FLOD: Oblivious Defender for Private Byzantine-Robust Federated Learning with Dishonest-Majority

Ye Dong[1,2], Xiaojun Chen[1,2(✉)], Kaiyun Li[1,2], Dakui Wang[1], and Shuai Zeng[1]

[1] Institute of Information Engineering,Chinese Academy of Sciences,Beijing,China
[2] School of Cyber Security, University of Chinese Academy of Sciences,Beijing,China
{dongye,chenxiaojun,likaiyun,wangdakui,zengshuai}@iie.ac.cn

Abstract. *Privacy* and *Byzantine-robustness* are two major concerns of federated learning (FL), but mitigating both threats simultaneously is highly challenging: privacy-preserving strategies prohibit access to individual model updates to avoid leakage, while Byzantine-robust methods require access for comprehensive mathematical analysis. Besides, most Byzantine-robust methods only work in the *honest-majority* setting.
We present FLOD, a novel oblivious defender for private Byzantine-robust FL in dishonest-majority setting. Basically, we propose a novel Hamming distance-based aggregation method to resist $> 1/2$ Byzantine attacks using a small *root-dataset* and *server-model* for bootstrapping trust. Furthermore, we employ two non-colluding servers and use additive homomorphic encryption (AHE) and secure two-party computation (2PC) primitives to construct efficient privacy-preserving building blocks for secure aggregation, in which we propose two novel in-depth variants of Beaver Multiplication triples (MT) to reduce the overhead of Bit to Arithmetic (Bit2A) conversion and vector weighted sum aggregation (VSWA) significantly. Experiments on real-world and synthetic datasets demonstrate our effectiveness and efficiency: (i) FLOD defeats known Byzantine attacks with a negligible effect on accuracy and convergence, (ii) achieves a reduction of $\approx 2\times$ for offline (resp. online) overhead of Bit2A and VSWA compared to ABY-AHE (resp. ABY-MT) based methods (NDSS'15), (iii) and reduces total online communication and run-time by 167-1416× and 3.1-7.4× compared to FLGUARD (Crypto Eprint 2021/025).

Keywords: Privacy-Preserving · Byzantine-Robust · Federated Learning · Dishonest-Majority

1 Introduction

Federated Learning (FL) is an emerging collaborative machine learning trend, in which the training is distributed and executed in parallel, and used in real-world applications, e.g., next word prediction [17], medical imaging [15]. More importantly, FL offers an appealing solution to privacy preservation by enabling clients to train a global model via an aggregator (a.k.a server) while keeping private data at local to avoid violating related regulations and laws [28,35].

Despite its benefits, FL has been shown to be vulnerable to Byzantine and inference attacks [26,16]. In the former, the adversary $\mathcal{A}^c$, who controls some clients, aims to manipulate the global model, e.g., compromise the model performance. In the latter, adversary $\mathcal{A}^s$, who corrupts the aggregator, follows the execution transcript honestly but tries to learn additional information about the clients' private local data by analyzing their model updates. Mitigating both kinds of attacks simultaneously is highly challenging: Defending Byzantine attacks require access to the clients' model updates [1,6,25,38], while privacy-preserving strategies, such as the methods based on secure computation which is provable security, prohibit this to avoid information leakage [8,12,31,33]. Additionally, existing Byzantine-robust methods mainly rely on clients *honest-majority* assumption, which means $\mathcal{A}^c$ controls $<$ 1=2 clients.

Recently, Cao *et al.* proposed FLTrust [11] to overcome the honest-majority limitation by using a small clean *root-dataset* to compute server-model update for bootstrapping trust. But it requires complex calculation operations such as cosine distance, rescaling, and comparison, which are very expensive when evaluated in secure computation.

Nguyen *et al.* employed two servers P_0 and P_1 as aggregators and proposed FLGUARD [27] to preserve privacy while thwarting Byzantine attacks by combining a novel robust aggregation approach with 2PC, but it only works under honest-majority assumption. Worsely, FLGUARD results in a severe P-P overhead and increases the client-aggregator (C-P) communication by 3 , which is also a serious burden since the C-P connection is usually in WAN and bandwidth limited. In terms of reducing C-P communication, quantization is one promising approach that approximates float model updates with low-bit precision. Among these researches, SIGNSGD [5] transmits only the sgn model update (1 for negative and 1 otherwise), and a majority vote decides the global update. However, SIGNSGD is also suffering from inference and Byzantine attacks (in dishonest-majority). What is worse, it introduces significant convergence degradation compared to traditional FL (cf. §5.1). Therefore, it is urgent and challenging to propose a solution to adequately tackle these obstacles simultaneously.

To address the challenge, we propose FLOD, a novel oblivious defender for private Byzantine-robust FL in a dishonest-majority setting based on bootstrapping trust and sgn quantization techniques. Though bootstrapping trust and quantization are inspired by prior works, our technical innovation lies in the novel in-depth aggregation method. Unlike existing works, our key insight is that Hamming distance [9] is highly more suitable and efficient than others to measure the similarity of binary vectors. Therefore, we aim to compute the Hamming distance between each local model update and the aggregator's to measure the similarity and remove local model updates with relatively a large hamming distance (small similarity). To this end, we introduce a sgn=Boolean conversion method to support XOR and propose to use ReLU with a pre-defined threshold for clipping the distance. The aggregated model update is the weighted average of all sgn model updates based on corresponding clipped results.

In terms of privacy preservation, we employ two non-colluding servers as aggregators and design efficient privacy-preserving blocks based on 2PC primitives to protect the immediate results. Concretely, we construct CXOR and PCBit2A to evaluate correlated XOR for free and realize Bit2A conversion efficiently. Hence, we can compute the Hamming distance privately. Then, we use garbled circuits (GC) with *single instruction multiple data* (SIMD) optimization to implement private -Clipping. Finally, we compute the weighted average of all secret-shared sgn model updates as CSWA securely. Specially, we propose two variants of MT [13] based on the correlations of secret-shared values to reduce the offline and online overheads of Bit2A and VSWA by 2 , respectively. Notably, these secure blocks are of independent interest and can be useful in other works.

Contributions In brief, we summarize our main contributions as follows.

- Byzantine-robustness: We propose FLOD, a novel Hamming distance-based aggregation method in the dishonest-majority setting. In contrast to existing works, FLOD is much more efficient, especially evaluated in 2PC, since our solution is mainly composed of lightweight operations, e.g., XOR, ADD, and MUL. Moreover, we achieve the same level of Byzantine-robustness as FLTrust for sgn model updates in theory.
- Privacy Preservation: To impede inference attacks by a semi-honest aggregator, we construct privacy-preserving building blocks based on 2PC and AHE for each component of FLOD. Furthermore, we propose two novel variants of MTs based on the correlations of secret-shared values to reduce the overhead (including communication and run-time) of Bit2A and VSWA by 2 . We also give a detailed analysis of the correctness and privacy of FLOD.
- Evaluations: We implement a proof-of-concept prototype and give the experimental results on neural networks: (i) FLOD defeats known Byzantine attacks with a negligible effect on accuracy and convergence, (ii) achieves a reduction of 2 for offline (resp. online) overhead compared to ABY-AHE (resp. ABY-MT) based method [13], (iii) and reduces total online communication and run-time by 167-1416 and 3.1-7.4 compared to FLGUARD [27].

Roadmap We present the preliminaries and definitions in §2. Then, we formulate our scope and threat model in §3. In §4, we give the concrete design of FLOD, including the proposed aggregation rule (cf. §4.1) and privacy-preserving building blocks (cf. §4.2). The prototype and experimental results are presented in §5. We discuss existing works in §6 and conclude this work in §7.

2 Background & Preliminaries

2.1 Federated Learning

Work ow Federated learning (FL) [19,20,22] enables K distributed clients to collaboratively build a global model W. In each training round, client C_i locally computes model updates w_i based on previous global model W and local dataset D_i, and sends w_i to the aggregator. Then, the aggregator aggregates all w_i as w according to the particular aggregation method, such as average $\mathsf{w} = \frac{1}{n} \sum_{i=1}^{K} \mathsf{w}_i$. Finally, the aggregator dispatches w to all clients for model update.

Inference Attack FL provides *data locality* such that D_i can be confined within its owner C_i. Despite this, clients still entail sharing locally trained $\mathbf{w}_i$, in order to synthesize the final global model $\mathbf{W}$. However, these local $\mathbf{w}_i$ are subject to information leakage. Specially, one semi-honest aggregator A^s can attempt to infer sensitive information and even restore private D_i from received $\mathbf{w}_i$ [39]. Therefore, it is an essential requirement to keep $\mathbf{w}_i$ confidential.

Byzantine Attacks In Byzantine attacks, adversary A^c controls some clients and mantipluates $\mathbf{w}_i^0$ to affect the final $\mathbf{W}$'s behavior. As the main accuracy (MA) is one of the dominant metrics in machine learning, we focus on the Byzantine attacks aiming at compromising MA in this work. To attack FL, existing work mainly uses *data poisoning* [38] and *model poisoning* [6] attacks. In the former, A^c poisons the instances of training data, e.g., *label flipping*. While in the latter, A^c can add well manipulated noises, e.g., *Gaussian noises*, to $\mathbf{w}_i$.

Hamming distance For two bit vectors $\mathbf{x}$, $\mathbf{y} \in \{0,1\}^d$ of equal length, their Hamming distance hd is the number of positions where $x_i \neq y_i$ for $x_i \in \mathbf{x}$ and $y_i \in \mathbf{y}$. Formally, $hd = \sum_{i=1}^{d} x_i \oplus y_i$.

2.2 Cryptographic Preliminaries

Secure 2-party computation (2PC) allows two parties to jointly compute a function without leaking private inputs. Basically, there are three techniques: Arithmetic sharing, Boolean sharing, and Yao's garbled circuits.

Arithmetic/Boolean Sharing For one ℓ-bit value x in finite ring R, party P_t for $t \in \{0,1\}$ holds an additive share $\langle x \rangle_t^{\mathsf{A}}$ such that $x = \langle x \rangle_0^{\mathsf{A}} + \langle x \rangle_1^{\mathsf{A}}$. For two arithmetic shared value $\langle x \rangle^{\mathsf{A}}$ and $\langle y \rangle^{\mathsf{A}}$, addition (ADD) can be evaluated locally. And multiplication (MUL) gate relies on Beaver's Multiplication Triples (MTs): P_0 and P_1 prepare triple $(\langle x \rangle^{\mathsf{A}}, \langle y \rangle^{\mathsf{A}}, \langle z \rangle^{\mathsf{A}})$ where $z = xy$ and P_t holds the t-th share. Then P_t computes $\langle e \rangle_t^{\mathsf{A}} = \langle x \rangle_t^{\mathsf{A}} - \langle a \rangle_t^{\mathsf{A}}$ and $\langle f \rangle_t^{\mathsf{A}} = \langle y \rangle_t^{\mathsf{A}} - \langle b \rangle_t^{\mathsf{A}}$. Both parties reconstruct e and f, and compute $\langle ab \rangle_t^{\mathsf{A}} = -tef + f\langle a \rangle_t^{\mathsf{A}} + e\langle b \rangle_t^{\mathsf{A}} + \langle z \rangle_t^{\mathsf{A}}$ [3]. Note we omit the modular operation for brevity. The triples can be generated offline using Additive Homomorphic Encrytion (AHE) or Oblivious Transfer (OT) as [13]. *Boolean Sharing* can be seen as arithmetic sharing in $\mathbb{Z}_2$, and hence all operations carry over: addition is replaced by XOR ($\oplus$), and multiplication is replaced by AND ($\wedge$).

Yao's Garbled Circuits (GC) is run between two parties called *garbler* and *evaluator*. The garbler generates the garbled circuits corresponding to the Boolean circuit by associating two random keys K_0^w, K_1^w for each wire w to represent bit value $\{0,1\}$, and then sends GC together with the keys for its inputs to the evaluator. While the evaluator obliviously obtains keys for its inputs via Oblivious Transfer (OT) [30], it evaluates the circuit to obtain the output key, which is used to decode the real output. For more details, please refer to [4,37]

Additive Homomorphic Encryption (AHE) A public key encryption scheme is additively homomorphic if given two ciphertexts $\llbracket x \rrbracket = \mathsf{AHE.Enc}_{\mathrm{pk}}(x)$ and $\llbracket y \rrbracket = \mathsf{AHE.Enc}_{\mathrm{pk}}(y)$, there is a public-key operation $\oplus$ such that $\mathsf{AHE.Enc}_{\mathrm{pk}}(x+y) = \llbracket x \rrbracket \oplus \llbracket y \rrbracket$, e.g., Paillier's encryption [29], exponential ElGamal encryption [14]. Be-

sides, adding or multiplying a ciphertext by a constant c is also efficiently supported: $\mathsf{AHE.Enc}_{\mathrm{pk}}(c + x) = c$ [illegible] and $\mathsf{AHE.Enc}_{\mathrm{pk}}(c \ x) = c$ [illegible]. Furthermore, we use *single instruction multiple data* (SIMD) technique [34] to pack multiple messages into one ciphertext for better efficiency.

Semi-honest Model A semi-honest adversary runs the protocol honestly but try to learn additional information from received messages. Let be a two-party protocol running in real-world and F be the ideal functionality completed by a trusted party. The ideal-world adversary is referred to as a simulator Sim. We define the two interactions as follows:

$\mathsf{Real}_{\pi}(\ ; C; x_1; x_2)$ run protocol with security , where P_t inputs x_t and C is the corrupted party.
Output $\{View_t; t \in C\}$, $(y_1; y_2)$. The P_t's view and output are $View_t$ and y_t.
$\mathsf{Ideal}_{F,\mathsf{Sim}}(\ ; C; x_1; x_2)$ compute $(y_1; y_2)$ $F(x_1; x_2)$.
Output $\mathsf{Sim}(C; \{x_i; y_i\}_{i \in C})$ and $(y_1; y_2)$

In the semi-honest model, a protocol is secure as long as the ideal-world adversary's view is indistinguishable from the view in the ideal-world.

3 Scope & Threat Model

We focus on the widely deployed horizontal scenario where the data is independent and identically distributed (*i.i.d.*) among clients, e.g., financial institutions or medical centers. Our goal is to preserve Byzantine-robustness and privacy at the same time in FL.

Additionally, we follow previous works [27] and consider two kinds of adversaries: A^c and A^s, as follows:

Adversary A^c controls K' (K 2) (if $K' = K$ 1, the aggregated result is likely to be the honest model update) clients and tries to compromise the global model performance. However, A^c has no control over the aggregators and honest clients. Note that A^c is not involved in the computation between the servers and only receives the aggregated results in each update. Thus, it learns nothing beyond what can be inferred from the aggregated results and its own inputs.
The second adversary A^s, which runs our protocols honestly, has access to no more than one server (two non-colluding servers) and does not perform Byzantine attacks (semi-honest). The non-colluding assumption can be guaranteed between two competing companies as it is in their interest to not give their customer's data to the competitor for protecting business secrets. Additionally, it is reasonable to assume that the two servers are semi-honest because cloud providers/companies are strictly regulated and threatened with severe financial and reputation damages once malicious behavior is detected.

4 Design of FLOD

FLOD follows the workflow of typical FL except maintaining a root-dataset and server-model as FLTrust [11], we hence elaborate our aggregation method, privacy-preserving building blocks, and the analysis of correctness and privacy.

4.1 Aggregation Method

In each round of model update, client C_i (resp. aggregator) computes the sgn model update $\tilde{w}_i$ (resp. $\tilde{w}_s$) in $\{-1, 1\}$ as SIGNSGD[5]. The sgn value can limit scaling attacks [2], but naively adding them up will ignore their direction property. Specially, an attacker can manipulate the direction of $\tilde{w}_i$ such that the global model might be updated towards the opposite of correct direction. FLTrust resolved a similar issue using cosine similarity and ReLU clipping [11], but it requires many expensive non-linear operations. When combining them with secure computation, the overhead will be much more serious.

Fig. 1: Overview of FLOD aggregation method.

To this end, we propose an efficient aggregation method as Fig. 1. First, we introduce sgn/Boolean conversion so that C_i (resp. aggregator) can convert the $\tilde{w}_i$ (resp. $\tilde{w}_s$) in $\{-1, 1\}$ to Boolean representation w_i (resp. w_s) and back. Then, aggregator computes their Hamming distance with mostly XOR/ADD and and -clip the distance with little comparison, where is the threshold. Finally, aggregator converts w_i back to $\tilde{w}_i$ and aggregates them based on clipping results.

sgn/Boolean Conversion Recall $\tilde{w}_i$ is in $\{-1, 1\}^d$, it is not suitable for XOR operation. Therefore, for $\tilde{w}_{ij} \in \tilde{w}_i$ we propose encoding method E as:

$$E(\tilde{w}_{ij}) = \begin{cases} 0, & \text{if } \tilde{w}_{ij} = 1; \\ 1, & \text{otherwise.} \end{cases} \tag{1}$$

And to aggregate $\tilde{w}_i$ in the last step of aggregation, we need to decode $E(\tilde{w}_{ij})$. To this end, we propose the decoding method D as:

$$D(E(\tilde{w}_{ij})) = 1 - 2E(\tilde{w}_{ij}) \tag{2}$$

It is straightforward to see that we can guarantee $D(E(\tilde{w}_{ij})) = \tilde{w}_{ij}$.

HD-Computing After computing and E-encoding sgn model update, we have the Boolean representation in Z_2^d. Now, for each C_i, we let the aggregator compute the Hamming distance between w_i and w_s as:

$$hd_i = \sum_{j=1}^{d} w_{ij} - w_{sj}. \tag{3}$$

-Clipping We let the aggregator use threshold to clip hd_i and assign weight $_i$ to w_i as:

$$_i = \mathrm{ReLU}(- hd_i); \tag{4}$$

Algorithm 1 FLOD Aggregation Method

Input: C_i inputs $\{w_i$ for $i \in [1, K]$ and aggregator inputs $\{w_s$.
Output: $w = \frac{1}{\sum_{i=1}^{K} \pi_i} (\sum_{i=1}^{K} \pi_i \{w_i)$.

1: Encode $\{w_s \in \{-1, 1\}^d$ using E into $\{0, 1\}^d$ as $w_s = E(\{w_s)$.
2: **for** $i \in [1, K]$ **do**
3: Encode $\{w_i \in \{-1, 1\}^d$ using E into $\{0, 1\}^d$ as $w_i = E(\{w_i)$.
4: Compute $hd_i = \sum_{j=1}^{d} w_{ij} \oplus w_{sj}$ for $w_{ij} \in w_i$ and $w_{sj} \in w_s$.
5: Clip hd_i as $\pi_i = \text{ReLU}(\quad - hd_i)$.
6: **end for**
7: **return** $w = \frac{1}{\sum_{i=1}^{K} \pi_i} (\sum_{i=1}^{K} \pi_i \{w_i)$, where $\{w_i = D(w_i)$.

where ReLU(x) returns x if $x > 0$, and 0 otherwise. Therefore, if $hd_i > \quad$, which indicates the difference between w_i and w_s is huge, ReLU will return 0 and we will exclude w_i from the aggregation. Otherwise, we assign positive weight $\quad - hd_i$ to w_i. Hence, when hd_i is smaller, which means w_i is more similar to w_s, the corresponding weight π_i is bigger.

Weighted Averaging Finally, we compute $w = \frac{1}{\sum_{i=1}^{K} v_i} (\sum_{i=1}^{K} \pi_i \{w_i)$ as the aggregated result to update the global model. The formulation is in algorithm 1, and the analysis of our Byzantine-robustness is illustrated in Appendix A.

4.2 Privacy-Preserving Building Blocks

Fig. 2: Workflow of privacy-preserving FLOD. Without losing generality, we let P_0 maintain root-dataset and server-model. As E can be evaluated locally by each party and secure D is implemented in CSWA, we omit them for brevity.

We construct the privacy-preserving blocks of FLOD as Fig. 2: We employ two non-colluding servers, P_0 and P_1, as aggregators. In each round of aggregation, P_0 computes and E encodes w_s, and C_i sends the Boolean share of E-encoded w_i to P_t. P_0 and P_1 firstly compute the coordinate-wise XOR of w_i and w_s privately using CXOR. Then, they jointly convert the Boolean shares to arithmetic shares using PCBit2A for efficient weighted averaging aggregation. Next, P_t sum up all coordinates of $\langle hd_i \rangle^A$ as $\langle hd_i \rangle^A$, and jointly clip it with $\quad$ to obtain π_i as private $\quad$-Clipping. Finally, two servers aggregate $\{w_i$s based on π_i as w in CSWA, where we compute the arithmetic shares of $\{w_i$ through the shares of w_i. And w is revealed to P_0 and all clients for model update.

Algorithm 2 PCBit2A

Input: For $t \in \{0,1\}$, P_t inputs $\langle w_i \rangle_t^B$ and $\langle hd_i \rangle_t^B$ for $i \in \{1,\dots,K\}$.
Output: For $t \in \{0,1\}$, P_t outputs $\langle w_i \rangle_t^A$ and $\langle hd_i \rangle_t^A$ for $i \in \{1,\dots,K\}$.

1: for $i = 1$ to K do
2: Offline:
3: P_0 samples lengthd vectors x_i, x_i', r_i, and r_i' at random.
4: P_1 samples lengthd vectors y_i at random.
5: P_1 encrypts and sends $y_0 = \mathrm{AHE.Enc}_{pk_1}(y_i)$ to P_0.
6: P_0 computes and sends $s_i = (x_i \quad y_0) \quad r_i$, $s_i' = (x_i' \quad y_0) \quad r_i'$ to P_1.
7: P_1 sets $\langle z_i \rangle_1^A = \mathrm{AHE.Dec}_{sk_1}(s_i)$, $\langle z_i' \rangle_1^A = \mathrm{AHE.Dec}_{sk_1}(s_i')$. P_0 sets $\langle z_i \rangle_0^A = r_i$, $\langle z_i' \rangle_0^A = r_i'$.
8: Online:
9: P_0 computes and sends $\langle w_i \rangle_0^B + x_i$ and $\langle hd_i \rangle_0^B + x_i'$ to P_1.
10: P_1 computes and sends $\langle w_i \rangle_1^B + y_i$ to P_0.
11: P_0 locally computes $\langle u_i \rangle_0^A = x_i(\langle w_i \rangle_1^B + y_i) + \langle z_i \rangle_0^A$ and $\langle u_i' \rangle_0^A = x_i'(\langle hd_i \rangle_1^B + y_i) + \langle z_i' \rangle_0^A$, and P_1 sets $\langle u_i \rangle_1^A = \langle w_i \rangle_1^B(\langle w_i \rangle_0^B + x_i) + \langle z_i \rangle_1^A$ and $\langle u_i' \rangle_1^A = \langle hd_i \rangle_1^B(\langle hd_i \rangle_0^B + x_i') + \langle z_i' \rangle_1^A$.
12: P_t computes $\langle w_i \rangle_t^A = \langle w_i \rangle_t^B \quad 2\langle u_i \rangle_t^A$ and $\langle hd_i \rangle_t^A = \langle hd_i \rangle_t^B \quad 2\langle u_i' \rangle_t^A$.
13: end for
14: return For $t \in \{0,1\}$, P_t outputs $\langle w_i \rangle_t^A$ and $\langle hd_i \rangle_t^A$ for $i \in \{1,\dots,K\}$.

Correlated XOR (CXOR) After model evaluation and E, P_0 has $w_s \in \{0,1\}^d$, P_t has $\langle w_i \rangle_t^B$ for $t \in \{0,1\}$. To compute the Hamming distance between w_i and w_s, we need to compute their coordinate-wise XOR rstly. As $w_i = \langle w_i \rangle_0^B \quad \langle w_i \rangle_1^B$, we have $hd_i = w_s \quad w_i = w_s \quad \langle w_i \rangle_0^B \quad \langle w_i \rangle_1^B$. Therefore, we let P_0 computes $\langle hd_i \rangle_0^B = w_s \quad \langle w_i \rangle_0^B$ and $\langle hd_i \rangle_1^B = \langle w_i \rangle_1^B$ with no communication.

Partial Correlated Bit to Arithmetic Conversion (PCBit2A) To achieve practical aggregation, we need to convert $\langle w_i \rangle^B$ and $\langle hd_i \rangle^B$ to arithmetic shares since the latter is more e cient for ADD and MUL. Considering $w_{ij} \in w_i$ is 0 or 1, we have $w_{ij} = \langle w_{ij} \rangle_0^B + \langle w_{ij} \rangle_1^B \quad 2\langle w_{ij} \rangle_0^B \langle w_{ij} \rangle_1^B$. As P_t has $\langle w_{ij} \rangle_t^B$, the main challenge is computing the arithmetic shares of $\langle w_{ij} \rangle_0^B \langle w_{ij} \rangle_1^B$ securely.

To this end, a naive approach is generating MT using AHE as ABY library [13] and computing the product in element-wise as §2.2, where we view $\{\langle a_i \rangle_0^A = \langle w_i \rangle_0^B; \langle a_i \rangle_1^A = 0\}$, $\{\langle b_i \rangle_0^A = 0; \langle b_i \rangle_1^A = \langle w_i \rangle_1^B\}$, and compute $\langle a_i b_i \rangle^A = \langle w_i \rangle_0^B \langle w_i \rangle_1^B$. But this method requires P_1 encrypts $4K$ length-d vectors and sends $4K$ dd=se ciphertexts to P_0 in o ine phase, and P_t sends $4Kd\ell$ bits to $P_{1\ t}$ in online phase for all $\langle w_i \rangle^B$ and $\langle hd_i \rangle^B$ conversions totally. To improve e ciency, we propose a novel partial correlated Bit2A (PCBit2A) method requires 4 less encryption and 2 less communication in o ine, and reduces $P_0 \ P_1$ (resp. $P_1 \ P_0$) communication by 2 (resp. 4) in online.

Firstly, as P_t holds $\langle w_i \rangle_t^B$ for $\langle w_i \rangle_0^B \langle w_i \rangle_1^B$, we propose a variant of MT (partial MT) $(x_i; y_i; z_i)$ inspired by [7] such that P_0 holds $\{x_i; \langle z_i \rangle_0^A\}$, P_1 holds $\{y_i; \langle z_i \rangle_1^A\}$, and $z_i = x_i y_i$. In online, P_0 computes and sends $\langle w_i \rangle_0^B + x_i$ to P_1,

Fig. 3: Boolean circuit for private -Clipping. "+" refers to integer addition and " " refers to integer subtraction. The "MUX" is multiplexer and ">" outputs 1 i.f.f. the input is larger than bR=2c.

while P_1 sends $\langle w_i \rangle_1^B + y_i$ to P_0. P_0 computes $\langle u_i \rangle_0^A = x_i(\langle w_i \rangle_1^B + y_i) + \langle z_i \rangle_0^A$ and P_1 computes $\langle u_i \rangle_1^A = \langle w_i \rangle_1^B(\langle w_i \rangle_0^B + x_i) + \langle z_i \rangle_1^A$ to obtain $\langle u_i \rangle_0^A + \langle u_i \rangle_1^A = \langle w_i \rangle_0^B \langle w_i \rangle_1^B$ in secret. And $\langle hd_i \rangle_0^B \langle hd_i \rangle_1^B$ computation is likewise.

The above method completes $\langle w_i \rangle^B$ and $\langle hd_i \rangle^B$ conversions independently. As $\langle w_i \rangle_1^B = \langle hd_i \rangle_1^B$, we further let P_1 prepare $y_i = y_i'$, while P_0 generate x_i, x_i', r_i, and r_i' independently at random. In online, P_1 only encrypts and sends y_i to P_0. While P_0 uses $(x_i; y_i; r_i)$ (resp. $x_i'; y_i'; r_i'$)) to compute $_i$ (resp. $_i'$). And P_t computes $\langle z_i \rangle_t^A$ and $\langle z_i' \rangle_t^A$ locally as the oine in Algorithm 2. Note that we can also generate the MTs using OT with much more communication and less run-time. As the network costs are charged much more than computation [18], we adopt AHE based method to minimize the monetary cost. In online, P_0 sends $\langle w_i \rangle_0^B + x_i$ and $\langle hd_i \rangle_0^B + x_i'$ to P_1, while P_1 only sends $\langle w_i \rangle_1^B + y_i$ to P_0. Thus P_t can complete $\langle w_i \rangle^B$ and $\langle hd_i \rangle^B$ conversions simultaneously as illustrated in online of Algorithm 2. Therefore, we need $2Kd\ell$ (resp. $Kd\ell$) bits for $P_0 \to P_1$ (resp. $P_1 \to P_0$) online communication.

Private -Clipping To evaluate -Clipping privately, we use GC to clip $\langle hd_i \rangle^A = \sum_{j=1}^{d} \langle hd_{ij} \rangle^A$. The Boolean circuit is illustrated in Fig. 3: P_0 inputs $x_0 = \langle hd_i \rangle_0^A$ and $y_0 = r$ (chosen at random), P_1 inputs $x_1 = \langle hd_i \rangle_1^A$. The first block computes the arithmetic sum of $hd_i = x_0 + x_1$ over integers. The second block computes the ReLU function. And the third block subtract y_0 from the result to obtain the P_1's share y_1. Finally, P_t sets $\langle \ \rangle_t^A = y_t$.

Correlated Secure Weighted Aggregation(CSWA) To weighted average all w_i privately, we first let P_t computes $\langle w_i \rangle_t^A = t \cdot 2\langle w_i \rangle_t^A$ to implement D securely in batch. With $\langle \ \rangle^A$ and $\langle w_i \rangle^A$, P_0 and P_1 are capable to compute $\langle w \rangle^A$ where the main challenge is computing $\langle \ w_i \rangle^A$ for $i \in \{1, \ldots, K\}$. A trivial method is using MT for each scalar-product as ABY [13], but this needs Kd triples in total, which requires P_t to encrypt (and send) $2Kd$ ciphertexts to P_{1-t} in oine, and $2Kd\ell$ bits online communication for each P_t in online. Hence, we propose an efficient method which achieves 2 reduction in communication.

Our key insight is that $_i$ is fixed for all coordinates of w_i. Therefore, we construct MT $(x_i; y_i; z_i)$ subject to $z_i = x_i y_i$ using AHE, where P_t has $(\langle x_i \rangle_t^A; \langle y_i \rangle_t^A; \langle z_i \rangle_t^A)$. P_t thus only sends $1 + Kd$ ciphertexts to P_{1-t} in oine. Note that we let P_t duplicate $\langle x_i \rangle_t^A$ into s copies and pack these copies into one ciphertext to support SIMD technique, and we use denotation $\langle \llbracket x_i \rrbracket \rangle_t^A = \mathrm{AHE.Enc}_{pk_t}(\langle x_i \rangle_t^A)$ for brevity. In online, P_t sends $\langle e_i \rangle_t^A = \langle \ _i \rangle_t^A - \langle x_i \rangle_t^A$, $\langle f_i \rangle_t^A =$

Algorithm 3 CSWA

Input: For $t \in \{0,1\}$, P_t inputs $\langle w_i \rangle_t^A$ and $\langle \_ \rangle_t^A$ for $i \in \{1,\ldots,K\}$.
Output: P_0 outputs $w = \frac{1}{\sum_{i=1}^{K} \__i} \left(\sum_{i=1}^{K} \__i w_i\right)$.

1: for $i = 1$ to K do
2: Offline
3: P_t locally samples scalar $\langle x_i \rangle_t^A$, length-d vector $\langle y_i \rangle_t^A$ and $r_{i,t}$.
4: P_t encrypts and sends $[\![\langle x_i \rangle_t^A]\!] = \text{AHE.Enc}_{pk_t}(\langle x_i \rangle_t^A)$ to P_{1-t}.
5: P_t computes and sends $s_{i,t} = \langle y_i \rangle_t^A [\![\langle x_i \rangle_t^A]\!] - r_{i,t}$ to P_{1-t}.
6: P_t decrypts and sets $\langle z_i \rangle_t^A = \langle x_i \rangle_t^A \langle y_i \rangle_t^A + \text{AHE.Dec}_{sk_t}(\__{i,1-t}) + r_{i,t}$.
7: Online
8: P_t computes $\langle \hat{w}_i \rangle_t^A = t \cdot 2\langle w_i \rangle_t^A$ to implement $D(\langle w_i \rangle^A)$ securely.
9: P_t sends $\langle e_i \rangle_t^A = \langle \__i \rangle_t^A - \langle x_i \rangle_t^A$; if $t = 0$, $\langle f_i \rangle_0^A = \langle \hat{w}_i \rangle_t^A - \langle y_i \rangle_t^A$ to P_{1-t}.
10: P_0 & P_1 reconstruct (e_i, f_i) locally.
11: P_t locally computes $\langle \__i \hat{w}_i \rangle_t^A = t \cdot e_i f_i + e_i \langle \hat{w}_i \rangle_t^A + \langle \__i \rangle_t^A f_i + \langle z_i \rangle_t^A$.
12: end for
13: P_t computes $\sum_{i=1}^{K} \langle \__i \hat{w}_i \rangle_t^A$, $\sum_{i=1}^{K} \langle \__i \rangle_t^A$, and P_1 sends its shares to P_0.
14: return P_0 reconstructs and computes $w = \frac{1}{\sum_{i=1}^{K} \__i} \left(\sum_{i=1}^{K} \__i w_i\right)$.

$\langle \hat{w}_i \rangle_t^A - \langle y_i \rangle_t^A$ to P_{1-t}, reconstruct both, and computes $\langle \__i \hat{w}_i \rangle_t^A = t \cdot e_i f_i + e_i \langle \hat{w}_i \rangle_t^A + \langle \__i \rangle_t^A f_i + \langle z_i \rangle_t^A$. Finally, P_t adds up $\sum_{i=1}^{K} \langle \__i \hat{w}_i \rangle_t^A$ and $\sum_{i=1}^{K} \langle \__i \rangle_t^A$, and P_1 reveals its shares to P_0, who computes $w = \frac{1}{\sum_{i=1}^{K} \__i} \left(\sum_{i=1}^{K} \__i w_i\right)$. This requires $K(d+1)\ell$ bits for P_0 and $(K+1)(d+1)\ell$ bits for P_1 in online. The details are in Algorithm 3.

4.3 Correctness & Privacy

Correctness FLOD is correct as long as the core building blocks are correct. First, it is straightforward that CXOR is correct. Then, PCBit2A conversion is correct since AHE and partial MT based secure multiplication are correct. Afterwards, GC guarantees that P_t can obtain the shares of $\__i = \text{ReLU}(\_ - hd_i)$ for w_i. Finally, AHE and MT also guarantee the correctness of CSWA.

Privacy We analyze the privacy of FLOD against semi-honest adversary A^s, who corrupts one server, in Theorem 1. The proof is illustrated in Appendix B.

Theorem 1 (Privacy of FLOD). FLOD guarantees that adversary A^s learns nothing beyond what can be inferred from the aggregated results $\left(\sum_{i=1}^{K} \__i w_i, \sum_{i=1}^{K} \__i\right)$ with $1 - \_$ probability where $\_$ is negligible, as long as there are no more than one corrupted server in semi-honest model.

Additionally, we can guarantee privacy even in the A^c-A^s collusion threat model. The reason is that A^s learns nothing no more than the aggregated results with an overwhelming probability, and A^c only receives the aggregated results (cf. §3). Therefore, they learn nothing more than what can be inferred from the aggregated results and their own inputs with an overwhelming probability.

(a) (b)

Fig. 4: MA along with the training iterations with = 0. 4(a) is for FNet on Fashion-MNIST, 4(b) is for ResNet-18 on CIFAR10.

Table 1: MA of FedAvg, Krum, Median, T-Mean, FLGUARD, FLTrust, and FLOD with = 30%. Note that we evaluate FedAvg with no Byzantine attack.

MA , =30%		FedAvg	Krum	Median	T-Mean	FLGUARD	FLTrust	FLOD
GA	FNet	0.86	0.85	0.85	0.85	0.77	0.85	0.84
	ResNet-18	0.79	0.76	0.75	0.76	0.74	0.76	0.76
LF	FNet	0.86	0.85	0.83	0.83	0.78	0.85	0.84
	ResNet-18	0.79	0.76	0.64	0.66	0.75	0.75	0.75

5 Evaluation

Evaluation Setup: We implement FLOD in C++ and Python3. We use ABY library [13] for 2PC and SIMD circuits, and rely on SEAL library [32] for AHE. Parameters for both schemes are set with 128-bit security level. And we employ widely used Convolutional Neural Networks: FNet with 507K parameters on Fashion-MNIST [36] and ResNet-18 light with 2.07M parameters on CIFAR-10 [21]. Experiments are executed on Intel(R) Xeon(R) CPU E5-2650 v3@ 2.30GHz servers with 64GB RAM, and we use PyTorch v1.4.0 equipped with CUDA v10.2 and two 12G memory TITAN Xp GPUs for model training. The P-P connection is equipped with 10 Gbps LAN with 0.2ms RTT. And the C-P connection is over 50Mbps WAN with 50ms RTT.

5.1 E ectiveness Analysis

We evaluate FLOD against state-of-the-art Byzantine attacks: Gaussian Attack (GA) and Label Flipping attack (LF). In GA, the poisoned model updates are drawn from a Gaussian distribution (model poisoning). While in LF, we replace training label y on the Byzantine machines with 9 y (data poisoning). We measure the main task Top-1 accuracy (MA) as the e ectiveness metric.

First, we measure MA with training iterations of FLOD, FedAvg [24], and SIGNSGD with $= \frac{K'}{K} = 0\%$ to show our model performance under no Byzantine attacks. Then, we x = 30% and compare the MA of FLOD to Krum [6], Median [38], T-Mean [38], FLGUARD [27], and FLTrust [11] to show our robustness in honest-majority. Finally, we measure the MA with dynamic to present our robustness is comparable to FLTrust, which is better than other works when > 50%. We train FNet for 150 iterations and ResNet-18 Light for 1,000 itera-

(a) (b)

Fig. 5: MA of FNet with = 10%-90% for all Byzantine-robust aggregation methods, where 5(a) is for GA and 5(b) is for LF.

tions, and set = $\frac{d}{3}$ and $\frac{d}{2}$ for FNet and ResNet-18 for best MA, respectively. Determining the optimal is a interesting task, and we leave it for future work.

Performance Analysis with = 0 One Byzantine-robust aggregation method should also apply to no attack cases in real applications: it should introduce little degradation to model performance when = 0. As FedAvg achieves the optimal performance, we compare FLOD to it. Also, we measure the MA of SIGNSGD since we encode model updates similar to it.

As shown in Fig 4, FLOD converges to a similar MA level as FedAvg achieved within the same training iterations for both FNet and ResNet-18 Light. The reason is that FLOD utilizes the root dataset to bootstrap trust; thus, it can almost aggregate all sgn model updates when = 0. Although the sgn encoded values lose some information, it has merely impacts on the overall model training. However, SIGNSGD converges much slower than FedAvg and ours because SIGNSGD only returns the sgn of the sum of all individual model updates to resist Byzantine attacks, which is equivalent to Median method for sign encoded model updates. As Median excludes K 1 values for each coordinate, SIGNSGD introduces signi cant degradation to model convergence.

Performance Analysis with = 30% Table 1 shows the MA of FLOD and existing methods: Krum, Median, T-Mean, FLGUARD, and FLTrust, in honest-majority (= 30%), under both attacks. As SIGNSGD converges much slower and is equivalent to Median for sign encoded model updates, we omit it here. And we present the MA of FedAvg without attack for a comprehensive comparison.

FLOD introduces little MA loss compared to FedAvg and other Byzantine-robust FL methods. Compared to FedAvg the MA loss is no more than 0:02 for FNet and 0:04 for ResNet-18. Moreover, the MA degradation is within 0:01 for all cases compared to FLTrust. The degradation has two main sources: (i) With Byzantine attacks, the correct model updates (or clean data) is less than that of FedAvg Therefore, FLOD and FLTrust with = 30% both introduce MA degradation compared to FedAvg with = 0. (ii) For FLOD, sgn encoding introduces information loss compared to original model updates. Hence, FLOD achieves a slightly lower MA than FLTrust. However, this degradation is so little that it can be acceptable in practical with enhancements on privacy preservation.

Table 2: P-P communication of PCBit2A (resp. CSWA) and ABY based methods for Bit2A (resp. VSWA) in one round of aggregation, where $K = 10; 50; 100$, models are FNet and ResNet-18, o ine (resp. online) communication is in GB (resp. MB), and X denotes AHE for o ine and MT for online.

Comm.		Bit2A				VSWA			
Model		FNet		ResNet-18		FNet		ResNet-18	
Method		PCBit2A	ABY-X	PCBit2A	ABY-X	CSWA	ABY-X	CSWA	ABY-X
O., K	10	1.43	2.90	7.44	15.28	0.74	1.45	4.03	7.64
	50	7.44	14.77	33.86	91.74	3.94	7.39	22.81	44.99
	100	14.07	30.60	67.93	153.13	7.64	15.30	38.63	76.56
On., K	10	58.16	151.22	237.17	616.64	41.07	81.14	165.83	329.58
	50	290.06	754.16	1185.06	3081.16	194.41	387.62	797.55	1594.78
	100	580.16	1508.42	2370.45	6163.17	387.69	772.38	1587.20	3173.68

Besides, compared to Krum, Median, T-Mean, and FLGUARD, we achieve a similar or better MA in honest-majority. Therefore, FLOD has much broader application prospects for resisting Byzantine attacks in FL.

Performance Analysis with Dynamic Fig. 5(a)-5(b) show the MA with the fraction of Byzantine clients for all methods. To completely test the Byzantine-robustness, we alter =10%-90%. Firstly, we see FLOD can reach a similar or even higher level of MA as other FL methods in honest-majority, which is consistent with our analysis. Secondly, when > 50% the MA of Krum, Median, and FLGUARD drops sharply, e.g. MA 0:1 when = 90%. The reason is that these methods all rely on the honest-majority assumption, and thus with > 50%, the poisoned model updates will be aggregated into the nal result. However, FLOD and FLTrust can still maintain a high MA since both methods utilize a root-dataset to bootstrap trust, and thus can exclude the poisoned model updates even in dishonest-majority. Similar results for ResNet-18 are illustrated as Fig. 8 in Appendix C.

5.2 E ciency Analysis

We test the costs and scalability of FLOD by varying the number of clients ($K = 10; 50; 100$) and size of model updates. Concretely, we measure the communication overhead and run-time in respective o ine and online phases.

Communication We test the P-P communication costs of Bit2A and VSWA for o ine, and compare our costs to ABY-AHE based method. For online, we measure each block communication and compare it to ABY-MT based method, and further compare our total online communication overhead, including the C-P overhead, of one aggregation to FLGUARD to demonstrate our improvements.

P-P O ine Communication We measure the P-P communication costs for the o ine of PCBit2A and CSWA, and compare our costs to ABY-AHE based method as the o ine part of Table 2. Firstly, our PCBit2A reduces the communication costs by 2 for Bit2A. This is because we propose partial MTs and reuse ψ_i to generate correlated partial MTs for $\langle w_i \rangle^B$ and $\langle nd_i \rangle^B$ conversions simultane-

(a) (b) (c)

Fig. 6: Total online C-P and P-P communication of FLOD and FLGUARD for FNet and ResNet-18 in one aggregation. 6(a) shows each C-P communication, 6(b) shows the P-P costs for FNet, and 6(c) shows the P-P costs for ResNet-18. Note the y-axis of 6(b) and 6(c) is in log-scale.

ously. Secondly, we also reduce the CSWA online communication costs by 2 since we reuse the same x_i^A for all coordinates of w_i in CSWA. Hence, we improve the total online communication efficiency in one aggregation by 2 compared to ABY-AHE method.

Online Communication Table 2 online part shows the online communication between P_0 and P_1 caused by PCBit2A and CSWA in one aggregation, and we compare our costs to ABY-MT based method. For PCBit2A, we reduce the communication by around 2.5 due to our partial correlated triples optimization. And for CSWA we reduce the communication by 2 as $_i$ is same for all coordinates of w_i. Additionally, the costs of PCBit2A and CSWA are determined by K and model size since P_0 and P_1 need to conduct multiplication for each coordinate of w_i. Besides, private -Clipping introduces little online communication, and thus we present it as Table 4 in Appendix D due to page limitation.

Besides, we present the total online C-P and P-P communication costs of FLOD (including private -Clipping) in one aggregation, and compare our overhead to FLGUARD as Fig 6. Firstly, Fig. 6(a) shows that we reduce the C-P communication by 6 compared to FLGUARD. This reason is that FLGUARD encodes the model updates and aggregated results as 64-bit integers, while we use 1 bit to represent each share of the E-encoded binary sgn model update and 32 bits to encode the aggregated results (which is enough to achieve a comparable accuracy as FedAvg). Secondly, in Fig. 6(b)-6(c), we compare the P-P communication in one aggregation of FLOD to FLGUARD to show our improvements: FLOD requires 361-1416 less communication for FNet and 167-417 less communication for ResNet-18. The reason is that our methods are mainly composed of arithmetic operations and require a little GC for private -Clipping. While FLGUARD requires much tremendous expensive garbled circuits for cosine distance calculation, clustering, and Euclidean distance calculation/clipping/aggregation. Thirdly, with K being increased, the communication of FLGUARD increases more sharply than FLOD. Therefore, our FLOD is much more communication efficient than FLGUARD.

Run-time We test the run-time, including the computation, data transferring, and network latency, for online and online phases. Also, we compare our online run-time to ABY-AHE based method, and online run-time to ABY-MT based method and FLGUARD in one aggregation.

Table 3: Run-time in seconds of PCBit2A (resp. CSWA) and ABY based methods for Bit2A (resp. VSWA) in one aggregation for FNet and ResNet-18, where $K = 10, 50, 100$ and X denotes AHE for offline and MT for online.

Run-time		Bit2A				VSWA			
Model		FNet		ResNet-18		FNet		ResNet-18	
Method		PCBit2A	ABY-X	PCBit2A	ABY-X	CSWA	ABY-X	CSWA	ABY-X
Offline, K	10	22.82	50.83	113.47	241.28	16.56	25.42	78.57	120.64
	50	114.36	241.28	582.13	1194.15	80.39	121.76	386.87	558.65
	100	222.54	479.47	1151.29	2381.73	157.71	289.78	766.14	1696.35
Online, K	10	9.34	23.35	22.42	56.05	5.16	5.22	20.75	21.84
	50	45.52	113.76	106.43	266.08	23.43	25.66	96.33	98.15
	100	94.19	235.98	213.50	533.75	48.36	49.78	192.66	193.75

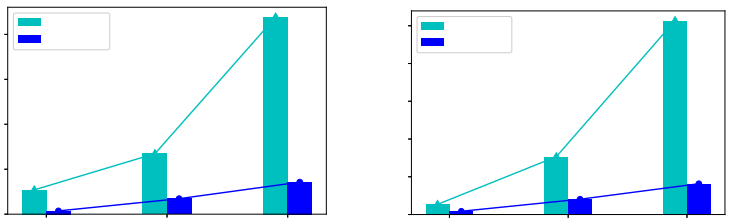
Offline Run-time Table 3 offline part shows the offline run-time of PCBit2A, CSWA and ABY-AHE based method for Bit2A and VSWA. Compared to ABY-AHE based method, our approaches reduce the run-time by around 2 and 1.5. The reason is that we propose partial MTs and utilize the correlations for $(\langle w_i \rangle^B, \langle d_i \rangle^B)$ in PCBit2A, and use $\langle x_i \rangle^A$ for all $w_{ij} \in w_i$ in CSWA, and hence we reduce the total instances of AHE operations by 2 for Bit2A and by 1.5 for VSWA. Moreover, it reduces the run-time of data transfer, which is consistent with offline communication analysis.

Online Run-time Table 3 online part presents the online run-time per aggregation of PCBit2A and CSWA, and we compare our costs to ABY-MT based method. As can be seen, we reduce the run-time of Bit2A by around 2.5 due to our reduction on the numbers of multiplication and communication. However, the online run-time improvements of CSWA is limited. The reasons are as follows: (i) We do not reduce the multiplication invocations, and the efficiency of scaler-vector and vector-vector element-wise multiplication is almost the same in batch processing; (ii) The reduction of communication saves little time in our LAN P_0-P_1 network setting. CXOR and private -Clipping introduce little overhead due to the efficient XOR operation and model update size independent GC invocations, respectively. We thus present their costs in Appendix D.

Fig. 7 presents the total online run-time of FLOD (including CXOR and private -Clipping) and FLGUARD in one aggregation. As the ML training is executed in plaintext and can be significantly accelerated using GPU, we omit its overhead as FLGUARD. As illustrated in the expermental results, we reduce the online run-time significantly compared to FLGUARD. (i) FLOD reduces the run-time by 3.9-7.4 for FNet and 3.1-6.2 for ResNet-18. (ii) Besides, we also observe that with K being increased, the run-time of FLGUARD increases much more significantly than FLOD. Therefore, our methods are more practical in real applications.

6 Related Works

Here, we review the work in the area of privacy preservation and Byzantine-robustness of FL. In terms of privacy preservation, the main used technologies are differential privacy (DP), (additively) homomorphic encryption ((A)HE), secret



(a) (b)

Fig. 7: Total online run-time of FLOD *and* FLGUARD *for FNet and ResNet-18 in one aggregation. 7(a) is for FNet and 7(b) is for ResNet-18.*

sharing, and etc. Shokri *et al.* used DP to protect model update to achieve the balance between privacy and accuracy [33]. Liu *et al.* further combined the local DP with Top-k gradients selection to improve the performance [23]. Phong *et al.* proposed to protect the clients' gradients utilizing AHE [31]. Bonawitz *et al.* designed a secure aggregation scheme for sum function by exploiting secret sharing and key agreement protocol [8]. Besides, Gibbs *et al.* combined secret sharing with Zero-Knowledge Proof to verify the validity of clients' gradients [12]. But these schemes all focus on simple linear aggregation, e.g., average.

Meanwhile, Byzantine-robust aggregation rules have been extensively studied using clear gradients. Among these methods, the main mechanism is to compare gradients received and remove the outliers. Blanchard *et al.* proposed Krum combining the intuitions of *majority*-based and *squared-distance*-based methods to guarantee convergence tolerating K^{0} $bK=2c$ 1 adversaries [6]. Mhamdi *et al.* showed that convergence is not enough and introduced Bulyan to reduce the attacker's leeway to narrow $O(1=\overline{p}\overline{d})$ bound [25]. Alistarh *et al.* proposed a variant of SGD which finds $"$-approximate minimizers of convex functions in $\Theta(\frac{1}{\varepsilon^2 m+\alpha^2/\varepsilon^2})$ iterations [1]. Yin *et al.* developed trimmed mean and median based robust distributed learning algorithms with a focus on optimal statistical performance [38]. Bernstein *et al.* proposed SIGNSGD where clients transmit only the sign of their gradient vector to a server, and the overall update is decided by a majority vote [5]. But these method all depends on the clients honest-majority assumption. Cao *et al.* proposed FLTrust to break this limitation by collecting a small clean dataset on the aggregation server to bootstrap trust, but this method also provides no privacy guarantee for clients [11].

To our best knowledge, only Nguyen *et al.* proposed a similar work, FLGUARD [11], as ours. But they requires much more significant overhead, including computation and communication, which limits their real application seriously.

7 Conclusion

We propose FLOD, an efficient oblivious defender for private Byzantine-robust FL in dishonest-majority. We introduce a Hamming distance-based aggregation method and then use 2PC and AHE based protocols, with several novel in-depth optimizations, to protect privacy. Evaluations show our effectiveness and efficiency. We aim to verify the correctness of the aggregated results in future.

References

1. Alistarh, D., Allen-Zhu, Z., Li, J.: Byzantine stochastic gradient descent. arXiv preprint arXiv:1803.08917 (2018)
2. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: International Conference on Artificial Intelligence and Statistics. pp. 2938–2948. PMLR (2020)
3. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Annual International Cryptology Conference. pp. 420–432. Springer (1991)
4. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 784–796 (2012)
5. Bernstein, J., Wang, Y.X., Azizzadenesheli, K., Anandkumar, A.: signsgd: Compressed optimisation for non-convex problems. In: International Conference on Machine Learning. pp. 560–569. PMLR (2018)
6. Blanchard, P., El Mhamdi, E.M., Guerraoui, R., Stainer, J.: Machine learning with adversaries: Byzantine tolerant gradient descent. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 118–128 (2017)
7. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. In: European Symposium on Research in Computer Security. pp. 192–206. Springer (2008)
8. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1175–1191. ACM (2017). https://doi.org/10.1145/3133956.3133982
9. Bookstein, A., Kulyukin, V.A., Raita, T.: Generalized hamming distance. Information Retrieval **5**(4), 353–375 (2002)
10. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)
11. Cao, X., Fang, M., Liu, J., Gong, N.Z.: Fltrust: Byzantine-robust federated learning via trust bootstrapping. arXiv preprint arXiv:2012.13995 (2020)
12. Corrigan-Gibbs, H., Boneh, D.: Prio: Private, robust, and scalable computation of aggregate statistics. In: 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17). pp. 259–282 (2017)
13. Demmler, D., Schneider, T., Zohner, M.: Aby-a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
14. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985)
15. Erickson, B.J., Korfiatis, P., Akkus, Z., Kline, T.L.: Machine learning for medical imaging. Radiographics **37**(2), 505–515 (2017)
16. Fang, M., Cao, X., Jia, J., Gong, N.: Local model poisoning attacks to byzantine-robust federated learning. In: 29th {USENIX} Security Symposium ({USENIX} Security 20). pp. 1605–1622 (2020)
17. Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., Ramage, D.: Federated learning for mobile keyboard prediction. arXiv preprint arXiv:1811.03604 (2018)

18. Ion, M., Kreuter, B., Nergiz, A.E., Patel, S., Saxena, S., Seth, K., Raykova, M., Shanahan, D., Yung, M.: On deploying secure computing: Private intersection-sum-with-cardinality. In: 2020 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 370–389. IEEE (2020)
19. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al.: Advances and open problems in federated learning. arXiv preprint arXiv:1912.04977 (2019)
20. Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)
21. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
22. Li, M., Andersen, D.G., Park, J.W., Smola, A.J., Ahmed, A., Josifovski, V., Long, J., Shekita, E.J., Su, B.Y.: Scaling distributed machine learning with the parameter server. In: 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14). pp. 583–598 (2014)
23. Liu, R., Cao, Y., Yoshikawa, M., Chen, H.: Fedsel: Federated sgd under local differential privacy with top-k dimension selection. In: International Conference on Database Systems for Advanced Applications. pp. 485–501. Springer (2020)
24. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics. pp. 1273–1282. PMLR (2017)
25. Mhamdi, E.M.E., Guerraoui, R., Rouault, S.: The hidden vulnerability of distributed learning in byzantium. arXiv preprint arXiv:1802.07927 (2018)
26. Nasr, M., Shokri, R., Houmansadr, A.: Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In: 2019 IEEE symposium on security and privacy (SP). pp. 739–753. IEEE (2019)
27. Nguyen, T.D., Rieger, P., Yalame, H., Möllering, H., Fereidooni, H., Marchal, S., Miettinen, M., Mirhoseini, A., Sadeghi, A.R., Schneider, T., et al.: Flguard: Secure and private federated learning. arXiv preprint arXiv:2101.02281 (2021)
28. Nosowsky, R., Giordano, T.J.: The health insurance portability and accountability act of 1996 (hipaa) privacy rule: implications for clinical research. Annu. Rev. Med. **57**, 575–590 (2006). https://doi.org/10.1146/annurev.med.57.121304.131257
29. Paillier, P.: 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)
30. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Annual international cryptology conference. pp. 554–571. Springer (2008)
31. Phong, L.T., Aono, Y., Hayashi, T., Wang, L., Moriai, S.: Privacy-preserving deep learning via additively homomorphic encryption. IEEE Transactions on Information Forensics and Security **13**(5), 1333–1345 (2018). https://doi.org/10.1109/TIFS.2017.2787987
32. Microsoft SEAL (release 3.6). `https://github.com/Microsoft/SEAL` (Nov 2020), microsoft Research, Redmond, WA.
33. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. pp. 1310–1321. ACM (2015). https://doi.org/10.1145/2810103.2813687
34. Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. Designs, codes and cryptography **71**(1), 57–81 (2014)

35. Team, I.P.: EU general data protection regulation (GDPR): an implementation and compliance guide. IT Governance Ltd (2017). https://doi.org/10.2307/j.ctt1trkk7x
36. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)
37. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). pp. 162–167. IEEE (1986)
38. Yin, D., Chen, Y., Kannan, R., Bartlett, P.: Byzantine-robust distributed learning: Towards optimal statistical rates. In: International Conference on Machine Learning. pp. 5650–5659. PMLR (2018)
39. Zhu, L., Han, S.: Deep leakage from gradients. In: Federated Learning, pp. 17–31. Springer (2020)

A Byzantine-robustness Analysis

Cosine similarity is one of the best metrics to measure the similarity of two vectors. Recall the cosine similarity of two $\mathsf{sgn}(\mathbf{w}_i)$ and $\mathbf{w}_s$ is $c_i = \frac{\langle \mathbf{w}_i, \mathbf{w}_s \rangle}{\|\mathbf{w}_i\| \|\mathbf{w}_s\|}$, and FLTrust clips c_i using ReLU function to remove the poisoned model updates with negative c_i [11]. Based on $\mathbf{w}_i$, $\mathbf{w}_s$ in $\{-1, 1\}^d$ and Eqn. (2, 3), we have

$$
\begin{aligned}
c_i &= \frac{\sum_{j=1}^{d} \mathbf{w}_{ij} \mathbf{w}_{sj}}{\sqrt{d}\sqrt{d}} \\
&= \frac{1}{d} \left(\sum_{j=1}^{d} (1 - 2E(\mathbf{w}_{ij})) \cdot (1 - 2E(\mathbf{w}_{sj})) \right) \\
&= 1 - \frac{2}{d} \left(\sum_{j=1}^{d} (E(\mathbf{w}_{ij}) + E(\mathbf{w}_{sj}) - 2E(\mathbf{w}_{ij})E(\mathbf{w}_{sj})) \right) \\
&= 1 - \frac{2}{d} \left(\sum_{j=1}^{d} E(\mathbf{w}_{ij}) - E(\mathbf{w}_{sj}) \right) \\
&= 1 - 2\frac{hd_i}{d}.
\end{aligned}
$$

Thus, we have $c_i > 0 \iff 1 - 2\frac{hd_i}{d} > 0 \iff hd_i < \frac{d}{2}$. Therefore, with $= \frac{d}{2}$ we have $_i > 0 \iff c_i > 0$, which means -clipping Hamming distance-based method is capable to exclude the poisoned sgn model updates equivalent to that the cosine similarity-based method achieved. What is more, our -clipping Hamming distance-based method is more flexible than the cosine similarity-based one since we can alter for different tasks to achieve the best Byzantine-robustness.

B Proof of Theorem 1

Proof (of Theorem 1). The universal composability framework [10] guarantees the security of arbitrary composition of different protocols. Therefore, we only need to prove the security of individual protocols. We give the proof of the security under the semi-honest model in the real-ideal paradigm [10].

Privacy of CXOR. There is nothing to simulate as the protocol is non-interactive.

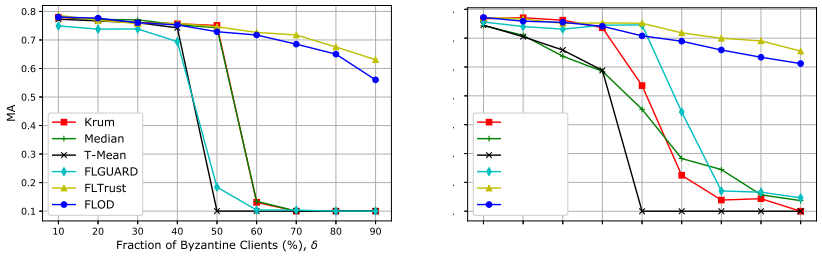
Privacy of PCBit2A. In offline phase, P_0's view in real-world is composed of $\{x_i, r_i, x_i^0, r_i^0, \mathsf{AHE.Enc}_{\mathrm{pk}_0}(y_i)\}$. To simulate it in ideal-world, the Sim can simply return $\{x_i, r_i, x_i^\theta, r_i^\theta, \mathsf{AHE.Enc}_{\mathrm{pk}_0^\theta}([0, 0, \ldots, 0])\}$ where $x_i, r_i, x_i^\theta, r_i^\theta$ are chosen from R^d at random and pk_0^θ is generated by Sim. Due to the semantic security of AHE, these two views are computationally indistinguishable from each other. And P_1's view in real execution can also be simulated by Sim which outputs two random vectors in R^d since the real-world view $\{x_i, \theta_i\}$ are masked by random vectors r_i and r_i^0. In online, the output of Sim for corrupted P_t is one share which is uniformly chosen from R^d, and thus P_t's view in the real-world is also indistinguishable from that in ideal-world.

Privacy of Private -Clipping. As the underlying garbled circuits are secure, P_t's view composed of *labels* in real-world is indistinguishable from the ideal-world view, which comprises of simulated labels.

Privacy of CSWA. In the offline, the view of P_t in the real-world is computationally indistinguishable from the ideal-world view because of the semantic security of AHE. Moreover, in the online, the real-world view of P_t is also masked random values. Sim can simulate it with random values of the same size.

Therefore, we guarantee that the adversary A^s (when corrupts P_0) learns nothing beyond what can be inferred from the aggregated results ($\sum_{i=1}^{K} \mathbb{1}\{\cdot\} w_i i_t^A$, $\sum_{i=1}^{K} \mathbb{1}\{\cdot\} i_t^A$) with an overwhelming probability. Completing the proof.

C MA of ResNet-18 on CIFAR10 with altering



(a) (b)

Fig. 8: MA of ResNet-18 on CIFAR10 with δ = 10%-90% for all Byzantine-robust aggregation methods, where 8(a) is for GA and 8(b) is for LF.

D Online Overhead of Free-HD and Private -Clipping

Table 4: Comm. and Run-time of Free − HD *and Private* τ-Clipping.

	Comm (MB)			Run-time (s)					
K	10	50	100	10		50		100	
Model				FNet	ResNet-18	FNet	ResNet-18	FNet	ResNet-18
CXOR	0	0	0	0.02	0.07	0.08	0.34	0.15	0.64
Private -Clipping	0.06	0.28	0.56	0.006		0.012		0.020	