BPFL: TOWARDS EFFICIENT BYZANTINE-ROBUST AND PROVABLY PRIVACY-PRESERVING FEDERATED LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Federated learning (FL) is an emerging distributed learning paradigm without sharing participating clients' private data. However, existing works show that FL is vulnerable to both Byzantine (security) attacks and data reconstruction (privacy) attacks. Existing FL defenses only address one of the two attacks, and also face the efficiency issue. We propose BPFL, an efficient Byzantine-robust and provably privacy-preserving FL method that addresses all the issues. Specifically, we draw on the state-of-the-art Byzantine-robust FL method and use similarity metrics to measure the robustness of each participating client in FL. The validity of clients are formulated as circuit constraints on similarity metrics and verified via a zero-knowledge proof. Moreover, the client models are masked by a shared random vector, which is generated based on homomorphic encryption. In doing so, the server receives the masked client models rather than the true ones, which are proven to be private. BPFL is also efficient due to the usage of non-interactive zero-knowledge proof. Experimental results on various datasets show that our BPFL is efficient, Byzantine-robust, and privacy-preserving.

1 INTRODUCTION

Federated learning (FL) (McMahan et al., 2017), an emerging distributed learning paradigm, enables multiple clients to collaboratively train a model with the coordination of a server, where the client data are kept locally and do not share with any other clients/server during the entire course of learning. As data from each participating client does not need to be shared, the FL provides a baseline level of privacy. Many applications, such as Google's Gboard (Bonawitz et al., 2019), healthcare informatics (Xu et al., 2021), and credit risk prediction (Yang et al., 2019), have shown the potential of FL in the real-world. However, recent works have shown that the current FL design faces both security and privacy issues.

On one hand, FL is vulnerable to Byzantine attacks (also called model poisoning attacks) (Bhagoji et al., 2019; Fang et al., 2020; Baruch et al., 2019; Shejwalkar & Houmansadr, 2021; Bagdasaryan et al., 2020; Xie et al., 2019a), where a few malicious clients can significantly reduce the overall performance by injecting carefully designed poisoned local models during FL training. To address this issue, several Byzantine-robust FL methods have been proposed (Blanchard et al., 2017; Chen et al., 2017; Guerraoui et al., 2018; Yin et al., 2018; Guerraoui et al., 2018; Chen et al., 2018; Pillutla et al., 2019; Xie et al., 2019b; Wu et al., 2020; Cao et al., 2021). Though with different techniques, the main idea of these Byzantine-robust schemes is that the server performs statistical analysis on client models and uncovers malicious client models as those largely deviate from others based on some similarities metrics. For instance, in the state-of-the-art FLTrust (Cao et al., 2021), the server holds a clean validation dataset and uses it to train a reference model. The server then assigns a trust score to each client model based on the cosine similarity between the client model and the reference model. A client model with a relatively small trust score will be flagged as malicious.

All the existing Byzantine-robust FL methods are based on *plaintext* (i.e., shared client model gradients or parameters). However, many existing works (Hitaj et al., 2017; Zhu et al., 2019; Wang et al., 2019; Geiping et al., 2020; Yin et al., 2021; Jeon et al., 2021; Luo et al., 2021; Balunovic et al., 2022; Fowl et al., 2022; Sun et al., 2021) show that FL is vulnerable to privacy attacks, particularly

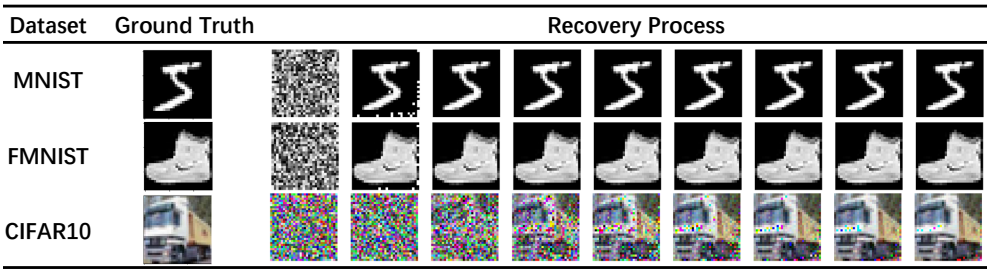


Figure 1: An honest-but-curious server recovers the raw data from the shared client models trained by FLTrust.

data reconstruction attacks where an honest-but-curious server can successfully recover the private client data from the shared client models. We note that the Byzantine-robust FL methods face the same issue. For instance, we successfully recover the raw training data from the shared client models trained by the-state-of-the-art FLTrust (Cao et al., 2021) using the DLG attack proposed in Zhu et al. (2019) (See Figure 1). To mitigate privacy attacks, various provably privacy-preserving FL mechanisms have been designed, and these methods are mainly based on three techniques: differential privacy (DP) (Pathak et al., 2010; Shokri & Shmatikov, 2015; Hamm et al., 2016; McMahan et al., 2018; Geyer et al., 2017; Wei et al., 2020), secure multi-party computation (MPC) (Danner & Jelasity, 2015; Mohassel & Zhang, 2017; Bonawitz et al., 2017; Melis et al., 2019), and homomorphic encryption (HE) (Aono et al., 2017; Zhang et al., 2020). However, these methods either have high utility losses (e.g., DP-based), or induce huge communication and computation overheads (e.g., MPC-based). Moreover, all of them cannot defend against the Byzantine attacks. More details about existing works are shown in Appendix A.

We advocate that a practical FL system should maintain the privacy of the participating clients' data, ensure the robustness against malicious clients during the entire learning, and be computation and communication efficient. However, as discussed above, all the current FL works only satisfy part of these requirements. We aim to design a novel FL that achieves the following goals simultaneously:

1. **Byzantine-robust**: The server can detect invalid and malicious local models submitted by clients and refuse them to participate in global model aggregation.
2. **Privacy-preserving**: The server cannot infer clients' private data during the entire FL training.
3. **Efficient**: Our method should not incur too many computation and communication overheads, compared to the standard FL methods.

Specifically, we propose an efficient Byzantine-robust and privacy-preserving FL method termed **BPFL**. To ensure Byzantine-robust, we draw ideas from existing Byzantine-robust methods such as Krum (Blanchard et al., 2017) and FLTrust (Cao et al., 2021), where the server in BPFL also holds a clean validation dataset to train a reference model and uses similarity metrics to measure the maliciousness/robustness of client models. Unlike existing methods, we use the Zero-Knowledge Proof (ZKP) (Goldwasser et al., 1989) to verify the validity of client models. Particularly, we use both cosine similarity and Euclidean distance as circuit constraints in the ZKP to verify the validity of the client model, and the proofs are generated by the clients and verified by the server. The client model that fails to validate will be rejected to participate in the aggregation. To guarantee privacy-preserving, the client model submitted to the server will be masked by a random vector; and we design a mask vector negotiation protocol (MVNP), based on HE, to generate a shared mask for all clients without sharing each client's data to others. The server then receives the masked client models rather than the true ones and we prove that the server cannot infer any useful data information from the masked client models. Finally, due to the non-interactive properties of ZKP and the efficiency to generate circuit constraints, BPFL is also communication and computation efficient. In summary, our contributions are as follows:

- To the best of our knowledge, BPFL is the first FL design that is Byzantine-robust, provably privacy-preserving, and communication and computation efficient as well.
- BPFL seamlessly integrates the ideas of existing Byzantine-robust FL methods, zero-knowledge proof, and homomorphic encryption into a unified framework.
- Evaluations on synthetic and real-world datasets show BPFL can defend against both Byzantine attacks and data reconstruction attacks, with small computation and communication overheads.

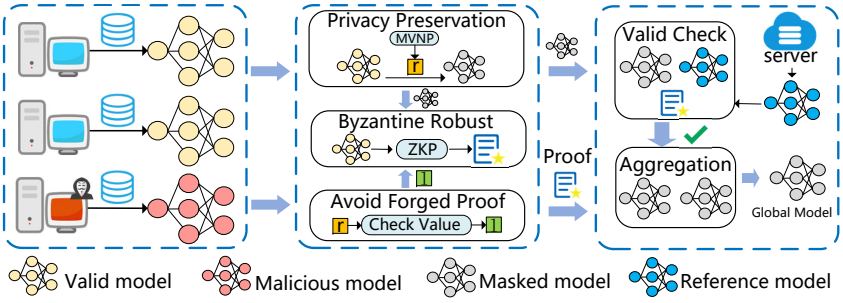


Figure 2: Overview of BPFL.

2 Preliminaries and Threat Model

Federated learning (FL). Suppose we are given a set of n clients $C = \{C_1, C_2, \cdots, C_n\}$ and a server S, where each client $C_i \in C$ holds a dataset D_i. At the beginning, the server initializes a global model w_g^1. In each round t, each client C_i downloads the global model w_g^t from the server S and updates its local model w_i^t by minimizing a task-dependent loss defined on its dataset D_i. Then the server S collects the updated local client models $\{w_i^t\}$ and updates the global model w_g^{t+1} for the next round via an aggregation algorithm. For instance, when using the most common federated averaging (FedAvg) aggregation (McMahan et al., 2017), the updated global model is: $w_g^{t+1} \leftarrow \sum_{i=1}^{n} \frac{1}{n} w_i^t$, assuming that all clients have the same dataset size.

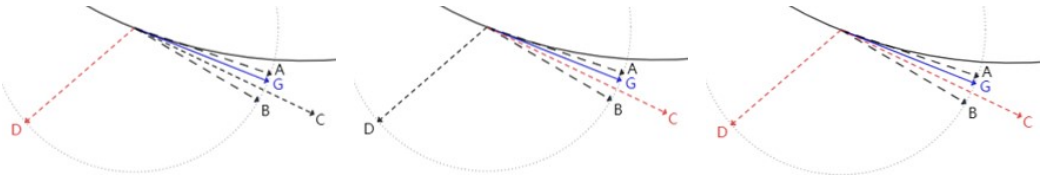
Similarity metrics in Byzantine-robust FL. Most of existing Byzantine-robust FL methods use similarity metrics for robust aggregation. For example, Krum (Bonawitz et al., 2017) uses the Euclidean distance, while state-of-the-art FLTrust (Cao et al., 2021) uses the cosine similarity. We simply introduce these two metrics, which are of interest in this work. Given two m-dimensional vectors $\mathbf{u} = \{u_1, u_2, \cdots, u_m\}$ and $\mathbf{v} = \{v_1, v_2, \cdots, v_m\}$, the Euclidean distance between $\mathbf{u}$ and $\mathbf{v}$ is defined as $s_{Euc}(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{i=1}^{m}(u_i - v_i)^2}$, and the cosine similarity between $\mathbf{u}$ and $\mathbf{v}$ is $s_{cos}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|} = \frac{\sum_{i=1}^{m} u_i \cdot v_i}{\sqrt{\sum_{i=1}^{m}(u_i)^2} \cdot \sqrt{\sum_{i=1}^{m}(v_i)^2}}$.

Zero-Knowledge Proof. The Zero-Knowledge Proof (Goldwasser et al., 1989) enables a prover to convince a verifier that an assertion is correct without providing any privacy information to the verifier. In this work, we use the Groth16 scheme (Groth, 2016), which is a famous Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARK) technology. Groth16 uses the Rank-1 Constraint System (R1CS) to describe the arithmetic circuit and then converts the R1CS satisfiability problem into a Quadratic Arithmetic Program (QAP) (Gennaro et al., 2013) satisfiability problem. The R1CS is a tuple containing seven elements $(\mathbb{F}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{io}, m, n)$, where $\mathbb{F}$ is the finite field, $\mathbf{io}$ is the public input and output vectors, $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{m \times m}, m \geq |\mathbf{io} + 1|, n$ is the maximum number of nonzero values in all matrices. R1CS is satisfiable if and only if there exists a proof $w \in \mathbb{F}^{m-|\mathbf{io}|-1}$ such that $(\mathbf{A}z) \odot (\mathbf{B}z) = (\mathbf{C}z)$, where $z = (\mathbf{io}, 1, w)^T$ and $\odot$ is the hadamard product. The Groth16 algorithm contains three parts, expressed as $\Pi = (Setup, Prove, Verify)$. Given a polynomial time judgeable binary relation $\mathcal{R}$ described by R1CS, Groth16 has the following three steps (more details about Groth16 can be seen in Appendix B.2):

- $(pk, vk) \leftarrow Setup(1^\lambda, \mathcal{R})$: It takes the security parameter λ and the relationship $\mathcal{R}$ as inputs. The algorithm reduces the satisfiability problem of arithmetic circuits to the QAP satisfiability problem, and then a proving key pk and a verification key vk are generated.
- $\pi \leftarrow Prove(pk, I_p, I_a)$: The prover generates a proof π by taking pk, I_p, I_a as input, where I_a is the auxiliary input that only the prover knows and I_p is the primary input that both prover and verifier know.
- $0/1 \leftarrow Verify(vk, \pi, I_p)$: The verifier verifies the proof π by taking vk, π, I_p as input. Only if the proof passes the verification the verifier will get 1 else get 0.

Threat model. In our BPFL, we consider the following two types of adversaries:

- **Malicious clients**. Malicious clients can actively deviate from the protocol to corrupt the global model. Their attack goal includes: (1) submitting a poisoned local model to the server and compromising the final aggregation; (2) attempting to submit forged proofs to deceive the server; (3) failing the valid check of an honest client by providing incorrect proof parameters. Without loss of generality, we assume the number of malicious clients is less than by 50% of the total clients.



(a) Unable to detect malicious update $\boldsymbol{C}$ only with cosine similarity (b) Unable to detect malicious update $\boldsymbol{D}$ only with Euclidean distance (c) When using both metrics, both malicious update can be detected

Figure 3: Motivation of using both the cosine similarity and Euclidean distance to detect malicious model updates. $\boldsymbol{G}$ is the real update, $\boldsymbol{A}$ and $\boldsymbol{B}$ are valid updates, while $\boldsymbol{C}$ and $\boldsymbol{D}$ are malicious updates.

- **Honest-but-curious server**. The server follows the protocol, but aims to infer clients' private information (e.g., raw data) via analyzing the received client models. We assume the server does not collude with malicious clients. Following the existing works (Cao et al., 2021), we also assume the server holds a small and clean dataset D_S, which is used to train a reference model w_S for malicious client models detection.

3 Design of BPFL

In this section, we will introduce the design of BPFL in detail. We first introduce our zero-knowledge proof-inspired valid robustness check algorithm that uses the similarity metrics in the state-of-the-art robust FL methods. Next, we introduce our privacy preservation mechanism based on homomorphic encryption. We further leverage a check value for vectors to address the possible forged proofs caused by malicious clients. Finally, we show the entire workflow of our BPFL. The important notations in this paper are shown in Table 3 in Appendix B.3.

3.1 Valid Robustness Check for Local Models via Zero-Knowledge Proof

As shown in the recent works (Blanchard et al., 2017; Guerraoui et al., 2018; Cao et al., 2021), a Byzantine-robust FL method should require that the local model be similar to the global model, in terms of both the model update direction and the model magnitude (also see an example in Figure 3). Motivated by this, we use both cosine similarity and Euclidean distance to measure the validity of a client model. To avoid complex interaction computation, we use the non-interactive zero-knowledge proof (ZKP) as the implementation of the valid check. Recall that the server holds a benign model w_S (trained on its clean dataset D_S), which we will treat as the reference for local model comparison. We adopt the cosine similarity and Euclidean distance between each client model w_i and the reference model w_S. Then, a valid robust local model should satisfy:

$$\sqrt{\sum_{j=1}^{m}(w_i^j - w_S^j)^2} \leq \tau_e, \quad \frac{\sum_{j=1}^{m} w_i^j \times w_S^j}{\sqrt{\sum_{j=1}^{m}(w_i^j)^2} \times \sqrt{\sum_{j=1}^{m}(w_S^j)^2}} \geq \tau_c, \tag{1}$$

where τ_e and τ_c are the threshold for cosine similarity and Euclidean distance, e.g., defined by the server. Note that Equation 1 contains square root and division computations that are difficult to be expressed by arithmetic circuits, as they only consist of addition and multiplication gates. To address it, we conduct some transformations in order to satisfy the Groth16 scheme. Consider that all computations in Groth16 are in an integer field, we use fixed-point numbers to approximate floating-point numbers. By default, we first define $k = 2^{16}$ and transform the number x to be x' by setting $x' = kx$, and then simply truncate the fractional part. In doing so, the to be verified Equation 1 in the proof circuit are expressed as:

$$\sum_{j=1}^{m}(kw_i^j - kw_S^j)^2 \leq (k\tau_e)^2, \quad \big(k\sum_{j=1}^{m}(kw_i^j \times kw_S^j)\big)^2 \geq (k\tau_c)^2 \times \sum_{j=1}^{m}(kw_i^j)^2 \times \sum_{j=1}^{m}(kw_S^j)^2 \tag{2}$$

As Equation 2 now can be represented by an arithmetic circuit containing only multiplication gates and addition gates, we use R1CS to describe each gate in the arithmetic circuit, and further generate the proof circuit for Groth16. Particularly, in the iteration t, each client C_i downloads the proof parameters $\sigma_t = \{w_S^t, \tau_c, \tau_e\}$ from the server, and defines the primary input $I_p = \{\sigma_t\}$ and auxiliary input $I_a = \{w_i^t\}$ used in the zero-knowledge proof. Each client then generates a proof using I_p and I_a as input, and the server verifies the proof using I_p.

3.2 PRIVACY PRESERVATION FOR LOCAL MODELS VIA HOMOMORPHIC ENCRYPTION

Our privacy protection mechanism is based on homomorphic encryption (see Appendix B.1 its background). Specifically, to protect clients' privacy, all clients hold a same but random vector to mask the true local models. Here, we propose to add this random vector to all local models. Hence, the global model update in FedAvg is rewritten as:

$$\bar{w}_g^{t+1} \leftarrow \mathbf{r} + \frac{1}{n}\sum_{i=1}^{n} w_i^t \leftarrow \sum_{i=1}^{n} \frac{1}{n}(w_i^t + \mathbf{r}) \leftarrow \sum_{i=1}^{n} \frac{1}{n}\bar{w}_i^t, \tag{3}$$

where $\bar{w}$ is the value of w after being masked; $\mathbf{r}$ is the random vector generated by the Mask Vector Negotiation Protocol (MVNP), which leverages the Paillier homomorphic encryption technique (Paillier, 1999). More details about how to generate $\mathbf{r}$ are shown in Algorithm 1 in Appendix B.1. In a word, the MVNP ensures all clients obtain a same random vector that is secret to the server. That is, the server cannot infer any private information from the masked local models as well as their aggregation.

A possible issue is that, since the server receives the masked local models, a malicious client may use the valid model to generate the proof, but submits an invalid model to the server. Hence, the server needs to verify that the model submitted by the client is consistent with the model used to generate the proof. Specifically, the server needs to use the masked client model $\bar{w}_i$ to check whether each local model satisfies $\bar{w}_i^t = w_i^t + \mathbf{r}$. Therefore, $\mathbf{r}$ will also be a parameter for the prover to generate a proof and $\bar{w}_i^t$ will be a primary parameter. So, I_a will be updated to $I_a = \{w_i^t, \mathbf{r}\}$ and I_p will be updated to $I_p = \{\bar{w}_i^t, \sigma_t\}$.

3.3 AVOIDING FORGERY OF PROOF VIA CHECK VALUE

In each iteration t, clients download parameters (including w_S^t in plaintext) from the server to generate the proof. However, a malicious client may use w_S^t as local model to generate a forged proof to fool the server into passing the valid check. Assuming a malicious client C_i holds a masked malicious model $\tilde{w}_i^t$ and has downloaded w_S^t, it can generate a forged proof by letting $\tilde{\pi}_i^t = Prove(pk, I_p, \tilde{I}_a)$, where $\tilde{I}_a = \{w_S^t, \tilde{\mathbf{r}}\}$ and $\tilde{\mathbf{r}}$ is well-constructed vector defined as $\tilde{\mathbf{r}} = \tilde{w}_i^t - w_S^t$. The client C_i submits $\tilde{w}_i^t$ with the proof $\tilde{\pi}_i^t$ and server will get $1 \leftarrow Verify(vk, \tilde{\pi}_i^t, I_p)$. This is because w_S^t in $\tilde{I}_a = \{w_S^t, \tilde{\mathbf{r}}\}$ and w_S^t in $I_p = \{\tilde{w}_i^t, \sigma_t\}$ ($\sigma_t = \{w_S^t, \tau_c, \tau_e\}$) satisfy Equation 2, and $\tilde{w}_i^t$ in $I_p = \{\tilde{w}_i^t, \sigma_t\}$ and $w_S^t, \tilde{\mathbf{r}}$ in $\tilde{I}_a = \{w_S^t, \tilde{\mathbf{r}}\}$ satisfy $\tilde{w}_i^t = w_S^t + \tilde{\mathbf{r}}$. Therefore, this malicious model will be allowed to join the aggregation, which will have a negative impact on the global model.

To avoid this, the server must check whether the random mask $\mathbf{r}$ used by clients to generate the proof is the true vector negotiated by MVNP. We propose a simple yet effective solution to address this issue, i.e., based on *check value*. Specifically, each client C_i maintains a vector M of length e and w.l.o.g, we set $e = 10$ in our work. Each client C_i gets a check value l_i by computing:

$$M_{j \bmod e} \leftarrow M_{j \bmod e} + r_j, \quad l_i \leftarrow \prod_{j=0}^{e} M_j, \tag{4}$$

where r_j is the j-th element of $\mathbf{r}$. The server maintains a set $L, l_i \in L$ and let $l = MODE(L)$, where $MODE(L)$ function finds the most frequent value in the set L. Based on clients' check values and their mode, though all malicious clients (less than 50%) may refuse to compute the true l_i, the true l_i's of more than 50% honest clients are sufficient for the server to obtain the correct l. To ensure the proof is not forged, the server only needs to add a constraint to check the correctness of the computation of Equation 4.

Equation 2, Equation 3, and Equation 4 together form the completed valid check for masked local models. In summary, one local model considered to be valid should satisfy: 1) Similar to reference model in direction and value; 2) Using the true mask vector and computing correctly. The satisfaction of these constraints is contained in a proof $\pi_i^t = Prove(pk, I_p, I_a)$, and the server checks: $0/1 \leftarrow Verify(vk, \pi_i^t, I_p)$, where $I_p = \{\bar{w}_i, l, \sigma_t\}$ and $I_a = \{w_i^t, \mathbf{r}\}$ in each t-th round.

3.4 BPFL WORKFLOW

The overall workflow of BPFL involves a setup phase followed by three rounds. The whole procedure of BPFL are shown in Figure 11 in Appendix B.4).

Setup Phase. The server (1) creates a valid check circuit, generates pk for proving and vk for verification, and broadcasts pk to all clients; and (2) randomly initializes the global model w_g^1 and sets the reference model $w_S^1 \leftarrow w_g^1$, and defines the thresholds τ_c and τ_e. All clients obtain the agreed random mask vector $\mathbf{r}$ through MVNP (Algorithm 1 in Appendix B.1), calculate each l_i, and send them to the server. The server computes l as the final value.

Round 1 (Local training). In iteration t, each client C_i first downloads the global model $\bar{w}_g^t$ from the server and recovers the true value by $w_g^t = \bar{w}_g^t - \mathbf{r}$ (except for the first round). Then each client C_i trains a local model w_i^t with the dataset D_i, masks w_i^t with $\mathbf{r}$, and submits the masked model $\bar{w}_i^t = w_i^t + \mathbf{r}$ to the server. The server trains the reference model w_S^t with the datasets D_s, which will be used as one of the parameters for this round to generate a proof.

Round 2 (Clients generate and submit proof). All clients first get the proof parameters $\sigma_t = \{w_S^t, \tau_c, \tau_e\}$ from the server. Then each C_i generates the proof of w_i^t with σ_t by letting $\pi_i^t = Prove(pk, I_p, I_a)$, where $I_p = \{\bar{w}_i^t, l, \sigma_t\}$ and $I_a = \{w_i^t, \mathbf{r}\}$ and submits π_i^t to the server.

Round 3 (Server verifies the proof and performs aggregation). The validity of the proof will be checked and the well-formed local models will be aggregated by the server S. Specifically, the server maintains a list, U^+ (initialized as empty), of local models it has so far identified as valid. The server checks the validity of all models by verifying the proofs. For a local model $\bar{w}_i^t$ with a proof π_i^t, if the proof passes the verification, i.e., $Verify(vk, \pi_i^t, I_p) = 1$, then the server treats $\bar{w}_i^t$ as a valid model and augments the set $U^+ \leftarrow U^+ \cup \bar{w}_i^t$. Otherwise, the local model will be flagged as malicious and dropped. Every masked local model $\bar{w}_i^t$ in U^+ will be aggregated into a global model $\bar{w}_g^{t+1}$ using the FedAvg algorithm and be downloaded by all clients used for the next iteration.

4 Theoretical Analysis

Complexity analysis. We show the complexity of BPFL w.r.t. #clients n and #model parameters d.

1) Computation cost. In each iteration, each client's computation cost can be split into two parts: (1) creating the masked local model is $O(d)$ in **Round 1**; (2) generating the zero-knowledge proof in **Round 2**. The prover's computation complexity of Groth16 is $O(m \log m)$, where m is the number of gates in the circuit. According to our valid check constraints in Equation 2, Equation 3, and Equation 4, the number of gates m increases linearly in d, so the complexity is $O(d \log d)$; (3) recovering the true global update is $O(d)$ in **Round 3**. Thus, the overall computation complexity of each client per iteration is $O(d \log d)$. The computation cost of server is mainly at **Round 3**, which includes verifying the proof for all clients and computing the final aggregation. The verifier's computation complexity of Groth16 is $O(1)$ and the aggregation complexity is $O(nd)$. Therefore, the total computation complexity of the server per iteration is $O(nd)$.

2) Communication cost. In each iteration, the client sends the local model, which has a $O(d)$ complexity, and a proof, which has a fixed size and the complexity is $O(1)$, to the server. Thus, the communication complexity for every client is $O(d)$. The server's communication costs include: (1) sending the global model to all clients which has a $O(nd)$ complexity; (2) sending the proof parameters to all clients which have a $O(nd)$ complexity. Hence, the overall communication complexity of the server is $O(nd)$.

Security analysis. We formally show that BPFL is privacy-preserving in the following theorems.

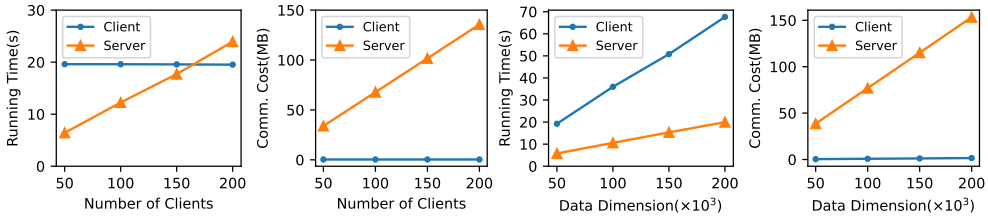
Theorem 1. *BPFL is privacy-preserving if Paillier homomorphic encryption used in Algorithm 1 is semantically secure and Gorth16 is zero-knowledge.*

Proof. See Appendix C. □

5 Experiments

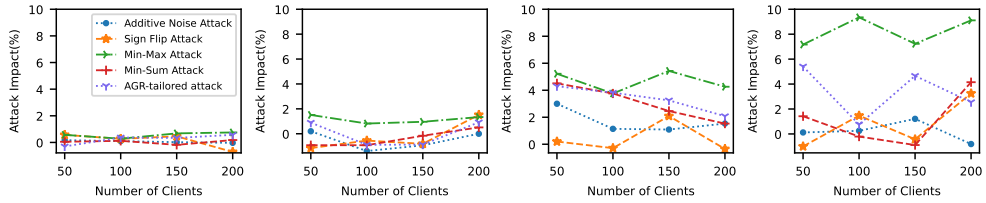
5.1 Experimental Setup

We implement BPFL in Python and using the C++ libsnark library for zkSNARK proofs. We run experiments on four physical machines and each with 40 Intel(R) Xeon(R) Silver 4210 CPU at 2.20GHz, 64GB memory, and an NVIDIA GeForce GTX 1080 Ti GPU. All experiments are under Ubuntu 20.04.4 LTS.



(a) $d = 50K$ (b) $d = 50K$ (c) $n = 50$ (d) $n = 50$

Figure 4: Overhead analysis of BPFL. (a) and (b) show the running time and communication cost per client with the number of clients, the data dimension is fixed to be 50K. (c) and (d) show the running time and communication cost per client with the dimension of data, the number of clients is fixed to be 50.



(a) MNIST (b) FMNIST (c) CIFAR10 (d) FEMNIST

Figure 5: Attack impact on BPFL under different model poisoning attacks and datasets vs. #total clients.

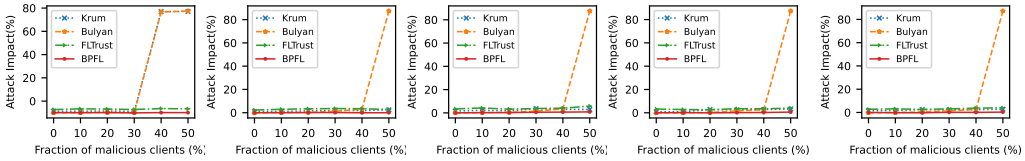
We evaluate BPFL on the following four image datasets, where the first three datasets are independent identically distributed (IID) and the last one is non-IID distributed.

- **MNIST**. A handwritten digit dataset consisting of 60K training images and 10K test images with ten classes. We train a CNN architecture that has five layers and 24,000 parameters for MNIST.
- **FMNIST**. A dataset has a predefined training set of 60K fashion images and a testing set of 10K fashion images. We use LeNet-5 to experiment on FMNIST.
- **CIFAR-10**. A dataset contains RGB images with ten object classes. It has 50K training and 10K test images. We use ResNet-20 and 294,000 parameters for our experiments on CIFAR-10.
- **FEMNIST**. Federated Extended MNIST, built by partitioning the data in Extended MNIST (Cohen et al., 2017), is a non-IID dataset with 3,400 clients, 62 classes, and a total of 671,585 grayscale images. We use LeNet-5 to experiment on FEMNIST and we randomly select n out of 3,400 clients for FL training.

For each dataset, we randomly select 200 samples from the training set as the server's validation dataset D_S and the remaining training data are randomly and evenly divided into n subsets as each client's local training data, where n is the total number of clients. For the thresholds τ_c and τ_e, we empirically set $\tau_c = 0.99$ on the four datasets; set $\tau_e = 0.93$ on MNIST, FMNIST and FEMNIST, and $\tau_e = 30.00$ on CIFAR-10, considering the different number of pixels in these datasets. We set the clients' local training epochs as 5 and global iterations as 300. Our source code is available at the Github: `https://github.com/BPFL/BPFL`.

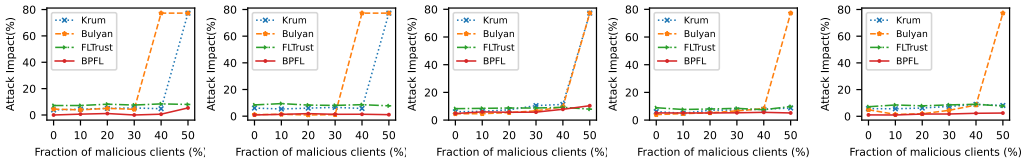
5.2 Experimental Results

Overhead evaluation. Figure 4 shows BPFL's runtime and communication cost with respect to the number of clients n and data dimension d. In Figure 4(a) and 4(b), we fixed the data dimension as $d = 50K$. We can observe that the server's runtime and communication cost is linear in n while the runtime and communication cost of per client is almost stable, i.e. independent of the number of clients. The result is the same as we expected because the zero-knowledge proof enables each client to independently generate proof locally without interacting with others and the server also independently verifies each client's proof locally. This ensures BPFL will not cause a sharp overhead increase when the number of clients increases. Figure 4(c) and 4(d) show the overhead with data dimension, where we fix the number of clients to be $n = 50$. We see that per clients' runtime and communication cost increases almost linearly with the increase of data dimension (recall that the clients have an $O(d \log d)$ computation complexity and $O(d)$ communication complexity). For server, the runtime and communication cost are both linear to the increasing data dimension. Note that both the server's computation complexity and communication complexity are $O(d)$.



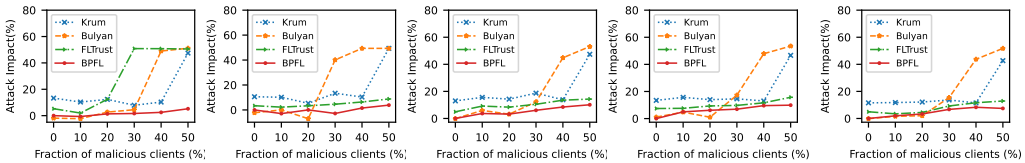
(a) Add Noise Attack (b) Sign Flip Attack (c) Min-Max Attack (d) Min-Sum Attack (e) AGR Attack

Figure 6: Impact of the fraction of malicious clients on the attack impact of different Byzantine-robust FL methods under different attacks on MNIST.



(a) Add Noise Attack (b) Sign Flip Attack (c) Min-Max Attack (d) Min-Sum Attack (e) AGR Attack

Figure 7: Impact of the fraction of malicious clients on the attack impact of different Byzantine-robust FL methods under different attacks on FMNIST.



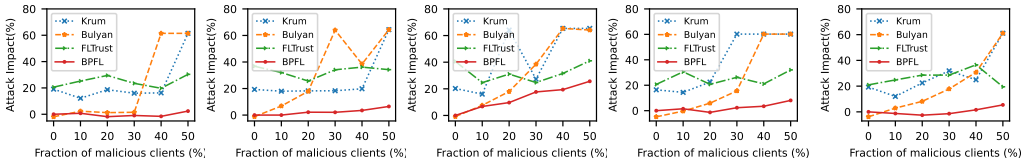
(a) Add Noise Attack (b) Sign Flip Attack (c) Min-Max Attack (d) Min-Sum Attack (e) AGR Attack

Figure 8: Impact of the fraction of malicious clients on the attack impact of different Byzantine-robust FL methods under different attacks on CIFAR-10.

Robustness evaluation. In this experiment, we evaluate BPFL in terms of Byzantine-robustness. For comparison, we also choose three well known Byzantine-robust methods, i.e., Krum (Blanchard et al., 2017), Bulyan (Guerraoui et al., 2018), and the state-of-the-art FLTrust (Cao et al., 2021). We consider five model poisoning attacks: (1) *Add Noise Attack* (Li et al., 2019): malicious clients add a noise to the local model; (2) *Sign Flip Attack* (Damaskinos et al., 2018): malicious clients flip the sign of their local model; (3) *Min-Max Attack*; (4) *Min-Sum Attack*; and (5) *AGR attacks*: three state-of-the-art model poisoning attacks proposed in (Shejwalkar & Houmansadr, 2021). Following Shejwalkar & Houmansadr (2021), we use the metric *attack impact*, which is defined as the reduction in the accuracy of the global model due to the attack, to measure the impact of attacks. An attack having a large attack impact implies it is more effective.

1) Impact of the total number of clients: Figure 5 shows the attack impact of the five attacks to BPFL on the four datasets, where the fraction of malicious clients is set to be 20%. We observe that 1) Min-Max, Min-Sum, and AGR attacks have larger attack impacts than Add Noise and Sign Flip attacks, showing they are more effective; 2) BPFL achieves a small attack impact under different attacks when facing different number of clients, i.e., less than 6% (most less than 2%) in almost all cases. The attack impact on the non-IID FEMNIST is slightly larger than that on the IID datasets. One reason is that the local models across different clients can be more diverse when trained on non-IID data, which thus makes it more challenging to use a threshold to differentiate between honest models and malicious models.

2) Impact of the fraction of malicious clients: Figure 6 to Figure 9 shows the attack impact of the fraction of malicious clients on Byzantine-robust FL methods against the five attacks on the four datasets. We fix the total number of clients to be 50. We have the following observations: 1) BPFL performs the best to defend against the five attacks and the attack impact is less than 6% in all IID datasets and less than 10% in the non-IID FEMNIST. This is because BPFL leverages both the Euclidean distance and cosine similarity as the metrics to identify malicious clients, while the remaining robust methods only use one of the two metrics. 2) The state-of-the-art FLTrust performs next to our BPFL. One reason is that both of them train a reference model in the server using a



(a) Add Noise Attack (b) Sign Flip Attack (c) Min-Max Attack (d) Min-Sum Attack (e) AGR Attack

Figure 9: Impact of the fraction of malicious clients on the attack impact of different Byzantine-robust FL methods under different attacks on FEMNIST.

clean validation dataset and use this reference model to guide the correct model update. 3) Krum and Bulyan fail to defend against some attacks (e.g., Add Noise atack and Sign Flip attack) when the fraction of malicious clients is larger than a certain threshold (e.g., 40% on MNIST). Note that Cao et al. (2021) also draw this conclusion. The above observations verify that, it is important to consider two similarity metrics and train a reference model to defend against Byzantine attacks.

Table 1: BPFL and plaintext against model inversion attacks. ↓ (↑) means a smaller (larger) value indicates a larger privacy leakage.

		MSE ↓		PSNR ↑		SSIM ↑	
		plaintext	BPFL	plaintext	BPFL	plaintext	BPFL
MNIST	DLG	2.9×10^{-7}	1.10	73.38	-0.436	0.999	0
	iDLG	3.8×10^{-7}	1.11	69.201	-0.469	0.989	0
	R-GAP	1×10^{-8}	0.01	251.843	10.00	0.999	0.102
FMNIST	DLG	1.9×10^{-5}	1.22	72.668	-0.839	0.999	0.001
	iDLG	5.3×10^{-6}	1.21	70.413	-0.829	0.998	0.002
	R-GAP	1.2×10^{-6}	0.09	245.656	7.623	0.999	0.069
CIFAR10	DLG	7.9×10^{-4}	1.26	50.94	-0.988	0.987	0.008
	iDLG	5.8×10^{-5}	1.26	47.297	-1.01	0.998	0.008
	R-GAP	4.4×10^{-5}	0.34	31.25	4.181	0.97	0.172
FEMNIST	DLG	7×10^{-8}	1.94	73.24	-2.877	0.999	0.013
	iDLG	1.3×10^{-7}	1.96	71.337	-2.92	0.999	0.012
	R-GAP	4.2×10^{-4}	0.93	239.605	0.172	0.999	0

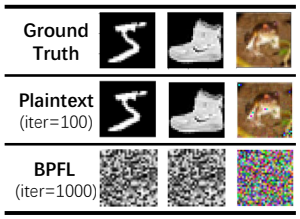


Figure 10: Recovered images by the DLG attack against the plaintext and BPFL. Detailed recovery process and recovered images by iDLG and R-GAP attacks are referred to Appendix D.

Privacy-preserving evaluation. In this experiment, we evaluate four well-known model inversion attacks, i.e., DLG (Zhu et al., 2019), iDLG (Zhao et al., 2020), and R-GAP (Zhu & Blaschko, 2020), against conventional plaintext local models and BPFL's local models on the four datasets. To measure the attack effectiveness, we randomly select 100 images in each dataset and compute the average value of Mean Square Error (MSE), Peak Signal to Noise Ratio (PSNR), and Structural Similarity Index Measure (SSIM) between the recovered images by the four attacks and the real images. Table 1 shows the results. We observe that all attacks achieve very good attack performance on plaintext (i.e., low MSE, large PSNR, and large SSIM), but perform poorly on BPFL. We also randomly show in Figure 10 the recovered images under the DLG attack. More recovered images and under iDLG and R-GAP attacks can be seen in Appendix D. We observe that for plaintext models, the real images from different datasets can be recovered within 100 iterations. In contrast, with BPFL, the attack cannot recover any useful information of the true images even with 1,000 iterations. This is because BPFL can theoretically protect the client models from being inferred.

6 Conclusion

We study defenses against Byzantine (security) attacks and data reconstruction (privacy) attacks to FL. To this end, we propose BPFL, the first FL method that is efficient, Byzantine-robust, and provably privacy-preserving. BPFL seamlessly integrates the ideas of existing Byzantine-robust FL methods, zero-knowledge proof, and homomorphic encryption into a framework. For instance, the validity of clients are verified via a zero-knowledge proof, where the circuit constraints define the similarity metrics between client models and the reference model, motivated by the existing robust FL methods. The clients' data privacy are protected by only sending the server masked models, where the shared mask for all clients is generated based on homomorphic encryption. Experimental results demonstrate that BPFL is efficient, Byzantine-robust, and privacy-preserving.

REFERENCES

Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2017.

Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 2938–2948. PMLR, 2020.

Mislav Balunovic, Dimitar Iliev Dimitrov, Robin Staab, and Martin Vechev. Bayesian framework for gradient leakage. In *International Conference on Learning Representations*, 2022.

Gilad Baruch, Moran Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pp. 634–643, 2019.

Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in Neural Information Processing Systems*, 30, 2017.

Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. *CoRR*, abs/1902.01046, 2019. URL http://arxiv.org/abs/1902.01046.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM SIGSAC Conference on Computer and Communications Security*, 2017.

Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. In *NDSS*, 2021.

Lingjiao Chen, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. Draco: Byzantine-resilient distributed training via redundant gradients. In *International Conference on Machine Learning*, pp. 903–912, 2018.

Yudong Chen, Lili Su, and Jiaming Xu. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):1–25, 2017.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017.

Georgios Damaskinos, Rachid Guerraoui, Rhicheek Patra, Mahsa Taziki, et al. Asynchronous byzantine machine learning (the case of sgd). In *International Conference on Machine Learning*, 2018.

Gábor Danner and Márk Jelasity. Fully distributed privacy preserving mini-batch gradient descent learning. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, 2015.

Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine-robust federated learning. In *29th {USENIX} Security Symposium*, 2020.

Marc Fischlin and Arno Mittelbach. An overview of the hybrid argument. Cryptology ePrint Archive, Paper 2021/088, 2021. URL https://eprint.iacr.org/2021/088. https://eprint.iacr.org/2021/088.

Liam H Fowl, Jonas Geiping, Wojciech Czaja, Micah Goldblum, and Tom Goldstein. Robbing the fed: Directly obtaining private data in federated learning with modified models. In *International Conference on Learning Representations*, 2022.

Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients-how easy is it to break privacy in federated learning? *Advances in Neural Information Processing Systems*, 33:16937–16947, 2020.

Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 626–645. Springer, 2013.

Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv*, 2017.

Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pp. 305–326. Springer, 2016.

Rachid Guerraoui, Sébastien Rouault, et al. The hidden vulnerability of distributed learning in byzantium. In *International Conference on Machine Learning*, pp. 3521–3530. PMLR, 2018.

Jihun Hamm, Yingjun Cao, and Mikhail Belkin. Learning privately from multiparty data. In *International Conference on Machine Learning*, 2016.

Meng Hao, Hongwei Li, Guowen Xu, Hanxiao Chen, and Tianwei Zhang. Efficient, private and robust federated learning. In *Annual Computer Security Applications Conference*, pp. 45–60, 2021.

Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *CCS*, 2017.

Jinwoo Jeon, Jaechang Kim, Kangwook Lee, Sewoong Oh, and Jungseul Ok. Gradient inversion with generative image prior. *Advances in Neural Information Processing Systems*, 2021.

Liping Li, Wei Xu, Tianyi Chen, Georgios B Giannakis, and Qing Ling. Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Xinjian Luo, Yuncheng Wu, Xiaokui Xiao, and Beng Chin Ooi. Feature inference attack on model predictions in vertical federated learning. In *IEEE 37th International Conference on Data Engineering*, 2021.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *IEEE Symposium on Security and Privacy (SP)*, 2019.

Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017.

Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pp. 223–238. Springer, 1999.

Manas Pathak, Shantanu Rane, and Bhiksha Raj. Multiparty differential privacy via aggregation of locally trained classifiers. In *Advances in Neural Information Processing Systems*, 2010.

Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *arXiv preprint arXiv:1912.13445*, 2019.

Virat Shejwalkar and Amir Houmansadr. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *NDSS*, 2021.

Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 1310–1321. ACM, 2015.

Jingwei Sun, Ang Li, Binghui Wang, Huanrui Yang, Hai Li, and Yiran Chen. Provable defense against privacy leakage in federated learning from representation perspective. In *CVPR*, 2021.

Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *INFOCOM*, 2019.

Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

Zhaoxian Wu, Qing Ling, Tianyi Chen, and Georgios B Giannakis. Federated variance-reduced stochastic gradient descent with robustness to byzantine attacks. *IEEE Transactions on Signal Processing*, 2020.

Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2019a.

Cong Xie, Sanmi Koyejo, and Indranil Gupta. Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance. In *International Conference on Machine Learning*, pp. 6893–6901. PMLR, 2019b.

Jie Xu, Benjamin S Glicksberg, Chang Su, Peter Walker, Jiang Bian, and Fei Wang. Federated learning for healthcare informatics. *Journal of Healthcare Informatics Research*, 5(1):1–19, 2021.

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, 2018.

Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via gradinversion. In *CVPR*, 2021.

Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. {BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning. In *USENIX ATC*, 2020.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610*, 2020.

Junyi Zhu and Matthew Blaschko. R-gap: Recursive gradient attack on privacy. *arXiv preprint arXiv:2010.07733*, 2020.

Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in neural information processing systems*, 32, 2019.

A Related Work

A.1 Byzantine-robust FL

A series of Byzantine-robust FL has been proposed recently (Blanchard et al., 2017; Chen et al., 2017; Guerraoui et al., 2018; Yin et al., 2018; Guerraoui et al., 2018; Chen et al., 2018; Pillutla et al., 2019; Xie et al., 2019b; Wu et al., 2020; Cao et al., 2021). They majorly leverage the similarity between local client models and the global model to perform robust aggregation. Blanchard et al. (2017) proposed Krum, the first solution to defend against Byzantine attacks. The idea of Krum is to treat a local model as benign if this local model is similar to other local client models, where the similarity is measured by Euclidean distance. Krum is shown to tolerate f malicious clients out of the total n clients, where n and f satisfy $2f + 2 < n$. Yin et al. (2018) proposed a median-based aggregation, where the server obtains the j-th parameter of its global model by calculating the median of the j-th parameter of all the n local client models. Guerraoui et al. (2018) proposed Bulyan, which combines the idea of Krum and median. Bulyan first iteratively uses Krum to select $k(\leq n - 2f)$ client model. Then, Bulyan aggregates the k client models using a variant of the trimmed mean. These methods can defend against malicious clients to some extent, but are still not effective enough. To further enhance the performance, Cao et al. (2021) proposed FLTrust. In FLTrust, the server holds a clean validation dataset, which is used to train a reference model that guides the correct model update direction. Specifically, the server assigns a trust score to each client model update based on the cosine similarity between the client model and the reference model. The trust score is used as the weight of the client's local update to participate in the model aggregation. Only client models with relatively larger weights are allowed to participate in the global model aggregation. FLTrust is shown to obtain the state-of-the-art robustness against Byzantine attacks.

A.2 Privacy Preserving FL

Existing provably privacy-preserving FL methods can be divided into three categories: differential privacy (DP) (Pathak et al., 2010; Shokri & Shmatikov, 2015; Hamm et al., 2016; McMahan et al., 2018; Geyer et al., 2017; Wei et al., 2020), secure multi-party computation (MPC) (Danner & Jelasity, 2015; Mohassel & Zhang, 2017; Bonawitz et al., 2017; Melis et al., 2019), and homomorphic encryption (HE) (Aono et al., 2017; Zhang et al., 2020). For example, Wei et al. (2020) proposes a novel differential privacy FL framework, in which carefully designed noises are added to the local client models before aggregation. However, the current DP-based methods have high utility losses. MPC-based methods ensure local clients and the server to jointly complete the aggregation without disclosing the clients' private data. However, they incur an intolerable computation and communication overhead. For instance, the secure aggregation based on MPC in (Bonawitz et al., 2017) has a computation complexity $O(n^2 + nd + nd^2)$, which is quadatic in the number of clients n and parameters d, while our BPFL is linear to both n and d. HE-based methods encrypt local client models before submitting them to the server. Due to the property of HE, the server can complete the global model aggregation without the need to perform decryption. HE-based methods obtain state-of-the-art efficiency and BPFL leverages HE to protect client's data.

Table 2: Comparison between BPFL and existing representative works. B.R.: Byzantine-Robust; P.P.: Privacy-Preserving; Eff.: Efficient

Method	B.R.	P.P.	Eff.
Krum Blanchard et al. (2017)	✓	✗	✓
Bonawitz et al. (2017)	✗	✓	✗
Wei et al. (2020)	✗	✓	✓
FLTrust Cao et al. (2021)	✓	✗	✓
Hao et al. (2021)	✓	✓	✗
BPFL	✓	✓	✓

Note that almost all the current works focus on either designing Byzantine-robust FL or privacy-preserving FL, but not the both. To our best knowledge, only one work (Hao et al., 2021) called SecureFL considered both. However, SecureFL requires an additional server to do cryptographic computations and the overhead is significant as the number of clients increases. Moreover, the two servers need to have interactive communications, which induces intolerable communication overheads as well. In contrast, our BPFL simultaneously achieves the both goals of byzantine-robust and privacy-preserving without incurring excessive overhead to the server via non-interactive ZKP. Table 2 shows the comparisons between BPFL and the existing works.

B MORE PRELIMINARIES AND BPFL

B.1 HOMOMORPHIC ENCRYPTION AND MVNP

Homomorphic encryption (HE) schemes allow certain mathematical operations to be performed directly on ciphertexts, without prior decryption. The Paillier encryption algorithm is an encryption algorithm for additive homomorphism and is semantically secure. The construction of the Paillier encryption system consists of three algorithms: key generation algorithm, encryption algorithm and decryption algorithm.

- $(\boldsymbol{pk}, \boldsymbol{sk}) \leftarrow KeyGen(1^{\kappa})$: The $KeyGen$ algorithm outputs a public key $\boldsymbol{pk}$ and a private key $\boldsymbol{sk}$ with inputting a security parameter κ.
- $c \leftarrow Enc(\boldsymbol{pk}, m)$: The $Enc(\cdot)$ algorithm uses the public key $\boldsymbol{pk}$ to encrypt plaintext m to ciphertext c.
- $m \leftarrow Dec(\boldsymbol{sk}, c)$: The $Dec(\cdot)$ algorithm uses the private key $\boldsymbol{sk}$ to decrypt ciphertext c into plaintext m.

The Paillier encryption scheme supports homomorphic addition operations and scalar multiplication operations. For ciphertext $c_1 = Enc(\boldsymbol{pk}, m_1)$, $c_2 = Enc(\boldsymbol{pk}, m_2)$ and a scalar l, it satisfies: $Dec(\boldsymbol{sk}, c_1 \times c_2) = m_1 + m_2$ and $Dec(\boldsymbol{sk}, \text{pow}(c_i, l)) = m_i \times l$, for $i = 1, 2$. Algorithm 1 designs a Mask Vector Negotiation Protocol (MVNP) to generate a shared mask vector for all clients based on Paillier homomorphic encryption.

Algorithm 1 Mask Vector Negotiation Protocol (MVNP) via Paillier homomorphic encryption

Input: A set of n clients; a Paillier encryption public key $\boldsymbol{pk}$ and a private key $\boldsymbol{sk}$;
Output: A random mask vector $\mathbf{r}$

```
// Client side.
for i = C_1, C_2, ..., C_n do
    C_i generate a random seed s_i randomly.
    Encrypt the s_i with pk: [s_i] = Enc(s_i, pk).
    Send [s_i] to the server.
end for
// Server side.
Compute [s] = sum_{i=1}^{n} [s_i].
Broadcast [s].
// Client side.
Decrypt [s]: s = Dec([s], sk).
Generate a random mask vector r over the finite field with the seed s.
return r
```

B.2 GROTH16 AND QAP

The Groth16 scheme is based on Quadratic Arithmetic Program (QAP) (Gennaro et al., 2013). The QAP, expressed as $\mathcal{Q} = (t(z), \mathcal{U}, \mathcal{W}, \mathcal{Y})$, in finite field $\mathbb{F}$ contains three polynomials $\mathcal{U} = u_k(z), \mathcal{W} = w_k(z), \mathcal{Y} = y_k(z)(k \in 0 \cup [m])$ and a target polynomial $t(z)$. Let $(c_1, c_2, \cdots, c_N)$ be the public input, $\mathcal{Q}$ is *satisfiable* if and only if there exists $(c_{N+1}, c_{N+2}, \cdots, c_m)$ such that $t(z)$ divides $p(z)$, where

$$p(z) = (u_0(z) + \sum_{k=1}^{m} c_k \cdot u_k(z)) \cdot (w_0(z) + \sum_{k=1}^{m} c_k \cdot w_k(z)) - (y_0(z) + \sum_{k=1}^{m} c_k \cdot y_k(z)).$$

That is, there exists a polynomial $h(z)$ such that $p(z) - h(z)t(z) = 0$.

The three phases in the protocol of Groth16 run as follow:

- Setup Phase.
  1. Generate QAP$(t(z), \mathcal{U}, \mathcal{W}, \mathcal{Y})$ according to arithmetic circuit C.

2. Generate $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ which are groups of prime order p. The pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map. Let $[a]_1$ be g^a, $[b]_2$ be h^b, $[c]_T$ be $e(g,h)^c$. Select random numbers $\alpha, \beta, \gamma, \delta, s \xleftarrow{\$} \mathbb{F}$.
3. Generate reference string $\sigma = ([\sigma_1]_1, [\sigma_2]_2)$ and an analog threshold $\tau = (\alpha, \beta, \gamma, \delta, s)$, with

$$\sigma_1 = \begin{pmatrix} \alpha, \beta, \delta, \{s^i\}_{i=0}^{d-1}, \{\frac{\beta u_i(s)+\alpha w_i(s)+y_i(s)}{\gamma}\}_{i=0}^{N} \\ \{\frac{\beta u_i(s)+\alpha w_i(s)+y_i(s)}{\delta}\}_{i=N+1}^{m}, \{\frac{s^i t(s)}{\delta}\}_{i=0}^{n-2} \end{pmatrix}, \sigma_2 = (\beta, \gamma, \delta, \{s^i\}_{i=0}^{d-1}).$$

- Prove Phase. The prover $\mathcal{P}$ generates the proof. $\mathcal{P}$ randomly selects $r_1, r_2 \xleftarrow{\$} \mathbb{F}$ and generates proof $\pi = ([A]_1, [C]_1, [B]_2)$, where

$$A = \alpha + \sum_{i=0}^{m} c_i u_i(s) + r_1\delta, B = \beta + \sum_{i=0}^{m} c_i w_i(s) + r_2\delta,$$

$$C = \frac{\sum_{i=N+1}^{m} c_i(\beta u_i(s) + \alpha w_i(s) + y_i(s)) + h(s)t(s)}{\delta} + Ar_2 + Br_1 - r_1 r_2 \delta.$$

- Verify Phase. Validator $\mathcal{V}$ verifies the proof. The $\mathcal{V}$ check:

$$e([A]_1, [B]_2) \stackrel{?}{=} e([\alpha]_1, [\beta]_2) e(\sum_{i=0}^{N} c_i [\frac{\beta u_i(s) + \alpha w_i(s) + y_i(s)}{\gamma}]_1, [\gamma]_2) e([C]_1, [\delta]_2).$$

Table 3: Important notations used in the paper.

Notation	Meaning
w_i	the local model for client C_i
w_S	the reference model on server
w_g	the global model on server
w_i^t	w_i in round t
w_S^t	w_S in round t
w_g^t	w_g in round t
r	random mask vector
$\bar{w}_i$	the masked local model for client C_i
$\bar{w}_g$	the masked global model
τ_c	threshold for cosine similarity
τ_e	threshold for Euclidean distance
σ_t	proof parameters for round t
I_a	the auxiliary input
I_p	the primary input
π_i^t	the proof of client C_i for round t
$\tilde{r}$	the forged r
$\tilde{w}$	the forged w
$\tilde{\pi}$	the forged π

B.3 Notations

The used important notations in the paper are shown in Table 3.

B.4 Detailed BPFL

The whole BPFL procedure is illustrated in Figure 11.

C Proof of Theorem 1

Before representing the proof, we will need to involve the following definition.

Setup Phase:
Client i:
- Generate the agreed random mask vector r through MVNP.
- Calculate l_i and submit to server.
Server:
- Creates a valid check zero-knowledge circuit.
- Generates pk for proving and vk for verification and broadcasts pk to all client.
- Initializes the global model w_g^1 randomly and sets the reference model $w_S^1 \leftarrow w_g^1$.
- Defines the thresholds τ_c and τ_e.
- Collects l_i from clients and computes l.
Round 1(Local Training):
Client i:
- Download the global model $\bar{w}_g^t$ from the server and recover the true value by $w_g^t = \bar{w}_g^t - \mathbf{r}$ (except for the first round).
- Trains a local model w_i^t with the dataset D_i.
- Mask w_i^t with $\mathbf{r}$ by $\bar{w}_i^t = w_i^t + \mathbf{r}$, and submits the masked local model $\bar{w}_i^t$ to the server.
Server:
- Trains the reference model w_S^t with the datasets D_s.
Round 2(Clients generate and submit proof):
Client i:
- Download the proof parameters $\sigma_t = \{w_S^t, \tau_c, \tau_e\}$ from the server.
- Generate the proof of w_i^t with σ_t by letting $\pi_i^t \leftarrow Prove(pk, I_p, I_a)$, where $I_p = \{\bar{w}_i^t, l, \sigma_t\}$ and $I_a = \{w_i^t, \mathbf{r}\}$.
- Submit π_i^t to the server.
Round 3(Server verifies the proof and performs aggregation):
Server:
- Initializes the list U^+ as empty.
- For a local model $\bar{w}_i^t$ with a proof π_i^t, checks the validity of local models by $Verify(vk, \pi_i^t, I_p)$.
- Sets $U^+ \leftarrow U^+ \cup \bar{w}_i^t$ if $Verify(vk, \pi_i^t, I_p) = 1$ else drop $\bar{w}_i^t$.
- Aggregates U^+ into a global model $\bar{w}_g^{t+1}$ using the FedAvg algorithm.

Figure 11: The whole procedure of BPFL.

Definition 1. *A protocol is privacy-preserving if, for every efficient real-world adversary $\mathcal{A}$, there exists an efficient ideal-world simulator $\mathcal{S}_\mathcal{A}$ such that for every efficient environment $\mathcal{E}$ the output of $\mathcal{E}$ when interacting with the adversary $\mathcal{A}$ in a real-world execution and when interacting with the simulator $\mathcal{S}_\mathcal{A}$ in an ideal-world execution are computationally indistinguishable.*

Theorem 1. *BPFL is privacy-preserving if Paillier homomorphic encryption used in Algorithm 1 is semantically secure and Gorth16 is zero-knowledge.*

Proof. We use the hybrid argument (Fischlin & Mittelbach, 2021) to prove that our BPFL satisfies the security Definition 1. To do this, for every real-world (efficient) adversary $\mathcal{A}$, we construct an ideal-world (efficient) simulator $\mathcal{S}$ whose random variable is distributed exactly as real-world. We use a sequence of hybrids, each identified by Hyb_i and denote by $\text{Output}_i(\mathcal{E})$ the output of $\mathcal{E}$ in Hyb_i, and by Hyb_0 the real-world execution.

- Hyb_1. Let Hyb_1 be the same as Hyb_0, except the followings: $\mathcal{S}$ replaces the encrypted r_i with encrypted random values (e.g., 0, with appropriate length). Given the Paillier homomorphic encryption is semantically secure, $\text{Output}_1(\mathcal{E})$ is perfectly indistinguishable from $\text{Output}_0(\mathcal{E})$.
- Hyb_2. In this hybrid, all honest clients replace the mask value r with uniformly random number to get check value. The Integer Factorization Problem guarantees $\text{Output}_2(\mathcal{E})$ is perfectly indistinguishable from $\text{Output}_1(\mathcal{E})$.
- Hyb_3. In this hybrid, the simulator $\mathcal{S}$ changes the behavior of all honest clients. Specifically, for each client C_i, a uniformly random number is selected to replace the w_i. Because the ideal-world random variable is distributed exactly as real-world and Groth16 algorithm is zero-knowledge, $\text{Output}_3(\mathcal{E})$ is perfectly indistinguishable from $\text{Output}_2(\mathcal{E})$.

Therefore, the simulator has already completed the simulation since $\mathcal{S}$ successfully simulates real-world without knowing anything about the private inputs. The final hybrid is distributed identically to the operation of $\mathcal{S}$ from the point of view of $\mathcal{E}$. We have thus shown that $\mathcal{E}$'s advantage in distinguishing the interaction with $\mathcal{S}$ from the interaction with $\mathcal{A}$ is negligible. □

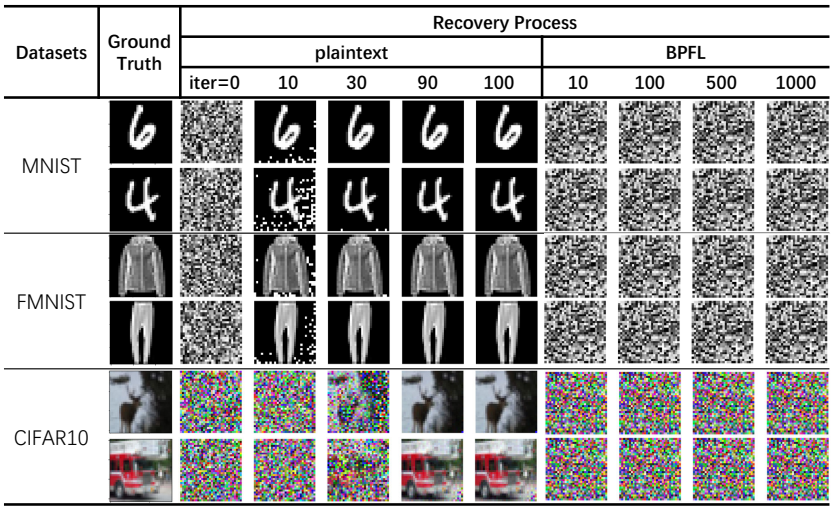


Figure 12: More recovered images by the DLG attack against the plaintext and the privacy-preserving BPFL.

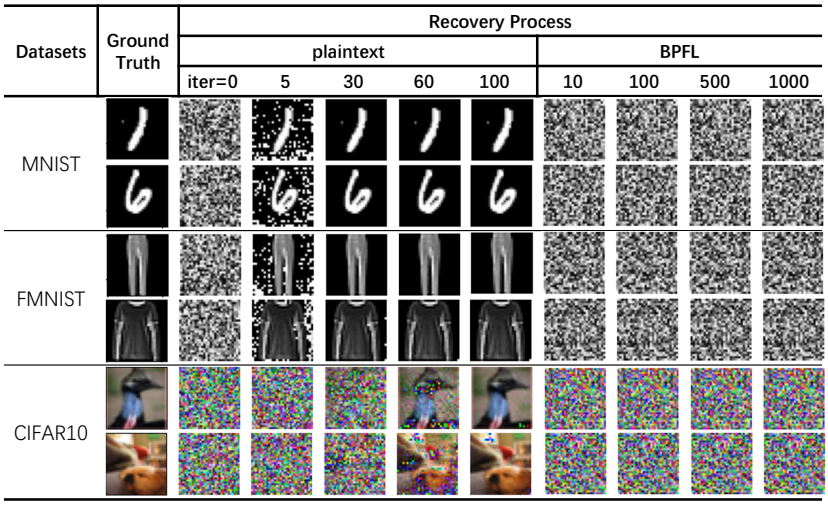


Figure 13: Recovered images by the iDLG attack against the plaintext and the privacy-preserving BPFL.

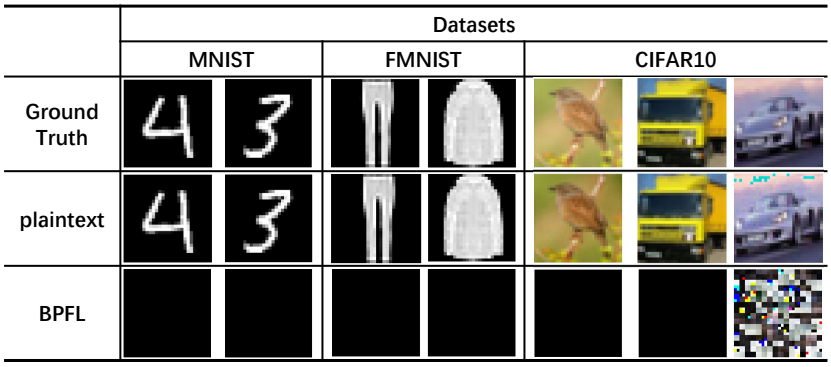


Figure 14: Recovered images by the R-GAP attack against the plaintext and the privacy-preserving BPFL.

D Additional Results

Detailed image recovery process The detailed recovery process and recovered images by DLG, iDLG, and R-GAP attacks are shown in Figure 12, Figure 13, and Figure 14, respectively. We can observe that the real images from the datasets can be successfully recovered based on plaintext within 100 iterations. While with BPFL, the attack cannot recover any useful information of the true images, as BPFL can theoretically protect the client models from being inferred.