epsilon}$ is $\Pr_{x \sim \chi_k^2}[x < \frac{\gamma_{k,\epsilon}}{c^2}]$, which can be extended to Eqn 8 after $\mathbf{b}_t$ is discretized. □

5.2 Cost Analysis

Now we analyze the cost of RiseFL under the assumption of $d >> k$, as summarized in Table 1. Here we count the number of cryptographic group exponentiations (g.e.) and finite field arithmetic (f.a.) separately for EIFFeL and RiseFL. The communication cost per client is measured in the number of group elements.

EIFFeL. The commitment includes the Shamir secret shares ($O(nmd)$ f.a.) and the check string for each coordinate ($O(md)$ g.e.). The verification of the check strings of one client takes a multi-exponentiation of length $(m+1)d$, or $O(md/\log md)$ g.e. using a Pippenger-like algorithm [51]. The server cost is small compared to client cost.

RoFL. The dominating cost of proof generation is a proof of bound of each coordinate ($O(bd)$ g.e.). The proof verification is executed by the server, where the verification of the bound proof per client takes 1 multi-exponentiation of length about $2bd$.

ACORN. The ZKP part uses a variant of Bulletproofs by Leveraging Lagrange's four-square theorem. Compared to RoFL, its computational cost does not depend on b. It uses PRG-SecAgg [7], whose communication cost is only $(1 + \log(n)/b)$ times of plaintext.

RiseFL (Ours). On the client side, the main cost of proof generation is to verify the correctness of $\mathbf{h}$ received from the server, using VerCrt. It costs $O(d/\log d)$ g.e. and $O(kd)$ f.a. On the server side, the main cost can be divided into two parts: (1) computing multi-exponentiations $\mathbf{h}$ at the preparation stage (each h_t, $t \in [1, k]$ costs $O(d \log M/\log d \log p)$ g.e. because discrete normal samples have bit length $O(\log M)$); (2) verifying the correctness of $\mathbf{e}$ for each client using VerCrt at the proof verification stage ($O(d/\log d)$ g.e.).

6 EXPERIMENTS

We implement the proposed RiseFL system in C/C++. The cryptographic primitives are based on libsodium [37] which implements the Ristretto group [29] on Curve25519 that supports 126-bit security. The implementation consists of 9K lines of code in C/C++.

6.1 Methodology

Experimental Setup. We conduct the micro-benchmark experiments on a single server equipped with Intel(R) Core(TM) i7-8550U CPU and 16GB of RAM. The experiments of the federated learning tasks are simulated on a server with Intel(R) Xeon(R) W-2133 CPU, 64GB of RAM, and GeForce RTX 2080 Ti. Unless otherwise specified, we set the number of clients to 100 and the maximum number of malicious clients to 10 in the experiments. The default security parameter is 126 bits. We set $\epsilon = 2^{-128}$ to ensure that the level of security of RiseFL matches the baselines. Besides, we set $M = 2^{24}$ to make sure that the rounding error of discrete normal samples is small.

Datasets. We use three real-world datasets, namely, OrganAMNIST [75, 76], OrganSMNIST [75, 76] and Forest Cover Type [10], to run the FL tasks and measure the classification accuracy. The OrganAMNIST and OrganSMNIST datasets are medical image datasets based on abdominal clinical computed tomography. These datasets comprise 58850 and 25221 images of 784 numerical features (28×28) in 11 classes, respectively. The Forest Cover Type dataset is a

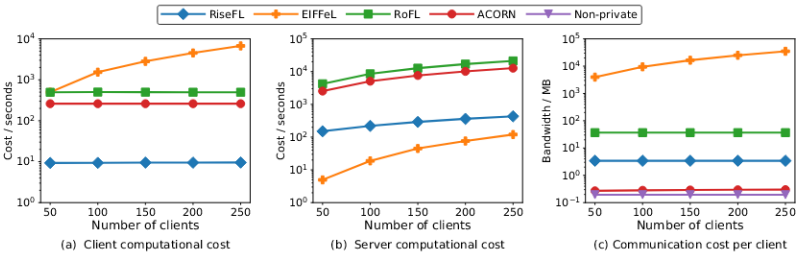


Figure 4: Cost comparison w.r.t. the number of clients.

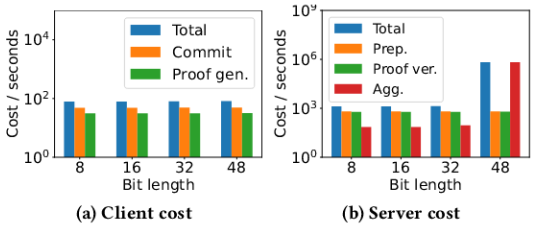


Figure 5: Cost comparison w.r.t. bit length.

tabular dataset consisting of 581012 rows with 7 classes. We also generate a synthetic dataset for micro-benchmarking the computational and communication costs.

Models. A CNN model is exploited for OrganAMNIST, ResNet-18 [31] for OrganSMNIST, and TabNet [2] for Forect Cover Type. The CNN model consists of four layers and 31K parameters. The ResNet-18 model consists of 18 layers and 11M parameters. The TabNet model consists of 471K parameters. For the micro-benchmark experiments on synthetic datasets, we use synthetic models with 1K, 10K, 100K, and 1M parameters for the evaluation.

Baselines. We compare RiseFL with three secure baselines, namely RoFL [41], EIFFeL [55], and ACORN [6], using the same secure parameter, to evaluate the performance. Furthermore, we compare RiseFL with two non-private baselines for integrity check to evaluate model accuracy. The details of the baselines are as follows:

- *RoFL* [41] adopts the ElGammal commitment scheme and uses the strict checking zero-knowledge proof for integrity check. It does not guarantee Byzantine robustness.
- *EIFFeL* [55] employs the verifiable Shamir secret sharing (VSSS) scheme and secret-shared non-interactive proofs (SNIP) for SAVI, ensuring Byzantine robustness. We implement EIFFeL as it is not open-sourced.
- *ACORN* [6] adopts the PRG-SecAgg protocol [7] and uses the strict checking zero-knowledge proof for integrity check. It does not guarantee Byzantine robustness.
- *NP-SC* is a non-private baseline with strict integrity checking, i.e., the server checks each client's update and eliminates the update that is out of the L_2-norm bound.
- *NP-NC* is a non-private baseline without any checking on clients' updates. In this baseline, malicious clients can poison the aggregated models through malformed updates.

Metrics. We utilize three metrics to evaluate the performance of our RiseFL system.

- *Computational Cost* refers to the computation time on each client and on the server. It measures the computational efficiency of the protocol.
- *Communication Cost per Client* refers to the size of messages transmitted between the server and each client. It measures the communication efficiency.
- *Model Accuracy* refers to the ratio of correct predictions for the trained FL model. It measures the effectiveness of the integrity check method in RiseFL.

6.2 Micro-Benchmark Efficiency Evaluation

We first compare the cost of our RiseFL with EIFFeL, RoFL, and ACORN, using micro-benchmark experiments. We use 16 bits for encoding floating-point values and run the experiments on one CPU thread on both the client and server sides.

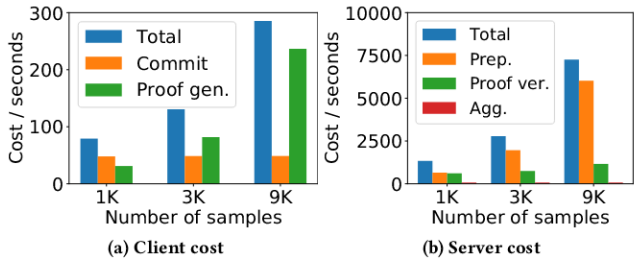
Effects of d. Tables 2 shows the breakdown cost comparison of RiseFL, EIFFeL, RoFL, and ACORN w.r.t. the number of parameters d in the FL model, the client computational cost, the server computational cost, and the communication cost per client. We set the number of samples $k = 1000$. We failed to run the experiments with $d = 1$M on EIFFeL, RoFL, and ACORN due to insufficient RAM, i.e., out of memory (OOM).

From the table, we observe that RiseFL is superior to the baselines. For example, when $d = 100$K, the client cost of RiseFL is 28x smaller than ACORN, 53x smaller than RoFL, and 164x smaller than EIFFeL. Compared to RoFL and ACORN, the savings occur at the proof generation stage, which is attributed to our probabilistic check technique. The cost of EIFFeL is large as every client is responsible for computing a proof digest of every other client's proof in the proof verification stage, which dominates the cost. In comparison, the client-side proof verification cost of RiseFL is negligible since every client $\mathcal{C}_i$ only verifies the check string of the Shamir share of one value r_j from every other client $\mathcal{C}_j$. This is consistent with the theoretical cost analysis in Table 1.

With regard to server cost, RiseFL is 23x faster than ACORN and 39x faster than RoFL when $d = 100$K, due to our probabilistic check method. RiseFL incurs a higher server cost than EIFFeL because in EIFFeL, the load of proof verification is on the client side, and the dominating cost of the server is at the aggregation stage. Nevertheless, the total server cost of RiseFL is 7 times smaller than the cost of EIFFeL of every client.

For communication cost, when $d = 100$K, RiseFL results in 16x more than transmitting the weight updates in clear text: the committed value of a 16-bit weight update is 256-bit long. The proof size is negligible because $k << d$. ACORN transmits only about 2x more than clear text due to its use of PRG-SecAgg. RoFL transmits about 10x more elements than RiseFL in the form of proofs of well-formedness and proofs of squares. The communication cost of EIFFeL is three orders of magnitude larger than RiseFL as it transmits the share of every bit of every coordinate of the weight update to every other client.

Effects of n. Figure 4 shows the comparison of the computational cost and communication cost per client in terms of the number of clients. We vary the number of clients $n \in \{50, 100, 150, 200, 250\}$,



(a) Client cost (b) Server cost

Figure 6: Cost w.r.t. the number of samples (d = 1M).

and set the number of parameters d = 100K, and the maximum number of malicious clients $m = 0.1n$. We set the number of samples k = 1K for RiseFL in this experiment. From Figures 4a and 4c, we observe that both the client computational cost and communication cost per client of RiseFL are at least one order of magnitude lower than RoFL, EIFFeL, and ACORN, while the server cost of RiseFL is linear in n. In addition, the costs of EIFFeL on both the client and server sides increase quadratically with n. When n gets larger, the advantage of RiseFL is even larger compared to EIFFeL.

Effects of k. Figure 6 shows the breakdown of RiseFL with d = 1M by varying $k \in \{1\text{K}, 3\text{K}, 9\text{K}\}$. On the client side, proof generation is the only stage that scales with k. On the server side, as k gets larger, the preparation cost of computing $\mathbf{h}$ becomes dominant. The effects of k on the cost breakdown can be explained by Table 1. Specifically, the terms that scale linearly with k are the $O(kd)$ f.a. term of the client's proof generation and the server's preparation costs. The linear-in-k terms become dominant as k becomes larger.

Effects of bit-length of weight updates. We study the effects of the encoding bit-length of model updates on client and server computational costs. Specifically, we fix the number of parameters d = 1M, the number of samples k = 1K, the number of clients n = 100, and the maximum number of malicious clients m = 10, and vary bit-length in $\{8, 16, 32, 48\}$. Figure 5 shows the experimental results. There are two main observations. First, the effect of the bit-length on the client computational cost is negligible because the dominating costs are the computation of Pedersen commitment and the verification of the combined group elements, which are not sensitive to bit length. Second, the server computation cost is relatively stable as the bit-length increases, except for bit-length 48. The reason is that, when the bit-length is 48, the server's aggregation cost is dominated by solving the discrete log problem $g^w = x$ for a b-bit integer w (see Section 4.5), which is exponential in b using the baby-step giant-step algorithm [58]. Nevertheless, as will be shown in Table 3, bit-length 32 already results in satisfactory FL model accuracy.

6.3 Robustness Evaluation

To evaluate the effectiveness of our probabilistic check method, we present the FL model accuracy of RiseFL against four commonly used attacks on OrganAMNIST, OrganSMNIST, and Forest Cover Type datasets. We use 32-bits to encode model updates. The first is the sign flip attack [17], where each malicious client submits $-c \cdot \mathbf{u}$ as its model update with $c > 1$. The second is the scaling attack [9], where each malicious client submits $c \cdot \mathbf{u}$ as its model update with $c > 1$. The third is the label flip attack [61], which labels one category of samples as another. The fourth is the additive noise attack [34], which adds Gaussian noises to the model update.

Table 3: Accuracy comparison w.r.t. bit-length

bit-length	OrganAMNIST	OrganSMNIST	Forest
8	85.97 ± 0.61	14.92 ± 5.77	41.27 ± 3.61
16	86.61 ± 0.30	61.59 ± 2.57	55.45 ± 2.31
32	86.80 ± 0.51	71.98 ± 1.37	89.96 ± 0.10
48	87.13 ± 0.80	71.83 ± 2.14	89.87 ± 0.09

We experiment with three different defenses: the L_2 norm defense, the sphere defense [60], and the cosine similarity defense [3, 15]. In this set of experiments, we use FLSim [54] to simulate federated learning with 100 clients and 10 malicious clients. Figure 7 shows the training curves of RiseFL compared to two non-private baselines, NP-SC and NP-NC (see Section 6.1). There are two main observations. First, RiseFL achieves better accuracy than the no-checking baseline NP-NC. This is expected as the malicious clients can poison the aggregated models by invaliding model updates if the server does not check the input integrity, leading to lower accuracy or non-converging curves. Second, the training curves of RiseFL and the strict integrity check baseline NP-SC (e.g., L_2-norm check in Figure 7j, cosine similarity check in Figure 7k, and sphere defense in Figure 7h) are very close. This confirms the effectiveness of RiseFL in identifying malformed updates and robust aggregation.

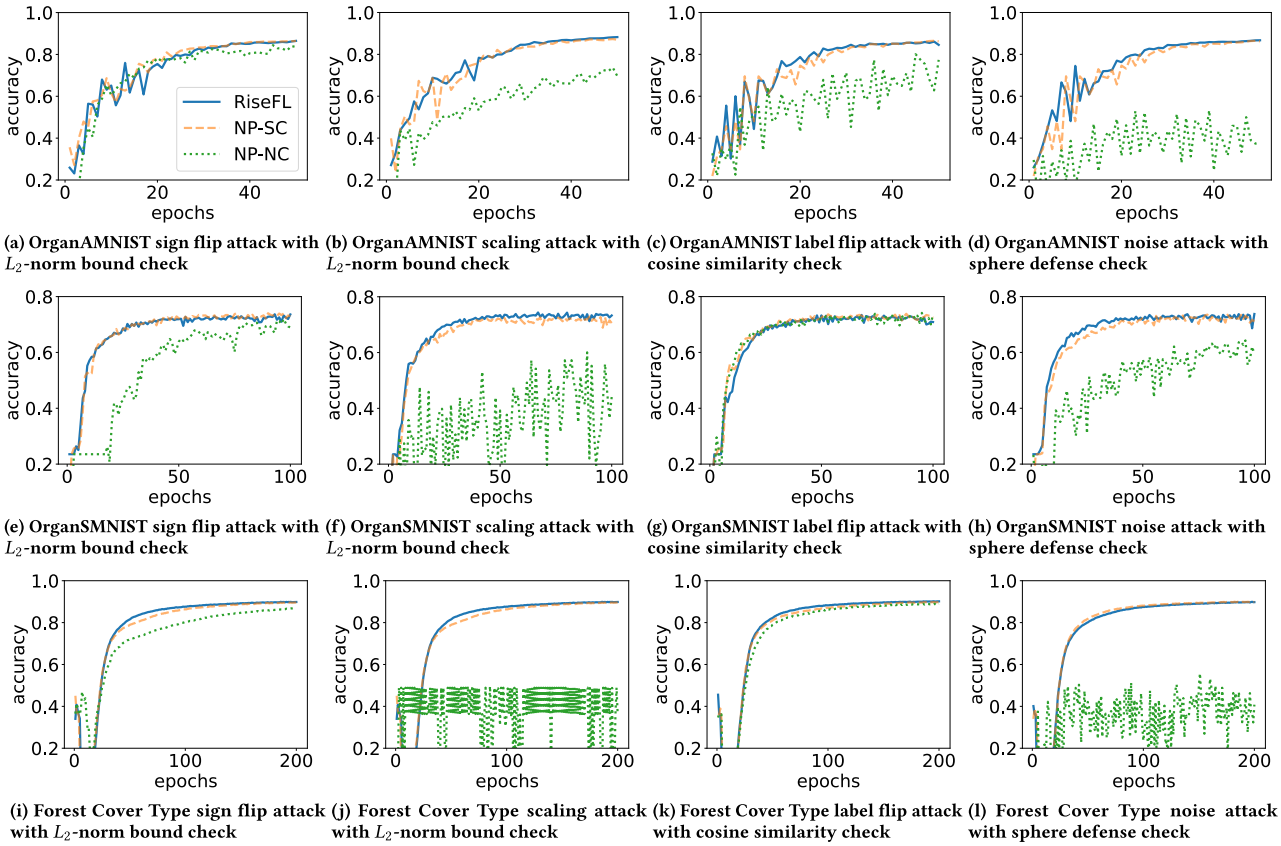
We further study the impact of the bit-length used for encoding weight updates on the FL model accuracies under the scaling attack. Table 3 shows the experimental results w.r.t. mean and standard deviation. We observe that the accuracies increase significantly when the bit-length is increased from 8 to 32. This is expected because with a larger bit-length, the precision loss for commitments and ZKP computations is smaller, resulting in higher accuracy. However, when the bit-length is 48, the accuracy gain is minor compared to the bit-length of 32.

7 RELATED WORKS

Secure Aggregation. To protect the client's input privacy in FL, a number of studies have explored secure aggregation [12, 32, 81], which enables the server to compute the aggregation of clients' model updates without knowing individual updates. A widely used approach [12] is to let each client use pairwise random values to mask the local update before uploading it to the server. The server can then securely cancel out the masks for correct aggregation. Nonetheless, these solutions do not guarantee input integrity, as malicious clients can submit arbitrary masked updates.

Robust Learning. Several works have been proposed for robust machine learning, including [15, 42, 48, 69, 74, 78, 79], which ensures Byzantine resilient gradient aggregation. These solutions operate by identifying and eliminating client updates that deviate significantly from the majority of clients' updates. However, these approaches require the server to access the plaintext model updates, which compromises the client's input privacy.

Input Integrity Check with Secure Aggregation. In this research direction, Prio [16] and Elsa [53] ensure both input privacy and integrity with multiple non-colluding servers. Under the



(a) OrganAMNIST sign flip attack with L_2-norm bound check (b) OrganAMNIST scaling attack with L_2-norm bound check (c) OrganAMNIST label flip attack with cosine similarity check (d) OrganAMNIST noise attack with sphere defense check

(e) OrganSMNIST sign flip attack with L_2-norm bound check (f) OrganSMNIST scaling attack with L_2-norm bound check (g) OrganSMNIST label flip attack with cosine similarity check (h) OrganSMNIST noise attack with sphere defense check

(i) Forest Cover Type sign flip attack with L_2-norm bound check (j) Forest Cover Type scaling attack with L_2-norm bound check (k) Forest Cover Type label flip attack with cosine similarity check (l) Forest Cover Type noise attack with sphere defense check

Figure 7: The comparison of training curves between RiseFL and non-private baselines.

stronger assumption of at least two non-colluding servers, secure aggregation can be achieved with a lower computation cost. In contrast, we focus on a single-server setting. Under this setting, [6, 41, 55] ensure both input privacy and integrity. RoFL [41] utilizes ElGamal commitments [25] and adopts a discrete-log based zero-knowledge proof system [13]. ACORN [6] presents a generalized secure aggregation protocol of PRG-SecAgg [7] and constructs ZKP protocols based on Schnorr proof of knowledge [56] and Bulletproofs. However, they do not support Byzantine-robust aggregation. EIFFeL [55] uses the verifiable secret sharing [57] and secret-shared non-interactive proofs (SNIP) [16] techniques to tolerate Byzantine attacks. Nevertheless, the efficiency of these three solutions is quite low, especially when the number of model parameters is large. In contrast, in RiseFL, we propose a probabilistic integrity check method with a hybrid commitment scheme, supporting Byzantine-robust aggregation and achieving high efficiency.

Zero-Knowledge Proof (ZKP). ZKP allows a prover to convince a verifier that a certain statement is true without revealing extra information. Efforts in recent years have significantly reduced the overhead of ZKP protocols, such as zkSNARKs [8, 26, 49], EMP-ZK [65, 66, 77], and ZKSQL [36]. However, these ZKP protocols are designed for single provers and lack additive homomorphism. In our scenario, there are multiple provers (i.e., the parties), with each holding a unique private key for its commitment; the verifier (i.e., the server) needs to not only verify the correctness of each prover's proof but also aggregate valid proofs, which cannot be supported by the above ZKP protocols. We design a hybrid commitment scheme and the corresponding ZKP protocol to efficiently aggregate proofs from multiple provers.

8 CONCLUSIONS

In this paper, we propose RiseFL, a secure and verifiable data collaboration system that guarantees both input privacy and input integrity of the participating clients. We present a probabilistic input integrity check method and design a hybrid commitment scheme based on Pedersen commitment and verifiable Shamir secret sharing, which achieves a comparable security guarantee to state-of-the-art solutions while significantly reducing the computation and communication costs. The experimental results confirm the efficiency and effectiveness of our solution.

ACKNOWLEDGMENTS

This work is supported by the National Research Foundation, Singapore under its Emerging Areas Research Projects (EARP) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

REFERENCES

[1] 2016. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). (2016).

[2] Sercan Ö Arik and Tomas Pfister. 2021. Tabnet: Attentive interpretable tabular learning. In *AAAI*, Vol. 35. 6679–6687.

[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How To Backdoor Federated Learning. In *AISTATS*. 2938–2948.

[4] Ergute Bao, Yizheng Zhu, Xiaokui Xiao, Yin Yang, Beng Chin Ooi, Benjamin Hong Meng Tan, and Khin Mi Mi Aung. 2022. Skellam Mixture Mechanism: a Novel Approach to Federated Learning with Differential Privacy. *Proc. VLDB Endow.* 15, 11 (2022), 2348–2360.

[5] Sebastian Baunsgaard, Matthias Boehm, Ankit Chaudhary, Behrouz Derakhshan, Stefan Geißelsöder, Philipp M. Grulich, Michael Hildebrand, Kevin Innerebner, Volker Markl, Claus Neubauer, Sarah Osterburg, Olga Ovcharenko, Sergey Redyuk, Tobias Rieger, Alireza Rezaei Mahdiraji, Sebastian Benjamin Wrede, and Steffen Zeuch. 2021. ExDRa: Exploratory Data Science on Federated Raw Data. In *SIGMOD*. ACM, 2450–2463.

[6] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. 2023. {ACORN}: Input Validation for Secure Aggregation. In *USENIX Security 23*. 4805–4822.

[7] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. In *CCS*. 1253–1269.

[8] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *CRYPTO*. Springer, 90–108.

[9] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin B. Calo. 2019. Analyzing Federated Learning through an Adversarial Lens. In *ICML*. 634–643.

[10] Jock Blackard. 1998. Covertype. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C50K5N.

[11] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract). In *STOC*. 103–112.

[12] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *CCS*. 1175–1191.

[13] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *S&P*. IEEE, 315–334.

[14] Jan Camenisch and Markus Stadler. 1997. Proof systems for general statements about discrete logarithms. *Technical Report/ETH Zurich, Department of Computer Science* 260 (1997).

[15] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. 2021. FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. In *NDSS*.

[16] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics.. In *NSDI*. 259–282.

[17] Georgios Damaskinos, Rachid Guerraoui, Rhicheek Patra, Mahsa Taziki, et al. 2018. Asynchronous Byzantine machine learning (the case of SGD). In *ICML*. PMLR, 1145–1154.

[18] Ivan Damgård and Mads Jurik. 2001. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *Public Key Cryptography*. 119–136.

[19] Zhenan Fan, Huang Fang, Zirui Zhou, Jian Pei, Michael P. Friedlander, Changxin Liu, and Yong Zhang. 2022. Improving Fairness for Data Valuation in Horizontal Federated Learning. In *ICDE*. 2440–2453.

[20] Paul Feldman. 1987. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*. IEEE, 427–438.

[21] Fangcheng Fu, Yingxia Shao, Lele Yu, Jiawei Jiang, Huanran Xue, Yangyu Tao, and Bin Cui. 2021. VF^{2}Boost: Very Fast Vertical Federated Gradient Boosting for Cross-Enterprise Learning. In *SIGMOD*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 563–576.

[22] Fangcheng Fu, Huanran Xue, Yong Cheng, Yangyu Tao, and Bin Cui. 2022. BlindFL: Vertical Federated Machine Learning without Peeking into Your Data. In *SIGMOD*. 1316–1330.

[23] Rui Fu, Yuncheng Wu, Quanqing Xu, and Meihui Zhang. 2023. FEAST: A Communication-efficient Federated Feature Selection Framework for Relational Data. *Proc. ACM Manag. Data* 1, 1 (2023), 107:1–107:28.

[24] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. 2018. Mitigating Sybils in Federated Learning Poisoning. *CoRR* abs/1808.04866 (2018).

[25] Taher El Gamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory* 31, 4 (1985), 469–472.

[26] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT*. Springer, 626–645.

[27] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. 2022. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *EUROCRYPT*. Springer, 458–487.

[28] Oded Goldreich. 2001. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press.

[29] The Ristretto Group. [n.d.]. https://ristretto.group/.

[30] Jamie Hayes and Olga Ohrimenko. 2018. Contamination Attacks and Mitigation in Multi-Party Machine Learning. In *NeurIPS*. 6604–6616.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.

[32] Peter Kairouz, Ziyu Liu, and Thomas Steinke. 2021. The Distributed Discrete Gaussian Mechanism for Federated Learning with Secure Aggregation. In *ICML*. 5201–5212.

[33] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *CCS*. 1575–1590.

[34] Liping Li, Wei Xu, Tianyi Chen, Georgios B Giannakis, and Qing Ling. 2019. RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *AAAI*, Vol. 33. 1544–1551.

[35] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. 2022. Federated Learning on Non-IID Data Silos: An Experimental Study. In *ICDE*. 965–978.

[36] Xiling Li, Chenkai Weng, Yongxin Xu, Xiao Wang, and Jennie Rogers. 2023. ZKSQL: Verifiable and Efficient Query Evaluation with Zero-Knowledge Proofs. *Proc. VLDB Endow.* 16, 8 (2023), 1804–1816.

[37] Libsodium. [n.d.]. https://doc.libsodium.org/.

[38] Junxu Liu, Jian Lou, Li Xiong, Jinfei Liu, and Xiaofeng Meng. 2021. Projected Federated Averaging with Heterogeneous Differential Privacy. *PVLDB* 15, 4 (2021), 828–840.

[39] Yejia Liu, Weiyuan Wu, Lampros Flokas, Jiannan Wang, and Eugene Wu. 2021. Enabling SQL-based Training Data Debugging for Federated Learning. *Proc. VLDB Endow.* 15, 3 (2021), 388–400.

[40] Zhaojing Luo, Shaofeng Cai, Yatong Wang, and Beng Chin Ooi. 2023. Regularized Pairwise Relationship based Analytics for Structured Data. *Proc. ACM Manag. Data* 1, 1 (2023), 82:1–82:27. https://doi.org/10.1145/3588936

[41] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. 2023. Rofl: Robustness of secure federated learning. In *S&P*. IEEE, 453–476.

[42] Xu Ma, Xiaoqian Sun, Yuduo Wu, Zheli Liu, Xiaofeng Chen, and Changyu Dong. 2022. Differentially Private Byzantine-Robust Federated Learning. *IEEE Trans. Parallel Distributed Syst.* 33, 12 (2022), 3690–3701.

[43] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*. 1273–1282.

[44] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting Unintended Feature Leakage in Collaborative Learning. In *S&P*. 691–706.

[45] Ralph C. Merkle. 1978. Secure Communications Over Insecure Channels. *Commun. ACM* 21, 4 (1978), 294–299.

[46] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In *S&P*. 739–753.

[47] Beng Chin Ooi, Kian-Lee Tan, Sheng Wang, Wei Wang, Qingchao Cai, Gang Chen, Jinyang Gao, Zhaojing Luo, Anthony K. H. Tung, Yuan Wang, Zhongle Xie, Meihui Zhang, and Kaiping Zheng. 2015. SINGA: A Distributed Deep Learning Platform. In *ACM MM*. ACM, 685–688.

[48] Xudong Pan, Mi Zhang, Duocai Wu, Qifan Xiao, Shouling Ji, and Min Yang. 2020. Justinian's GAAvernor: Robust Distributed Learning with Gradient Aggregation Agent. In *USENIX Security*. 1641–1658.

[49] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2016. Pinocchio: Nearly practical verifiable computation. *Commun. ACM* 59, 2 (2016), 103–112.

[50] Torben Pryds Pedersen. 2001. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*. Springer, 129–140.

[51] Nicholas Pippenger. 1980. On the evaluation of powers and monomials. *SIAM J. Comput.* 9, 2 (1980), 230–250.

[52] C. Pomerance and S. Goldwasser. 1990. *Cryptology and Computational Number Theory*. American Mathematical Society. https://books.google.com.sg/books?id=yyfS7MKQhJUC

[53] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. 2023. Elsa: Secure aggregation for federated learning with malicious actors. In *S&P*. IEEE, 1961–1979.

[54] Facebook Research. [n.d.]. FLSim: https://github.com/facebookresearch/FLSim.

[55] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. 2022. EIFFeL: Ensuring Integrity for Federated Learning. In *CCS*. 2535–2549.

[56] Claus-Peter Schnorr. 1989. Efficient Identification and Signatures for Smart Cards. In *CRYPTO*, Gilles Brassard (Ed.), Vol. 435. 239–252.

[57] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[58] Daniel Shanks. 1971. Class number, a theory of factorization, and genera. In *Proc. Symp. Math. Soc., 1971*, Vol. 20. 415–440.

[59] Virat Shejwalkar, Amir Houmansadr, Peter Kairouz, and Daniel Ramage. 2022. Back to the Drawing Board: A Critical Evaluation of Poisoning Attacks on Production Federated Learning. In *IEEE S&P*. IEEE, 1354–1371.

[60] Jacob Steinhardt, Pang Wei Koh, and Percy Liang. 2017. Certified Defenses for Data Poisoning Attacks. In *NeurIPS*. 3517–3529.

[61] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. 2019. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963* (2019).

[62] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris S. Papailiopoulos. 2020. Attack of the Tails: Yes, You Really Can Backdoor Federated Learning. In *NeurIPS*.

[63] Yansheng Wang, Yongxin Tong, Zimu Zhou, Ruisheng Zhang, Sinno Jialin Pan, Lixin Fan, and Qiang Yang. 2023. Distribution-Regularized Federated Learning on Non-IID Data. In *ICDE*. IEEE, 2113–2125.

[64] Yatong Wang, Yuncheng Wu, Xincheng Chen, Gang Feng, and Beng Chin Ooi. 2023. Incentive-Aware Decentralized Data Collaboration. *Proc. ACM Manag. Data* 1, 2 (2023), 158:1–158:27.

[65] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2021. Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *S&P*. IEEE, 1074–1091.

[66] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning. In *USENIX Security*. 501–518.

[67] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. 2020. Privacy Preserving Vertical Federated Learning for Tree-based Models. *Proc. VLDB Endow.* 13, 11 (2020), 2090–2103.

[68] Yuncheng Wu, Naili Xing, Gang Chen, Tien Tuan Anh Dinh, Zhaojing Luo, Beng Chin Ooi, Xiaokui Xiao, and Meihui Zhang. 2023. Falcon: A Privacy-Preserving and Interpretable Vertical Federated Learning System. *Proc. VLDB Endow.* 16, 10 (2023), 2471–2484.

[69] Zihang Xiang, Tianhao Wang, Wanyu Lin, and Di Wang. 2023. Practical Differentially Private and Byzantine-resilient Federated Learning. *Proc. ACM Manag. Data* 1, 2 (2023), 119:1–119:26.

[70] Chulin Xie, Minghao Chen, Pin-Yu Chen, and Bo Li. 2021. CRFL: Certifiably Robust Federated Learning against Backdoor Attacks. In *ICML*, Vol. 139. PMLR, 11372–11382.

[71] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. 2020. DBA: Distributed Backdoor Attacks against Federated Learning. In *ICLR*.

[72] Cong Xie, O Koyejo, and I Gupta. 2019. Zeno++: robust asynchronous SGD with arbitrary number of Byzantine workers. *arXiv preprint arXiv:1903.07020* (2019).

[73] Yuexiang Xie, Zhen Wang, Dawei Gao, Daoyuan Chen, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. 2023. FederatedScope: A Flexible Federated Learning Platform for Heterogeneity. *Proc. VLDB Endow.* 16, 5 (2023), 1059–1072.

[74] Chang Xu, Yu Jia, Liehuang Zhu, Chuan Zhang, Guoxie Jin, and Kashif Sharif. 2022. TDFL: Truth Discovery Based Byzantine Robust Federated Learning. *IEEE Trans. Parallel Distributed Syst.* 33, 12 (2022), 4835–4848.

[75] Jiancheng Yang, Rui Shi, and Bingbing Ni. 2021. MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis. In *ISBI*. 191–195.

[76] Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. 2023. MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification. *Scientific Data* 10, 1 (2023), 41.

[77] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *CCS*. 2986–3001.

[78] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter L. Bartlett. 2018. Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates. In *ICML*. 5636–5645.

[79] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter L. Bartlett. 2019. Defending Against Saddle Point Attack in Byzantine-Robust Distributed Learning. In *ICML*. 7074–7084.

[80] Zhihao Zeng, Yuntao Du, Ziquan Fang, Lu Chen, Shiliang Pu, Guodong Chen, Hui Wang, and Yunjun Gao. 2023. FLBooster: A Unified and Efficient Platform for Federated Learning Acceleration. In *ICDE*. IEEE, 3140–3153.

[81] Yifeng Zheng, Shangqi Lai, Yi Liu, Xingliang Yuan, Xun Yi, and Cong Wang. 2023. Aggregation Service for Federated Learning: An Efficient, Secure, and More Resilient Realization. *IEEE Trans. Dependable Secur. Comput.* 20, 2 (2023), 988–1001.