Sanitizer: Blazing-Fast, Private, and Robust Federated Learning

Hanxiao Chen, *Member, IEEE*, Hongwei Li, Meng Hao, *Member, IEEE*, Jia Hu, *Graduate Student Member, IEEE*, Hao Ren, *Member, IEEE*, Haomiao Yang, *Senior Member, IEEE*, Tianwei Zhang, *Member, IEEE*, and Guowen Xu, *Senior Member, IEEE*

***Abstract*—Recently, private and robust federated learning (FL) schemes have been proposed to address privacy inference and Byzantine attacks simultaneously. However, existing schemes are inefficient in private and robust aggregation protocols due to the employment of heavy cryptographic techniques. To approach the above problem, we propose Sanitizer, an efficient, private, and robust FL framework. Specifically, we first design a Byzantine-robust defense for communication-efficient sign-based FL. We further propose a customized private and robust aggregation scheme built on our Byzantine-robust defense for FL. The core of our construction is two new efficient protocols, i.e., *high-dimensional boolean summation* and *weighted boolean majority vote*, which serve as the main building blocks of Sanitizer. Extensive evaluations on real-world datasets demonstrate that Sanitizer is blazing fast, achieving 19 $\sim$ 23$\times$ less runtime compared to the state-of-the-art. Meanwhile, Sanitizer achieves the same accuracy as the plaintext and superior Byzantine robustness against various classic attacks.**

***Index Terms*—Federated learning, secure multi-party computation, privacy protection, Byzantine robustness.**

I. Introduction

FEDERATED learning (FL) is a powerful distributed learning paradigm gaining widespread adoption across various fields. It's used in everyday applications like Android Gboard [1], in critical sectors like Healthcare [2], and for complex tasks like Loan risk prediction [3]. Many tech companies are actively promoting FL's development and application by offering industrial-grade platforms. Notable examples include Google's TensorFlow Federated [4], OpenMined's Pysyft [5] and Webank's FATE [6]. At a high level, FL enables distributed participants—such as mobile devices or institutions—to collaboratively train a global model without sharing their sensitive raw data. The process unfolds in rounds. In each round, the central server sends the current model to selected participants, who update it using their local data and return only the computed gradients or model updates. The server then aggregates these updates to refine the global model.

Unfortunately, FL remains vulnerable to two orthogonal yet mutually reinforcing threats: privacy leakage and Byzantine (i.e., malicious) behavior. Privacy-oriented attacks attempt to reverse-engineer a participant's local data from the gradients it uploads. This is often achieved through techniques like gradient inversion and deep leakage from gradients [7]. Byzantine adversaries, on the other hand, intentionally inject crafted updates, such as random noise, model poisoning, or backdoor triggers, to degrade the global model's accuracy, integrity, and usability [8]. Alarmingly, these two threat vectors can amplify each other [9], [10]. An attacker might first exploit Byzantine vulnerabilities to elicit more informative gradients, which then makes it easier to infer private data from other clients [9], [11]. Conversely, prior knowledge gleaned from leaked data can enable adversaries to launch more precise, adaptive, and undetectable poisoning attacks that bypass current defenses [12], [13]. This vicious feedback loop underscores a critical need for FL protocols that offer simultaneously privacy preservation and Byzantine robustness, rather than treating these two problems in isolation.

Addressing the intertwined challenges of privacy leakage and Byzantine robustness in FL, several recent works [7], [14], [15], [16], [17], [18], [19], [20], [21], [22] have proposed unified solutions. The key idea of these approaches lies in private and Byzantine-robust aggregation protocols, which are designed to first filter out malicious gradients in a privacy-preserving manner, and subsequently aggregate the benign ones privately. However, many of these proposed methods are constrained by substantial computational or communication overhead, thereby limiting their scalability and practicality in real-world deployments. We refer readers to Section VII for a comprehensive discussion of these prior works.

Received 12 July 2025; revised 10 March 2026; accepted 9 April 2026. Date of publication 27 April 2026; date of current version 14 May 2026. This work was supported in part by Chengdu Science and Technology Program under Grant 2023-XT00-00002-GX, in part by the National Natural Science Foundation of China under Grant U25B2079 and Grant 62502079, in part by Sichuan Science and Technology Program under Grant 2024ZHCG0188, in part by Sichuan Natural Science Foundation under Grant 2026NSFSC1463, and in part by China Post-Doctoral Science Foundation under Grant BX20240053. The associate editor coordinating the review of this article and approving it for publication was Prof. Zhiquan Liu. *(Corresponding author: Meng Hao.)*

Hanxiao Chen, Hongwei Li, Jia Hu, Haomiao Yang, and Guowen Xu are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: hanxiao.chen@uestc.edu.cn; hongweili@uestc.edu.cn; jiahu@std.uestc.edu.cn; haomyang@uestc.edu.cn; guowen.xu@foxmail.com).

Meng Hao is with the School of Computing and Information Systems, Singapore Management University, Singapore 178902 (e-mail: menghao303@gmail.com).

Hao Ren is with the School of Cyber Science and Engineering, Sichuan University, Chengdu 610065, China (e-mail: hao.ren@scu.edu.cn).

Tianwei Zhang is with the College of Computing and Data Science, Nanyang Technological University, Singapore 639798 (e-mail: tianwei.zhang@ntu.edu.sg).

Digital Object Identifier 10.1109/TIFS.2026.3688131

Below, we summarize the primary challenges faced by existing works. (1) *The inherent trade-off between privacy and efficiency.* Simultaneously achieving robust privacy protection and high efficiency remains elusive. Specifically, current secure aggregation schemes either expand the size of transmitted messages (e.g., in homomorphic encryption-assisted schemes [22], [23]) or necessitate multiple rounds of communication (a common drawback of protocols relying on secure multi-party computation [15], [24]). Consequently, these works often suffer from expensive communication overhead and high round complexity. (2) *The tension between privacy and robustness.* Similar to vanilla FL [10], a fundamental conflict exists because Byzantine-robust defenses typically require inspection of participant gradients, while privacy-preserving FL solutions aim to hide these gradients to avoid privacy leakage [16]. Current attempts to bridge this gap rely heavily on expensive homomorphic encryption or secure multi-party computation techniques [7]. Given these two challenges, it is generally difficult to guarantee both privacy and robustness without compromising performance.

To address the above challenges, we present Sanitizer, an FL framework that simultaneously achieves efficiency, privacy protection, and robustness against malicious participants. Specifically, Sanitizer is built upon SignSGD [8], [25], [26], a communication-efficient optimization method that leverages the sign of gradient vectors for exchange. Our protocol maintains SignSGD's communication advantage, achieving an approximate $32\times$ reduction in participants' communication compared to prior works, with only a small accuracy drop. Furthermore, we devise a Byzantine detection mechanism tailored for SignSGD. Inspired by recent Byzantine defenses [10], [27], the main idea of Sanitizer is to leverage the server's update as a baseline and filter out malicious local gradients via a sign similarity metric. We further enhance the ability of Byzantine detection by proposing a weighted majority vote for gradient aggregation.

To protect participants' privacy, we further extend the above strategy into a privacy-preserving context. Central to our construction are two new building blocks based on secret sharing primitives, which may be of independent interests. The first is *high-dimensional boolean summation*. We design a divide-and-conquer strategy to recursively break down the boolean summation problem into multiple sub-problems of smaller size, each enable to more efficient evaluation. The second is *weighted boolean majority vote*, where a boolean candidate from $\{0, 1\}$ is elected only if a majority of weighted voters support it. We remove costly operations by first reformulating this function into a crypto-friendly variant, and then designing a customized protocol based on mixed arithmetic and boolean sharing. What's more, these two protocols, along with the complete framework, are carefully designed to accommodate non-uniform bitlengths, rather than naive uniform bitlengths in prior works [7], [10]. Our main observation is that the complexity of secure protocols scales proportionally with the bitlength employed. Thus, we operate in lower bitlengths and only transition to higher bitlengths when necessary, all while ensuring the same precision as the plaintext.

With the above ingredients, we present a fully-fledged framework, Sanitizer, characterized by its lightweight and free of expensive cryptographic and public-key operations. We conduct extensive experiments, and the results show that our private and robust aggregation protocol achieves a $19 \sim 23\times$ reduction in runtime and a $50 \sim 63\times$ decrease in communication overhead compared to the state-of-the-art FLOD [7]. Besides, the communication cost of participants in Sanitizer is the same as that of sign-based FL in plaintext. Sanitizer also achieves comparable accuracy to its plaintext counterpart and exhibits superior Byzantine robustness against a variety of classic attacks [8]. Our contributions can be summarized as follows.

- We present Sanitizer, a blazing-fast, private, and robust federated learning framework.
- We propose two new privacy-preserving building blocks, i.e., high-dimensional boolean summation and weighted boolean majority vote.
- Extensive evaluations show that Sanitizer achieves up to $23\times$ runtime improvement, e.g., consuming 1 minute for private and robust aggregation on 2.07M-dimensional gradients with 50 participants.

The rest of this paper is organized as follows. Section II provides the necessary background and formalizes the problem statement for Sanitizer. Section III then introduces the designed Byzantine-robust FL strategy in plaintext. In Section IV, we detail our private and Byzantine-robust FL, Sanitizer. Experimental results are presented in Section V-C. Section VII discusses the related works, and Section VIII concludes this paper.

II. Background and Problem Statement

This section begins by defining the notations used in our work, followed by a detailed exposition of the system model, threat model, and design goals. Subsequently, we describe the cryptographic primitives employed in Sanitizer.

A. Notations

Let λ be the computational security parameter, and $[n]$ refers to the set $\{0, 1, \ldots, n-1\}$. We use bold lower-case letters (e.g., $\mathbf{x}$) to denote vectors, and $\mathbf{x}[i]$ to denote the i-th coordinate of $\mathbf{x}$. Besides, $\mathbb{1}\{\mathsf{event}\}$ denotes the indicator function that equals 1 when event is true, and 0 otherwise. $s \leftarrow D$ denotes uniformly at random sampling s from the set D.

B. System Model

Figure 1 illustrates our system model, which comprises a set of participants $P_1, P_2, \ldots, P_n$, an FL service provider S_0, and a computing server S_1. Consistent with prior works [7], [10], [21], [28], [29], S_0 coordinates the overall training process, while S_1 assists S_0 by performing private and robust aggregation through secure 2-party computation (2PC) protocols. In particular, each participant P_i for $i \in [n]$ holds a local dataset $\mathcal{D}_i$. Furthermore, aligned with state-of-the-art Byzantine-robust methods [7], [27], [30], [31], S_0 maintains a small, clean seed dataset $\mathcal{D}_s$ for Byzantine detection (refer to Section III

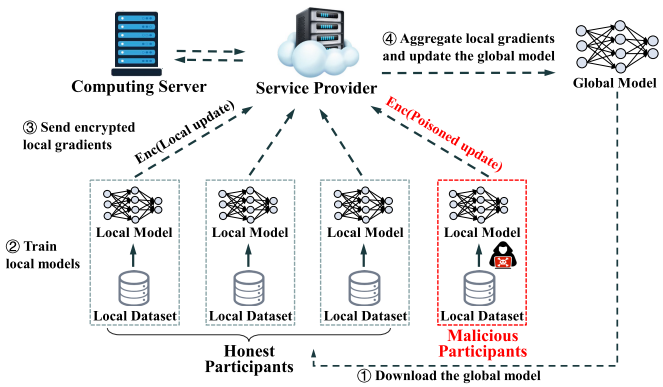


Fig. 1. System model.

for more details). We denote the overall training dataset as $\mathcal{D} = \bigcup_{i\in[n]} \mathcal{D}_i$.

The FL system collaboratively trains a global model by solving the optimization problem $\min_{\omega} \mathbb{E}_{\mathcal{D}}[\mathcal{L}(\mathcal{D}, \omega)]$, where ω represents the global model's parameters and $\mathcal{L}$ is the loss function, e.g., cross-entropy. In practice, FL typically employs stochastic gradient descent (SGD) for iterated optimization, which generally consists of three steps in each iteration. Specifically, at step ①, the service provider (called S_0) broadcasts the latest global model to the participants selected for the current iteration. Then at step ②, each participant P_i locally trains the received global model, computes its local gradient $\boldsymbol{g}_i = \mathsf{SGD}(\mathcal{D}_i, b, \omega)$, where b is the batch size. Next at step ③, P_i secretly shares $\boldsymbol{g}_i$ with S_0 and the computing server (called S_1). Note that if P_i is a malicious participant, it may submit arbitrary or poisoned gradients to compromise the model's integrity. Finally, at step ④, S_0 and S_1 securely aggregate the received local gradients by exploiting our private and robust aggregation scheme, Sanitizer, and subsequently update the global model.

Remark: Following established Byzantine-robust methods [10], [27], we assume that the server can maintain a small, clean seed dataset for the learning task. Since this dataset typically requires only a minimal number of samples (e.g., 100 samples as per [27]), it is often affordable for the server to perform manual collection and labeling. For instance, Google has successfully employed its own employees to generate seed data for Gboard's next-word prediction task [32]. Similarly, for privacy-sensitive tasks like digit recognition, the server can easily leverage crowdsourcing or internal labeling [27]. Furthermore, the seed dataset is not strictly required to follow the same distribution as the participants' training data. The robustness of this approach under distribution shifts has been validated by our evaluation and prior works [10], [27].

C. Threat Model

In Sanitizer, we consider a hybrid threat model, similar as approaches found in [33] and [34]. This model incorporates two types of adversaries: *malicious participants* that aim to actively corrupt the global model by injecting poisoning gradients, and *honest-but-curious servers* that adhere to the private and robust aggregation protocol but passively attempt to infer sensitive information, such as the target participant's training data $\mathcal{D}_i$. Note that we assume that the servers are non-colluding. Such a hybrid threat model is well-established in prior Byzantine-robust and private FL works, and is considered realistic for real-world FL systems [7], [10]. That is, the service providers (e.g., Google and Amazon) are desirable to maintain a strong reputation to provide more FL services. Furthermore, due to public accountability and strict regulations, malicious actions by these servers could lead to severe consequences [34]. In contrast, participants are numerous and may possess diverse motivations, such as for competitive interests, to maliciously corrupt ongoing FL systems.

Formally, security is modeled in the simulation paradigm [35], which defines a *real* interaction and an *ideal* interaction. In the real interaction, the parties execute the protocol according to the specification in the presence of $\mathcal{A}$. In the ideal interaction, the parties submit their inputs to an ideal functionality that faithfully executes the intended operation. A protocol Π is deemed to securely realize a functionality $\mathcal{F}$ if, for every real-world adversary $\mathcal{A}$, there exists a corresponding simulator Sim in the ideal-world interaction such that no distinguisher can differentiate between the executions of the real and ideal interactions. The protocols within Sanitizer invoke multiple sub-protocols, which are described using *hybrid model*, similar to prior works [36], [37]. This model is analogous to the real interaction, except that sub-protocols are replaced by their corresponding ideal functionalities. By convention, a protocol that invokes a functionality $\mathcal{F}$ is referred to as operating in the "$\mathcal{F}$-hybrid model".

Remark: Our design adopts a distributed trust setting with two non-colluding servers. This setting has been widely formalized and instantiated in previous research [38], [39], [40], [41], [42], [43], [44], especially within the domain of privacy-preserving FL [7], [10], [21], [29]. More importantly, this setting has found practical application in various real-world scenarios, including privacy-preserving collection of aggregated statistics [45], COVID-19 exposure notification analytics [46], and digital asset wallets [47]. For instance, Prio [38], a system for private aggregated statistics, has been integrated into Mozilla's Firefox browser[1] and employed in both iOS and Android[2] to measure the effectiveness of their Exposure Notification systems. This two-server setting offers two primary advantages.

- *Reduced participant overhead:* Unlike some privacy-preserving FL systems [24], [48] where participants are directly involved in the secure aggregation procedure, schemes within this threat model significantly reduce participant-side overhead by offloading the procedure to the two servers. This aligns with a core design goal of Sanitizer, i.e., minimizing additional overhead for local participants.
- *Efficient 2PC protocol design:* This setting benefits the design of efficient 2PC protocols for our private and robust aggregation scheme. As demonstrated in subsequent sections, we provide customized secure protocols

[1]https://blog.mozilla.org/security/
[2]https://github.com/google/exposure-notifications-android

that efficiently integrate arithmetic sharing and Boolean sharing, achieving substantial performance improvement over existing works.

D. Design Goals

Under the above threat model, our design adheres to the following key requirements.

- *Byzantine robustness*: In highly competitive real-world business environments, adversarial participants may orchestrate Byzantine attacks specifically designed to undermine a competitor's FL framework. Consequently, our primary objective is to guarantee that the global model maintains both accuracy and robustness, even when subjected to such malicious interference.
- *Privacy protection*: During the training process, participants iteratively submit local updates to the service provider. As these updates implicitly reflect the underlying information of private training data, ensuring privacy protection against potential leakage is a critical requirement.
- *Protocol efficiency*: FL is frequently deployed on resource-constrained mobile devices where computational power and communication bandwidth are at a premium. Consequently, it is imperative that any proposed scheme avoids introducing significant overhead for participants compared to vanilla FL. Furthermore, the privacy-preserving and robust aggregation protocol must be sufficiently streamlined to ensure its viability in practical, real-world deployments.

E. Cryptographic Primitives

1) Secret Sharing: Sanitizer is built on the 2-out-of-2 additive secret sharing [49], which includes *arithmetic sharing* and *boolean sharing*. Specifically, for the arithmetic sharing over the ring $\mathbb{Z}_{2^\ell}$, an ℓ-bit input x is split into two random shares $\langle x\rangle_0^\ell$, $\langle x\rangle_1^\ell$, held by S_0 and S_1, respectively, satisfying $x = \langle x\rangle_0^\ell + \langle x\rangle_1^\ell \bmod 2^\ell$. The boolean shares of $x \in \mathbb{Z}_2$ are $\langle x\rangle_0^B$ and $\langle x\rangle_1^B$, held by S_0 and S_1 respectively, satisfying $x = \langle x\rangle_0^B \oplus \langle x\rangle_1^B$. In our protocols, $\langle x\rangle^\ell$ (resp., $\langle x\rangle^B$) denotes that S_b holds $\langle x\rangle_b^\ell$ (resp., $\langle x\rangle_b^B$) for $b \in \{0,1\}$. The secret sharing technique guarantees that given a boolean or arithmetic share $\langle x\rangle_0$ or $\langle x\rangle_1$, the value of x is perfectly hidden. Additive secret sharing naturally supports linear operations without communication. For instance, to compute $cx+y$ with a public constant $c \in \mathbb{Z}_{2^\ell}$ and secret shares $\langle x\rangle^\ell$ and $\langle y\rangle^\ell$, S_b for $b \in \{0,1\}$ can locally compute $\langle z\rangle_b^\ell = c\langle x\rangle_b^\ell + \langle y\rangle_b^\ell$, where $\langle z\rangle_0^\ell + \langle z\rangle_1^\ell = cx + y \bmod 2^\ell$.

2) Correlated Oblivious Transfer: The 1-out-of-2 correlated oblivious transfer (COT), denoted by $\mathcal{F}_{\text{COT}_\ell}$, is defined as follows [50]: given a correlation function $f(r) = r + x$ from the *sender* and a choice bit $b \in \{0,1\}$ from the *receiver*, the protocol outputs a random element $r \in \{0,1\}^\ell$ to the sender and $r + b \cdot x \in \{0,1\}^\ell$ to the receiver. COT is typically realized through OT extension [51], a method that efficiently generates a large amount of COT instances from a few computationally expensive base-OT instances, relying solely on simple symmetric cryptographic operations. In this work, we utilize the state-of-the-art low-communication *silent* OT extension from Ferret [52] to construct our protocols, which incur a communication cost of $\ell + 1$ bits per COT.

3) Functionalities: We adopt the following 2-party functionalities in Sanitizer. For completeness, we also give the detailed protocols of these functionalities in Appendix A.

- *ReLU* ($\mathcal{F}_{\text{ReLU}}$): This functionality takes as input $\langle x\rangle^\ell \in \mathbb{Z}_{2^\ell}$, and returns $\langle y\rangle^\ell \in \mathbb{Z}_{2^\ell}$ such that $y = \mathbb{1}\{x \geq 0\} \cdot x$. Cheetah [53] gave an efficient ReLU protocol with less than 11ℓ bits of communication.
- *Extension with known MSB* ($\mathcal{F}_{\text{Ext}}$): This functionality takes as input $\langle x\rangle^\ell \in \mathbb{Z}_{2^\ell}$, and returns $\langle y\rangle^{\ell'} \in \mathbb{Z}_{2^{\ell'}}$ ($\ell' > \ell$) such that $y = x$ holds. When the most significant bit (MSB) of x is known, Sirnn [37] gave an optimized extension protocol. We can instantiate this protocol with the above silent OT extension technique [52], which consumes $\ell' - \ell + 3$ bits of communication.

III. Byzantine-Robust FL in Plaintext

In this section, we propose a Byzantine-robust FL scheme in the plaintext environment. Our scheme is built upon SignSGD [8], [25], a representative sign-based FL approach, and we further address the communication bottleneck and Byzantine issues at the same time.

We first reformulate the sign-based aggregation in SignSGD [8], [25]. Specifically, each participant locally updates their downloaded copy of the global model and submits the compressed gradient, which is the gradient's sign, instead of its full-precision values. Formally, this compression procedure is defined by an encoder $\mathsf{Sign} : \mathbb{R} \to \{+1, -1\}$, operating on each coordinate of the gradient. It outputs + 1 if the input is non-negative and -1 otherwise. To facilitate cryptographic evaluation, we further encode each sign with a single bit, i.e., $-1 \to 0$ and $+1 \to 1$. The formula for this variant is $\mathsf{BSign}(x) = \mathbb{1}\{x \geq 0\}$. Compared to the vallina FL, this sign-based solution realizes a $32\times$ communication reduction [8], making it particularly suitable for resource-constrained devices.

Below, we enhance the sign-based aggregation to provide robustness against Byzantine participants. As illustrated in [8], malicious participants can corrupt model training at any iteration by sending poisoned gradients. To counteract this, we design a robust aggregation strategy specifically for the sign-based aggregation. The main insight is that the service provider collects a small but clean seed dataset, and computes its own gradient sign $\mathbf{g}_s$ on this dataset, which serves as the baseline to detect and exclude Byzantine participants. This seed dataset-based method has also been adopted in recent Byzantine-robust works [7], [10], [27] and demonstrates the most advanced robustness performance. More importantly, this method is more compatible with secure computation protocols than defenses relying on pairwise comparisons [10]. Specifically, our defense mechanism against Byzantine participants consists of the following three steps.

Step 1: similarity score calculation. The server first calculates a similarity score s_i for each i-th participant's sign gradient $\mathbf{g}_i$. This score quantifies the consistency between $\mathbf{g}_i$ and the server's own sign gradient $\mathbf{g}_s$, which is computed

from the clean seed dataset. Formally, this is expressed as $\sum_{j\in[d]} \mathbf{g}_s[j]\bar{\oplus}\mathbf{g}_i[j]$, where $\bar{\oplus}$ denotes the XNOR operation (i.e., XOR followed by NOT) and d is the dimension of the gradients.

Step 2: Byzantine participant Ffltering. Next, the server filters Byzantine participants based on their calculated similarity scores. If s_i falls below a predefined threshold t, the participant is identified as malicious. This process is formalized as the evaluation of $\mathsf{ReLU}(s_i - t)$, where $\mathsf{ReLU}(x) = x$ if $x \geq 0$ and 0 otherwise). The output of this function, w_i, serves as the i-th participant's weight in the subsequent aggregation process. We empirically set $t = d/2$, as the specific value of this threshold exhibited minimal impact on model accuracy in our evaluations.

It is worth noting that developing adaptive thresholds becomes non-trivial when operating over encrypted gradients, as the inability to access plaintext gradients precludes direct statistical analysis of the underlying data distribution. Specifically, since participants' gradients are protected via secure multi-party computation primitives in our work, the server is precluded from observing the raw gradient distribution. This lack of plaintext visibility prevents the application of standard statistical heuristics, such as median absolute deviation or variance-based profiling, which are typically essential for dynamic thresholding [54]. Implementing such logic within the encrypted domain would necessitate complex secure comparison protocols or oblivious sorting, thereby imposing expensive communication and computational overhead. Consequently, we employ a fixed threshold to strike an optimal balance between efficiency and Byzantine robustness.

Step 3: weighted aggregation. Finally, the server executes a weighted aggregation of the gradients. This step further mitigates the impact of potentially surviving malicious gradients that might have passed the initial filtering. The aggregation is performed as follows

$$\mathbf{g}'[i] = \sum_{j\in\{j|\mathbf{g}_j[i]=1\}} w_j - \sum_{j\in\{j|\mathbf{g}_j[i]=0\}} w_j. \tag{1}$$

The server then broadcasts the sign of the aggregated gradient. This process can be viewed as a weighted majority aggregation. The remaining process is identical to vanilla FL, and is therefore omitted here. The complete scheme is presented in Algorithm 1.

Algorithm 1 Plaintext Byzantine-Robust Sanitizer

Input: Each participant P_i, $i \in [n]$, with a local dataset $\mathcal{D}_i$; the FL server with a dataset $\mathcal{D}_s$; dimension of each gradient d; learning rate η; batch size b; Byzantine filter threshold t; and the number of training iterations Iter.

Output: Global model weight $\boldsymbol{\omega}$.

$\boldsymbol{\omega} \leftarrow$ random initialization.
for iter $\in [\mathsf{Iter}]$ **do**
  I. Training
  for $i \in [n]$ **do**
    $\mathbf{g}'_i = \mathsf{SGD}(\boldsymbol{\omega}, \mathcal{D}_i, b)$. ▷$P_i$ computes the local gradient
    $\mathbf{g}_i = \mathsf{BSign}(\mathbf{g}'_i)$. ▷$P_i$ computes the sign of the gradient
    Submit $\mathbf{g}_i$ to the server.
  $\mathbf{g}'_s = \mathsf{SGD}(\boldsymbol{\omega}, \mathcal{D}_s, b)$. ▷The server computes its gradient
  $\mathbf{g}_s = \mathsf{BSign}(\mathbf{g}'_s)$. ▷The server computes the gradient's sign
  II. Robust aggregation
  for $i \in [n]$ **do**
    $\mathbf{s}_i[j] = \mathbf{g}_s[j]\bar{\oplus}\mathbf{g}_i[j]$, for $j \in [d]$.
    $s_i = \sum_{j\in[d]} \mathbf{s}_i[j]$. ▷Compute similarity scores
    $w_i = \mathsf{ReLU}(s_i - t)$. ▷Filter out Byzantine participants
  For $i \in [d], \mathbf{g}'[i] = \sum_{j\in\{j\in[n]|\mathbf{g}_j[i]=1\}} w_j - \sum_{j\in\{j\in[n]|\mathbf{g}_j[i]=0\}} w_j$. ▷Execute weighted gradient aggregation
  $\mathbf{g} = \mathsf{BSign}(\mathbf{g}')$.
  III. Broadcast
  Broadcast $\mathbf{g}$ to all participants.
  Each participant updates the local model $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta \cdot \mathbf{g}$.

IV. Private and Byzantine-Robust FL - Sanitizer

In this section, we adapt the Byzantine-robust aggregation strategy in Algorithm 1 to the privacy-preserving environment. Before detailing the complete framework, we will first introduce two new building blocks for Sanitizer.

A. Building Blocks

We identify two operations, *high-dimensional boolean summation* and *weighted boolean majority vote*, that dominate the computational and communication overhead of Sanitizer (refer to Table IV in Section VI-B). To facilitate their performance, we design new and efficient protocols specifically for these two operations.

TABLE I

Communication Complexity of the Private and Robust Aggregation Protocol in Sanitizer. k Is a Hyper-Parameter. $\ell_0 = \lceil \log(\lceil \frac{d}{k} \rceil + 1) \rceil + 1$ Is the Intermediate Bitlenth in Π_{BoolSum}, $\ell_1 = \lceil \log(D+1) \rceil + 1$, and $\ell_2 = \ell_1 + \lceil \log n \rceil$

Building Blocks	Communication (bits)
Gradient sign secret-sharing	nd
Sign similarity evaluation	$nd(\ell_0+1) + k(\ell_1 - \ell_0 + 3)$
Byzantine participant filtering	$11n\ell_1$
Weighted gradient aggregation	$n(\ell_2 - \ell_1 + 3) + 2nd(\ell_2 + 1)$

High-dimensional boolean summation. The high-dimensional boolean summation operation is crucial for evaluating the quality of participants' gradients and dominates the overall overhead as shown in Section VI-B. The corresponding functionality is formally defined in Figure 2. Specifically, two parties take boolean shares $\langle \mathbf{a} \rangle^B$ of a d-dimensional vector $\mathbf{a}$ as input. The output consists of arithmetic shares $\langle b \rangle^\ell$ where $b = \sum_{i\in[d]} \mathbf{a}[i]$ and $\ell = \lceil \log(d+1) \rceil + 1$. The parameter ℓ is critical as it ensures that the resulting signed value b does not overflow within the ring $\mathbb{Z}_{2^\ell}$.

Prior works [55], [56], [57] typically rely on a boolean adder such as the parallel prefix adder. These adders consist of AND gates and cost-free XOR gates, computing the final summation through bit-by-bit sums and carries. For a d-dimensional input,

TABLE II
OVERHEAD COMPARISON WITH THE STATE-OF-THE-ART FLOD [7]

# Gradients	# Participants	Runtime (Sec)			Communication (MB)		
		FLOD	Ours	Improvement	FLOD	Ours	Improvement
507K	10	53.91	2.30	23.43×	2321.37	36.46	63.68×
	50	263.79	12.32	21.41×	12137.87	231.03	52.54×
	100	522.97	24.90	21.00×	23199.45	462.59	50.15×
2.07M	10	235.29	11.53	20.41×	12148.34	192.72	63.04×
	50	1172.11	60.62	19.34×	60012.97	964.65	62.21×
	100	2324.25	106.84	21.76×	113075.65	1929.29	58.61×

TABLE III
ACCURACY COMPARISON WITH PLAINTEXT UNDER MINIMAL BITLENGTHS

Dataset	# Participants	Bitlengths			Accuracy	
		ℓ_0	ℓ_1	ℓ_2	Plaintext	Ours
MNIST	10	5	25	29	97.49%	97.49%
	50	5	25	31	93.08%	93.08%
CIFAR10	10	5	25	29	85.11%	85.11%
	50	5	25	31	79.90%	79.90%

Algorithm 2 High-Dimensional Boolean Summation Protocol, Π_{BoolSum}

Input: S_0 and S_1 hold $\langle \mathbf{a} \rangle_0^B$ and $\langle \mathbf{a} \rangle_1^B$, respectively, where $\mathbf{a} \in \{0,1\}^d$.

Output: S_0 and S_1 learn $\langle b \rangle_0^\ell$ and $\langle b \rangle_1^\ell$, respectively, where $b = \sum_{i\in[d]} \mathbf{a}[i]$ and $\ell = \lceil \log(d+1) \rceil + 1$.

1: For $i \in [d]$, S_0 and S_1 invoke $\mathcal{F}_{\mathsf{COT}_{\ell'}}$ with $\ell' = \lceil \log \left(\lceil \frac{d}{k} \rceil + 1 \right) \rceil + 1$, where S_0 is the sender with input the correlation function $f(\mathbf{r}[i]) = \mathbf{r}[i] - 2\langle \mathbf{a}[i] \rangle_0^B$, and S_1 is the receiver with the choice bit $\langle \mathbf{a}[i] \rangle_1^B$. After that, S_0 and S_1 learn $\langle \mathbf{x} \rangle_0^{\ell'} = \mathbf{r}[i]$ and $\langle \mathbf{x} \rangle_1^{\ell'} = \langle \mathbf{a}[i] \rangle_1^B \mathbf{r}[i] - 2\langle \mathbf{a}[i] \rangle_0^B$, respectively.

2: S_0 and S_1 compute $\langle \mathbf{y} \rangle_0^{\ell'} = \langle \mathbf{a} \rangle_0^B - \langle \mathbf{x} \rangle_0^{\ell'}$ and $\langle \mathbf{y} \rangle_1^{\ell'} = \langle \mathbf{a} \rangle_1^B + \langle \mathbf{x} \rangle_1^{\ell'}$, respectively.

3: For $j \in [k]$, S_0 and S_1 compute the sum of sub-vector $\langle \mathbf{z}[j] \rangle^{\ell'} = \sum_{i=j\cdot\frac{d}{k}}^{(j+1)\cdot\frac{d}{k}-1} \langle \mathbf{y}[i] \rangle^{\ell'}$.

4: For $i \in [k]$, S_0 and S_1 invoke $\mathcal{F}_{\mathsf{Ext}}$ with inputs $\langle \mathbf{z}[i] \rangle^{\ell'}$, and learn $\langle \mathbf{z}[i] \rangle^{\ell}$.

5: S_0 and S_1 output $\langle b \rangle_0^\ell = \sum_{j\in[k]} \langle \mathbf{z}[j] \rangle_0^\ell$ and $\langle b \rangle_1^\ell = \sum_{j\in[k]} \langle \mathbf{z}[j] \rangle_1^\ell$, respectively.

this approach requires approximately $O(d \log d)$ AND gates with a circuit depth of $\log d$ [55]. This creates a bottleneck in federated learning, where input vectors (i.e., the gradient signs) are often exceptionally high-dimensional, reaching millions for common neural networks like ResNet [58]. Consequently, this solution leads to substantial communication overhead when implemented via secret-sharing techniques. To address this, we propose a COT-based protocol in Algorithm 2.

Our initial idea is to perform a boolean-to-arithmetic sharing conversion, enabling a free summation in the arithmetic sharing. However, this approach would require d invocations of boolean-to-arithmetic sharing conversion, each of which consumes an $\mathcal{F}_{\mathsf{COT}_\ell}$ functionality. As a result, this solution incurs a communication cost of at least $d(\ell+1)$. To ensure correctness and prevent overflow, the bitlength ℓ for the COT execution must be set to at least $\lceil \log(d+1) \rceil + 1$, to accommodate the summation of all d components of the vector. To optimize communication performance, we employ a divide-and-conquer strategy that recursively decomposes the whole boolean summation problem into multiple sub-problems of smaller size. This allows us to operate with smaller bitlengths and only scale up when necessary, maintaining full precision while minimizing communication costs. Specifically, we decompose the d-dimensional boolean summation problem into k sub-problems, each involving $\lceil \frac{d}{k} \rceil$ coordinates. This significantly reduces the required bitlength in COT from $\lceil \log(d+1) \rceil + 1$ to just $\lceil \log \left(\lceil \frac{d}{k} \rceil + 1 \right) \rceil + 1$. Following this, we sum the k sub-summations by first invoking $\mathcal{F}_{\mathsf{Ext}}$ (as described in Section II-E3), which extends the bitlength from $\lceil \log \left(\lceil \frac{d}{k} \rceil + 1 \right) \rceil + 1$ to $\lceil \log(d+1) \rceil + 1$. Notably, in Sanitizer, the inputs of $\mathcal{F}_{\mathsf{Ext}}$ are always positive, allowing us to leverage the optimized variant that utilizes the known MSB [37].

Note that our silent OT-based implementation of $\mathcal{F}_{\mathsf{Ext}}$, detailed in Algorithm 5, requires about $\log k + 3$ bits of communication. Therefore, the total communication overhead of our improved boolean summation design is $d \left(\lceil \log \left(\lceil \frac{d}{k} \rceil + 1 \right) \rceil + 1 \right) + k(\log k + 3)$ bits. We strategically select k to minimize this communication cost. For example, in our applications, setting $\lceil \frac{d}{k} \rceil$ to 15 achieves an

Parameters: Input vector dimension d; output bitlength $\ell = \lceil \log(d+1) \rceil + 1$.

Execution: On input boolean shares $\langle \mathbf{a} \rangle^B$ of a d-dim vector $\mathbf{a}$ from S_0 and S_1:

1) Choose a uniformly random value $r \leftarrow \mathbb{Z}_{2^\ell}$.
2) Define $\langle b \rangle_0^\ell = r$ and $\langle b \rangle_1^\ell = b - r \mod 2^\ell$ where $b = \sum_{i\in[d]} \mathbf{a}[i]$.
3) Output $\langle b \rangle_0^\ell$ to S_0 and $\langle b \rangle_1^\ell$ to S_1.

Fig. 2. Ideal functionality $\mathcal{F}_{\mathsf{BoolSum}}$ for high-dimensional boolean summation.

TABLE IV
OVERHEAD BREAKDOWN OF SANITIZER WITH 10 PARTICIPANTS

Datasets	Metrics	Sign sharing	Similarity Evaluation	Byzantine Filtering	Weighted Aggregation	Total Cost
MNIST	Time (Sec)	0.12	18.80	0.13	34.85	53.91
	Comm. (MB)	1.33	149.71	0.0006	891.32	1042.37
CIFAR10	Time (Sec)	0.12	21.40	0.10	37.94	59.56
	Comm. (MB)	13.32	149.73	0.0006	891.42	1054.47

Parameters: The number of input n; input bitlength ℓ; intermediate output bitlength $\ell' = \ell + \lceil \log n \rceil$.
Execution: On input arithmetic shares $\langle a_i \rangle^\ell$ and boolean shares $\langle b_i \rangle^B$ of b_i for $i \in [n]$ from S_0 and S_1:
1) Compute $c = \sum_{i \in \{i \in [n] | b_i = 1\}} a_i - \sum_{i \in \{i \in [n] | b_i = 0\}} a_i \mod 2^{\ell'}$.
2) Compute $c' = \mathsf{BSign}(c) \in \{0, 1\}$.
3) Output (c, c') to S_0.

Fig. 3. Ideal functionality $\mathcal{F}_{\mathsf{WeightVote}}$ for weighted boolean majority vote.

approximate 4.5× communication reduction compared to the initial solution, while also yielding improved runtime. Table III in Section V-C presents the required bitlengths of this protocol across various vector dimensions.

Theorem 1: Protocol Π_{BoolSum} in Algorithm 2 securely realizes $\mathcal{F}_{\mathsf{BoolSum}}$ against semi-honest adversaries in the ($\mathcal{F}_{\mathsf{Ext}}$, $\mathcal{F}_{\mathsf{COT}}$)-hybrid model.

Proof: The detailed proof is given in Appendix B.■

Weighted boolean majority vote. In this functionality, two parties take as input arithmetic shares $\langle a_i \rangle^\ell$ of the i-th weight $a_i \in \mathbb{Z}_{2^\ell}$ and boolean shares $\langle b_i \rangle^B$ of the i-th vote $b_i \in \{0, 1\}$ for each participant $i \in [n]$. Their objective is to compute the weighted boolean majority vote $\mathsf{BSign}(c)$, where

$$c = \sum_{i \in \{i \in [n] | b_i = 1\}} a_i - \sum_{i \in \{i \in [n] | b_i = 0\}} a_i. \tag{2}$$

This implies that a *boolean* candidate from $\{0, 1\}$ is elected only when a majority of voters with *higher weights* have supported it. In Figure 3, we illustrate a simplified functionality, where both the aggregated weight c and the final boolean vote $\mathsf{BSign}(c)$ are sent to S_0. This design is compatible with the FL setting, since c represents the aggregated gradient that should be known by the FL server [24], [48].

The primary challenges in its secure evaluation stem from two factors: (1) the requirement for hybrid operations over both boolean and arithmetic shares, and (2) the need for n equality tests, which are expensive for secure multi-party computation [36], [53]. To mitigate these issues, prior works such as FLOD [7] convert the boolean shares into arithmetic shares using an improved invariant of ABY [49], subsequently performing multiplications based on Beaver's triples [59]. However, these operations incur prohibitively high overhead, particularly due to the frequent invocation of homomorphic encryption for Beaver's triple generation on high-dimensional inputs.

Algorithm 3 Weighted Boolean Majority Vote Protocol, $\Pi_{\mathsf{WeightVote}}$

Input: S_0 and S_1 hold $\langle a_i \rangle^\ell \in \mathbb{Z}_{2^\ell}$ and $\langle b_i \rangle^B \in \mathbb{Z}_2$, for $i \in [n]$.
Output: S_0 learns $\mathsf{BSign}(c)$ such that $c = \sum_{i \in \{i \in [n] | b_i = 1\}} a_i - \sum_{i \in \{i \in [n] | b_i = 0\}} a_i \mod 2^{\ell'}$, where $\ell' = \ell + \lceil \log n \rceil$.
1: S_0 and S_1 invoke $\mathcal{F}_{\mathsf{Ext}}$ with inputs $\langle a_i \rangle^\ell$, and learn $\langle a_i \rangle^{\ell'}$, for $i \in [n]$.
2: For $i \in [n]$, S_0 and S_1 invoke $\mathcal{F}_{\mathsf{COT}_{\ell'}}$, where S_0 is the sender with input the correlation function $f(r) = 2\langle a_i \rangle_0^{\ell'} - 4\langle b_i \rangle_0^B \cdot \langle a_i \rangle_0^{\ell'} + r$ and S_1 is the receiver with the choice bit $\langle b_i \rangle_1^B$. After that, S_0 and S_1 set the outputs as $\langle x_i \rangle_0^{\ell'}$ and $\langle x_i \rangle_1^{\ell'}$, respectively, such that $x_i = \langle b_i \rangle_1^B \cdot (2\langle a_i \rangle_0^{\ell'} - 4\langle b_i \rangle_0^B \cdot \langle a_i \rangle_0^{\ell'})$.
3: For $i \in [n]$, S_0 and S_1 invoke $\mathcal{F}_{\mathsf{COT}_{\ell'}}$, where S_1 is the sender with input the correlation function $f(r) = 2\langle a_i \rangle_1^{\ell'} - 4\langle b_i \rangle_1^B \cdot \langle a_i \rangle_1^{\ell'} + r$ and S_0 is the receiver with the choice bit $\langle b_i \rangle_0^B$. After that, S_0 and S_1 set the outputs as $\langle y_i \rangle_0^{\ell'}$ and $\langle y_i \rangle_1^{\ell'}$, respectively, such that $y_i = \langle b_i \rangle_0^B \cdot (2\langle a_i \rangle_1^{\ell'} - 4\langle b_i \rangle_1^B \cdot \langle a_i \rangle_1^{\ell'})$.
4: S_0 and S_1 set $\langle c \rangle_0^{\ell'} = \sum_{i \in [n]} ((2\langle b_i \rangle_0^B - 1) \cdot \langle a_i \rangle_0^{\ell'} + \langle x_i \rangle_0^{\ell'} + \langle y_i \rangle_0^{\ell'})$ and $\langle c \rangle_1^{\ell'} = \sum_{i \in [n]} ((2\langle b_i \rangle_1^B - 1) \cdot \langle a_i \rangle_1^{\ell'} + \langle x_i \rangle_1^{\ell'} + \langle y_i \rangle_1^{\ell'})$, respectively.
5: S_0 and S_1 reconstruct c to S_0, who outputs $\mathsf{BSign}(c)$.

To achieve superior efficiency, Sanitizer introduces a new protocol, presented in Algorithm 3, which incorporates two novel insights. First, we eliminate the aforementioned expensive operations by reformulating the original weighted boolean majority vote function into a crypto-friendly variant. This reformulated version relies exclusively on secure Boolean-arithmetic inner products, as defined below:

$$\begin{aligned} c &= \sum_{i \in \{i \in [n] | b_i = 1\}} a_i - \sum_{i \in \{i \in [n] | b_i = 0\}} a_i \\ &= \sum_{i \in \{i \in [n] | b_i = 1\}} a_i - \left(\sum_{i \in [n]} a_i - \sum_{i \in \{i \in [n] | b_i = 1\}} a_i \right) \\ &= 2 \sum_{i \in \{i \in [n] | b_i = 1\}} a_i - \sum_{i \in [n]} a_i \\ &= \sum_{i \in [n]} (2 a_i b_i - a_i). \end{aligned} \tag{3}$$

With this reformulation, the computationally intensive component is the first term (i.e. $2a_ib_i$), which represents a

boolean-arithmetic inner product, while the second term (i.e., $-a_i$) can be evaluated locally.

We compute this boolean-arithmetic inner product by leveraging our second insight. Specifically, Equation (3) can be efficiently evaluated via a tailored COT-based protocol designed for mixed-mode sharing. This approach significantly alleviates the communication and computational bottlenecks inherent in prior methods. The underlying relation is formulated as follows:

$$\begin{aligned}
&\sum_{i\in[n]}(2a_ib_i - a_i)\\
&= \sum_{i\in[n]}\left(\langle a_i\rangle_0^\ell + \langle a_i\rangle_1^\ell\right)\cdot\left(2\left(\langle b_i\rangle_0^B \oplus \langle b_i\rangle_1^B\right) - 1\right)\\
&= \sum_{i\in[n]}\left(\langle a_i\rangle_0^\ell + \langle a_i\rangle_1^\ell\right)\\
&\quad\cdot\left(2\langle b_i\rangle_0^B + 2\langle b_i\rangle_1^B - 4\langle b_i\rangle_0^B\cdot\langle b_i\rangle_1^B - 1\right)\\
&= \sum_{i\in[n]}(2\langle b_i\rangle_0^B - 1)\cdot\langle a_i\rangle_0^\ell\\
&\quad + \sum_{i\in[n]}\langle b_i\rangle_1^B\cdot\left(2\langle a_i\rangle_0^\ell - 4\langle b_i\rangle_0^B\cdot\langle a_i\rangle_0^\ell\right)\\
&\quad + \sum_{i\in[n]}(2\langle b_i\rangle_1^B - 1)\cdot\langle a_i\rangle_1^\ell\\
&\quad + \sum_{i\in[n]}\langle b_i\rangle_0^B\cdot\left(2\langle a_i\rangle_1^\ell - 4\langle b_i\rangle_1^B\cdot\langle a_i\rangle_1^\ell\right). \qquad (4)
\end{aligned}$$

Observe that the first and third terms can be computed locally by S_0 and S_1, respectively. The remaining two terms can be evaluated following the $\mathcal{F}_{\mathsf{COT}}$-based boolean-arithmetic multiplication [37], [40]. The complete algorithm is presented in Algorithm 3. Notice that in line 1 of Algorithm 3, we invoke $\mathcal{F}_{\mathsf{Ext}}$ to extend the bitlength ℓ to $\ell' = \ell + \lceil\log n\rceil$. This non-uniform bitlength is essential to prevent overflow and ensure enough space to accommodate the output.

Theorem 2: Protocol Π_{BoolSum} in Algorithm 3 securely realizes $\mathcal{F}_{\mathsf{WeightVote}}$ against semi-honest adversaries in the ($\mathcal{F}_{\mathsf{Ext}}$, $\mathcal{F}_{\mathsf{COT}}$)-hybrid model.

The security proof is similar to Π_{BoolSum}. We omit the detailed proof due to space limitations.

Remark: Our two core components, high-dimensional Boolean summation and weighted Boolean majority vote, may be independently useful. To demonstrate the independent interest of our protocols, we offer two representative scenarios. The first is privacy-preserving statistical analysis. This scenario involves participants protecting sensitive inputs, such as whether they are diagnosed with a disease or have clicked a specific link, using boolean shares. To determine the aggregate count of individuals satisfying specific criteria, a service provider can employ our high-dimensional Boolean summation protocol to facilitate secure computation without compromising individual privacy. The second is secure risk assessment. Consider a financial institution evaluating a borrower's risk profile based on multiple weighted risk factors. Each factor is represented as a boolean value indicating whether a specific criterion is met. By utilizing our weighted Boolean majority vote protocol, the institution can securely assess the weighted interplay between positive and negative factors. This allows for a final determination on risk controllability without exposing the borrower's raw sensitive attributes.

B. The Fully-Fledged Sanitizer Framework

Building upon the aforementioned building blocks, we construct the complete Sanitizer framework, depicted in Figure 4. This framework consists of three procedures: local training on the global model, private and robust aggregation, and broadcasting the global model. Our subsequent discussion primarily focuses on the private and robust aggregation phase, which comprises four steps: gradient sign secret-sharing, sign similarity evaluation, Byzantine participant filtering, and weighted gradient aggregation. A detailed breakdown of the communication complexity for this phase is presented in Table I. In the following, before elaborating on these steps, we first analyze the required minimum bitlength Sanitizer needed.

1) Remark (Minimum Bitlength Requirements): We analyze the minimum bitwidth ℓ for arithmetic shares required to achieve both optimal communication cost and exact correctness, mirroring the plaintext scheme. Specifically, to prevent overflow in intermediate calculation, ℓ must satisfy $\ell \geq \lceil\log(d+1)\rceil + \lceil\log n\rceil + 1$ bits, where d denotes the gradient dimension and n is the number of participants. This bound ensures the representation can accommodate all signed intermediate values, particularly during the "bottleneck" evaluation of $\mathbf{g}'[j] = \sum_{i\in\{i\in[n]|\mathbf{g}_i[j]=1\}} w_i - \sum_{i\in\{i\in[n]|\mathbf{g}_i[j]=0\}} w_i$. Here, each sign similarity score w_i requires $\lceil\log(d+1)\rceil + 1$ bits (the derivation of this expression will be detailed later). While a uniform bitlength ℓ could be employed across all arithmetic shares in Sanitizer, such a naive approach would significantly increase the communication overhead, as the complexity of our underlying primitives is directly proportional to ℓ. To address this, we depart from the uniform bitlength conventions seen in prior works [7], [10] by proposing a non-uniform bitlength framework. At a high level, our protocols operate on minimal bitwidths by default, transitioning to higher bitwidths only when necessitated by specific computational functions, thereby optimizing overall efficiency. The subsequent sections will detail how protocols are constructed to effectively manage these non-uniform bitlengths.

2) Gradient Sign Secret-Sharing: In this step, each participant P_i secret-shares their gradient sign $\mathbf{g}_i$ with the two servers. To minimize communication overhead, we employ boolean sharing, which is fully compatible with our BSign sign encoding variant described in Section III. More precisely, P_i samples a random boolean vector $\mathbf{r} \in \{0,1\}^d$, and then computes and sends $\langle\mathbf{g}_i\rangle_0^B = \mathbf{g}_i \oplus \mathbf{r}$ and $\langle\mathbf{g}_i\rangle_1^B = \mathbf{r}$ to S_0 and S_1, respectively. This construction ensures that $\langle\mathbf{g}_i\rangle_0^B \oplus \langle\mathbf{g}_i\rangle_1^B = \mathbf{g}_i$. To further reduce the communication overhead, we implement a technique where P_i only sends $\langle\mathbf{g}_i\rangle_0^B$ to S_0. This is facilitated by a one-time key setup between each P_i and S_1, establishing random keys among them for a pseudo-random function (PRF) [60]. Subsequently, P_i and S_1 non-interactively sample the identical random vector $\mathbf{r}$ via these PRFs, which can be instantiated using methods such as AES in counter mode [61]. This technique has also been adopted in prior privacy-preserving machine learning works [56]. Through this optimization, the

PARAMETERS:

- Dimension of each gradient d; learning rate η; batch size b; filter threshold t; and the number of training iterations Iter.
- Each participant P_i with a local dataset $\mathcal{D}_i$, $i \in [n]$, S_0 with a seed dataset $\mathcal{D}_s$. Initial weights $\boldsymbol{\omega}$ of the global model.

PROTOCOL:

I. **Training**: Repeat the steps I-III until the stopping criterion.

1) For $i \in [n]$, each party P_i runs $\mathsf{SGD}(\omega, \mathcal{D}_i, b) \to \mathbf{g}'_i$, and computes the gradient sign $\mathbf{g}_i = \mathsf{BSign}(\mathbf{g}'_i) \in \mathbb{Z}_2^d$.
2) S_0 runs $\mathsf{SGD}(\omega, \mathcal{D}_s, b) \to \mathbf{g}'_s$, and computes the gradient sign $\mathbf{g}_s = \mathsf{BSign}(\mathbf{g}'_s) \in \mathbb{Z}_2^d$.

II. **Private and robust aggregation**:

Step 1: Gradient sign secret-sharing

1) P_i and S_1 generate (pseudo-)random $\mathsf{r}_i \in \mathbb{Z}_2^d$ for $i \in [n]$ using PRF, assuming a PRF key has been agreed in advance between P_i and S_1. Then, P_i sends $\langle \mathbf{g}_i \rangle_0^B = \mathbf{g}_i \oplus \mathsf{r}_i$ to S_0 while S_1 defines $\langle \mathbf{g}_i \rangle_1^B = \mathsf{r}_i$.

Step 2: Sign similarity evaluation

2) S_0 and S_1 compute $\langle \mathbf{s}_i \rangle^B = \langle \mathbf{g}_i \rangle^B \bar{\oplus} \langle \mathbf{g}_s \rangle^B$ for $i \in [n]$. The parties invoke $\mathcal{F}_{\mathrm{BoolSum}}$ with input $\langle \mathbf{s}_i \rangle^B$, and learn $\langle s_i \rangle^{\ell_1}$, where $\ell_1 = \lceil \log(d+1) \rceil + 1$.

Step 3: Byzantine participant filtering

3) For $i \in [n]$, S_0 and S_1 invoke $\mathcal{F}_{\mathrm{ReLU}}$ with input $\langle s_i - t \rangle^{\ell_1}$, and learn $\langle w_i \rangle^{\ell_1}$, where t is a pre-defined threshold.

Step 4: Weighted gradient aggregation

4) For $j \in [d]$, S_0 and S_1 invoke $\mathcal{F}_{\mathrm{WeightVote}}$ with input $\langle w_i \rangle^{\ell_1}$ and $\langle \mathbf{g}_i[j] \rangle^B$ for $i \in [n]$, and learn $\mathbf{g} = \mathsf{BSign}(\mathbf{g}')$, where $\mathbf{g}'[j] = \sum_{i \in \{i \in [n] | \mathbf{g}_i[j]=1\}} w_i - \sum_{i \in \{i \in [n] | \mathbf{g}_i[j]=0\}} w_i \mod 2^{\ell_2}$ and $\ell_2 = \ell_1 + \lceil \log n \rceil$.

III. **Broadcast**:

1) S_0 broadcasts $\mathbf{g}$ to all participants.
2) Each participant P_i updates the local model $\omega \leftarrow \omega - \eta \cdot \mathbf{g}$.

Fig. 4. The complete Sanitizer framework.

communication cost of Sanitizer on the participant side is effectively reduced to that of a plaintext scheme.

3) Sign Similarity Evaluation: In this step, we evaluate the similarity between the i-th local gradient $\langle \mathbf{g}_i \rangle^B$ and the server's gradient $\mathbf{g}_s$ (from S_0). This process is formalized as a component-wise XNOR operation between the two gradients. In particular, S_0 and S_1 locally compute $\langle \mathbf{s}_i \rangle_0^B = \langle \mathbf{g}_i \rangle_0^B \oplus \mathbf{g}_s$ and $\langle \mathbf{s}_i \rangle_1^B = \langle \mathbf{g}_i \rangle_1^B \oplus 1$, respectively. The sum of these shares, $\langle \mathbf{s}_i \rangle_0^B \oplus \langle \mathbf{s}_i \rangle_1^B$, correctly reconstructs $\mathbf{g}_i \bar{\oplus} \mathbf{g}_s$. To generate the sign similarity score for each participant P_i, we further invoke the boolean summation protocol in Algorithm 2 to sum all coordinates of $\langle \mathbf{s}_i \rangle^B$. Consequently, S_0 and S_1 jointly learn $\langle s_i \rangle^{\ell_1}$, where $\ell_1 = \lceil \log(d+1) \rceil + 1$ is the minimal length required to maintain correctness without overflow.

4) Byzantine Participant Filtering: Recall that this step involves filtering Byzantine participants, formalized as the evaluation of a ReLU function. The ReLU function takes $s_i - t$ as input, where t is a pre-defined threshold. It outputs $w_i = s_i - t$ if s_i is larger than the threshold t, and 0 otherwise. These outputs, $\{w_i\}_{i \in [n]}$, are then used as weights for the participants in the subsequent aggregation process, further eliminating the influence of potentially Byzantine participants. While this operation needs to be performed n times (e.g., for n ranging from 10 to 100), it constitutes a negligible portion of the overall overhead. Our performance breakdown discussed in Section V-C also confirms this claim. Therefore, we choose to directly adopt the state-of-the-art ReLU protocol proposed in [53]. The detailed protocol of ReLU is provided in Algorithm 4 in Appendix A.

5) Weighted Gradient Aggregation: In this step, we perform the weighted gradient aggregation by directly invoking the specialized weighted boolean majority vote protocol detailed in Algorithm 3. This protocol is designed to produce the aggregated gradient $\mathbf{g}'$, which is then revealed to S_0. Subsequently, S_0 computes the final aggregated sign gradient $\mathbf{g} = \mathsf{BSign}(\mathbf{g}')$. Formally, for each $j \in [d]$, $\mathbf{g}'[j]$ is calculated as

$$\mathbf{g}'[j] = \left(\sum_{i \in \{i \in [n] | \mathbf{g}_i[j]=1\}} w_i - \sum_{i \in \{i \in [n] | \mathbf{g}_i[j]=0\}} w_i \right) \mod 2^{\ell_2}. \quad (5)$$

Here, w_i is the weight computed for participant i during the Byzantine participant filtering step. The bitlength $\ell_2 = \ell_1 + \lceil \log n \rceil$ is specifically chosen to accommodate the summation of n weights, ensuring that the result $\mathbf{g}'[j]$ does not overflow the underlying ring $\mathbb{Z}_{2^{\ell_2}}$. This weighted aggregation effectively consolidates the reliable local gradients while diminishing the influence of potentially malicious ones.

Theorem 3: The complete Sanitizer protocol in Figure 4 securely achieves the private and robust aggregation in Algorithm 1 against semi-honest adversaries in the $(\mathcal{F}_{\mathsf{BoolSum}}, \mathcal{F}_{\mathsf{ReLU}}, \mathcal{F}_{\mathsf{WeightVote}})$-hybrid model, assuming that PRF is a pseudo-random function.

A sketch of our security proof is provided as follows. The simulator interacts with the adversaries by invoking the ideal functionalities' simulators, specifically $\mathcal{F}_{\mathsf{BoolSum}}$, $\mathcal{F}_{\mathsf{ReLU}}$, and $\mathcal{F}_{\mathsf{WeightVote}}$, and appends their respective outputs to its view. Briefly, the view generated by the simulator is computationally indistinguishable from the view of a real-world execution. This

indistinguishability is guaranteed by two key properties: the inherent indistinguishability properties of the underlying sub-protocol simulators and the pseudorandomness of PRF.

V. Discussion

A. Extend to Large Language Models

Privacy and robustness in the deployment of Large Language Models (LLMs) have gained significant research traction [62]. Applying the proposed scheme to federated learning for LLMs presents significant challenges. Specifically, as our work is built upon the SignSGD framework, a primary concern when scaling to LLMs is whether the SignSGD-style aggregation remains effective. Currently, Low-Rank Adaptation (LoRA) is the prevailing paradigm for federated fine-tuning of LLMs [63]. The core principle of LoRA lies in the low-rankness of parameter updates, which optimizes two compact low-rank matrices instead of the entire high-dimensional weight space. Since LoRA matrices are already highly condensed, the magnitude of each parameter is critical for reconstructing the original weight updates. Applying sign-based quantization to these already-lean matrices may introduce significant reconstruction errors. Consequently, designing customized, efficient, and secure schemes tailored for LLM federated learning remains a formidable challenge, which we defer to future work.

B. Convergence Analysis

By acting as a denoising and importance sampling filter, our Byzantine filtering and weighted aggregation steps promote more stable global convergence. While theoretically prone to minor information loss, any potential slowdown in convergence is empirically offset by the resulting gains in model precision and generalization, as evidenced by our experimental results and existing research [27]. Specifically, the filtering step identifies and prunes extreme outliers or high-noise updates, even in benign settings, thereby promoting a smoother loss trajectory and mitigating erratic training oscillations. Conversely, the weighted aggregation scheme prioritizes high-fidelity updates that align with the global consensus, thereby accelerating convergence toward a more generalizable local minimum in heterogeneous environments. More detailed analysis can be found in Ref. [27].

C. Extend to Other Training Algorithms

our cryptographic pipeline is technically agnostic to the underlying model training algorithm. This is because the training process is conducted entirely on the participant's side, which obviates the need for specialized cryptographic protocols to secure the local computation. Consequently, cryptographic protection is only required during the gradient aggregation phase. In the classic SignSGD framework [8], [25], [26], raw gradients are computed locally via standard SGD, after which only their sign bits are uploaded for aggregation. Adhering to this paradigm, our work employs standard SGD for local training. There is currently no convincing theory to prove whether applying SignSGD-style aggregation to local models trained using Momentum or Adam can achieve satisfactory convergence [25].

VI. Evaluation

A. Experimental Setup

1) Implementation: We employ two distinct codebases for the implementation of Sanitizer: the cryptographic protocols are developed in C++, relying on the EMP-tookit [64] for COT protocols, while the FL experiments are conducted in Python with Pytorch. To seamlessly integrate these two modules, we employ the SWIG tool,[3] which provides an interface for Python to invoke the encapsulated C++ protocols. Consistent with prior work [7], we simulate a network environment for the two servers featuring a 10 Gbps bandwidth and a 0.2 ms echo latency. All experiments are performed on servers equipped with Intel(R) Xeon(R) Silver 4214 CPUs @ 2.20GHz, and are configured as single-threaded processes to facilitate accurate quantification.

2) Datasets and Model Architectures: Our experimental evaluations are performed using the MNIST and CIFAR-10 datasets. As global models, we employ a convolutional neural network, LeNet [65], and a widely adopted residual neural network, ResNet18 [58], respectively. The entire dataset (either MNIST or CIFAR-10) is uniformly distributed among all participants.

3) Evaluated Byzantine Attacks: We assess the robustness of our scheme against two prominent Byzantine attacks: the sign flipping attack [8] and the stochastic sign attack [8]. In the former, malicious participants invert the sign of their entire gradient. In the latter, adversaries randomise the sign of each individual coordinate of the gradient. Following prior work [27], by default, we assume the server holds only 100 local samples. And our experiments are configured with 30 participants, where 30% of them are designated as malicious.

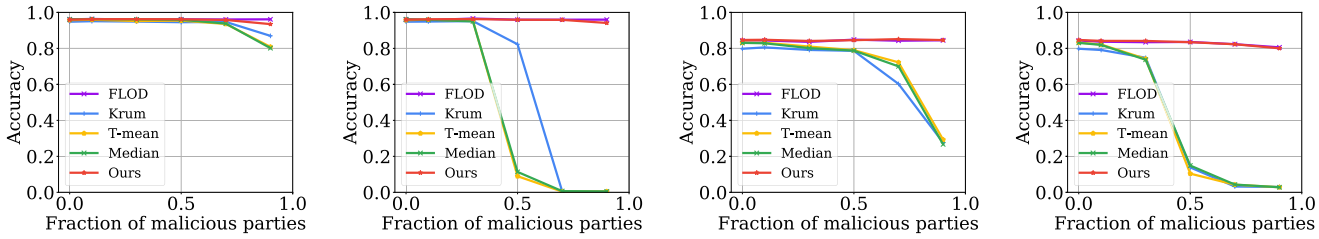
B. Protocol Performance

In this section, we analyze the performance of our private and robust aggregation protocol, specifically focusing on the overhead incurred by the two servers. It is important to note that participants' operations within Sanitizer remain identical to those in vanilla sign-based FL.

1) Overhead Comparison With Prior Art: Table II presents an overhead comparison between Sanitizer and the state-of-the-art work, FLOD [7]. Given the absence of open-source implementations for FLOD, its reported costs are directly sourced from [7]. For a fair comparison, Sanitizer is evaluated under identical conditions regarding gradient dimension, number of participants, and network bandwidth. Our results demonstrate that the runtime of Sanitizer is remarkably $19 \sim 23\times$ faster than that of FLOD, while its communication cost is substantially lower, by a factor of $50\times$ to $63\times$. This performance improvement is primarily attributable to our customized private and robust aggregation protocol and adoption of crypto-friendly algorithm variants.

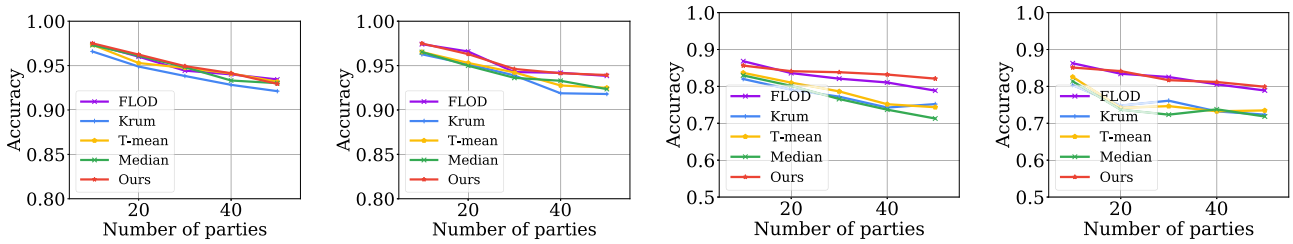
2) Accuracy Comparison With Plaintext: Table III shows a comparative analysis of model accuracy between Sanitizer and its plaintext counterpart (namely, Algorithm 1). This table additionally provides the non-uniform bitlengths employed

[3] https://www.swig.org/



(a) MNIST and Stoch. sign attack (b) MNIST and Sign flipping attack (c) CIFAR10 and Stoch. sign attack (d) CIFAR10 and Sign flipping attack

Fig. 5. Byzantine robustness comparison with prior art. The number of participants is fixed as 20.



(a) MNIST and Stoch. sign attack (b) MNIST and Sign flipping attack (c) CIFAR10 and Stoch. sign attack (d) CIFAR10 and Sign flipping attack

Fig. 6. Impact of the number of participants on accuracy. The fraction of malicious participants is fixed as 0.3.

within our private and robust aggregation protocol. From these results, it is observed that Sanitizer consistently achieves an accuracy level identical to that of the plaintext variant. This is a direct consequence of our precise protocol designs, which maintain numerical integrity throughout the secure computations, coupled with the reasonable selection and configuration of bitlengths to prevent information loss or overflow.

3) Overhead Breakdown: Table IV reports the detailed communication and runtime costs associated with each step of our private and robust aggregation protocol, as described in Section IV-B. An observation is that the sign similarity evaluation (step 2) and weighted gradient aggregation (step 4) constitute the dominant components of the overall overhead. This is due to the fact that the complexity of these specific protocols scales proportionally with both the number of participants n and the dimension of the gradients d. Furthermore, the substantial overhead reduction achieved by Sanitizer compared to prior work primarily stems from the customized protocol optimizations implemented for these two computationally expensive operations. In addition, observe that the transmission overhead incurred by individual participants remains quite low, as our design maintains the same communication efficiency as that of the plaintext SignSGD.

C. Byzantine Robustness

In this section, we present a comprehensive robustness comparison of Sanitizer against several prominent prior methods in Byzantine-robust FL. Specifically, our evaluation includes well-established aggregation rules such as Krum [66], Trimmed mean (T-mean) [67], Median [67], and the recent privacy-preserving FL scheme FLOD [7].

1) Impact of the Fraction Of Malicious Participants: Figure 5 shows the accuracy performance of these Byzantine-robust works under the sign flipping attack and stochastic sign attack on MNIST and CIFAR10 datasets. Throughout these experiments, the total number of participants is fixed at 20. We observe that Sanitizer is Byzantine robustness against both attack types, maintaining high accuracy even when a majority, specifically 90%, of participants are malicious. Moreover, Sanitizer achieves accuracy comparable to that of FLOD because of their similar sign-based defences. In contrast, other methods remain significantly vulnerable to Byzantine attacks when the malicious participant fraction exceeds 50%. In such scenarios, a malicious fraction reaching 90% would render the entire FL system entirely useless. Notably, while the sign flipping attack typically poses a greater threat to prior methods than the stochastic sign attack, Sanitizer exhibits considerable robustness to both attacks. For example, on the MNIST dataset, Sanitizer achieves only 0.13% and 0.18% accuracy loss when 70% of participants are compromised by sign flipping and stochastic sign adversaries, respectively. In addition, an increase in the proportion of Byzantine participants leads to a slight decrease in our method's accuracy, primarily due to the reduced contribution of valid training data from honest participants.

2) Impact of The Number Of Participants: We also explore the impact of the number of participants on model accuracy, as presented in Figure 6. For this evaluation, the fraction of malicious participants is consistently fixed at 30%. Our observations indicate Sanitizer consistently achieves higher or comparable accuracy compared to prior methods across varying numbers of participants, under both sign flipping and stochastic sign attacks on the MNIST and CIFAR-10 datasets. Moreover, a notable trend shows that model accuracy in Sanitizer tends to decrease as the number of participants increases. This phenomenon can be attributed to the consequent reduction in the amount of training data available to each individual participant, which can naturally affect the overall learning efficacy.

VII. Related Works

This section reviews the latest FL processes that explore unified solutions addressing both privacy leakages and Byzantine attacks.

A growing body of research [14], [15], [16], [17], [18] has begun exploring integrated defense mechanisms designed to mitigate privacy leakage and Byzantine attacks simultaneously. For instance, He et al. [14] coupled additive secret sharing with a variant of the Krum aggregation rule [66], to mitigate these dual threats. Similarly, So et al. [15] developed a scheme based on Krum, albeit utilizing distinct cryptographic primitives such as verifiable Shamir's secret sharing and Reed-Solomon codes. To further enhance Byzantine robustness, MLGuard [17] introduced a defense mechanism leveraging cosine similarity and generic secure multi-party computation. Unlike Krum-based approaches [14], [15] that rely exclusively on Euclidean distance, cosine-based metrics incorporate the orientation of participant updates, thereby providing more granular filtering against adversarial inputs.

Expanding this scope, Nguyen et al. [18] introduced FLGuard, which integrates comprehensive privacy protections with specialized defenses against backdoor attacks [68] by combining clustering-based cosine similarity with secure computation. A critical drawback of these prior approaches, however, is their prohibitive communication and computational overhead, which scales quadratically with the number of participants. This bottleneck arises from the necessity of performing pairwise Byzantine statistical analysis, where every participant's gradient must be compared against all others. To address this scalability issue, Hao et al. proposed SecureFL [10], extending the FLTrust framework [27] to a privacy-preserving setting while achieving linear complexity. Similarly, ELSA [21] utilizes customized, participant-assisted secure computation protocols to filter malicious updates based on the ℓ_2-norm metric efficiently. In addition, other efforts [15], [19], [20], [30] have explored defense solutions leveraging zero-knowledge proof [69] in single-server configurations. However, these approaches remain generally inefficient for large-scale deployment [21].

However, all of the above schemes are difficult to extend to our sign-based FL framework, primarily due to two fundamental limitations. On the one hand, the direct application of cryptographic protocols employed by these prior works would inherently compromise the substantial communication advantages offered by SignSGD, because SignSGD's efficiency stems from its ability to transmit only the signs of gradients. On the other hand, our specific setting, which relies on signed gradients, renders most existing aggregation algorithms either directly incompatible or results in suboptimal robustness against Byzantine attacks, a finding empirically supported by the results presented in our experimental section.

Consequently, to overcome these limitations and realize the full potential of sign-based FL in a secure and robust manner, we design customized protocols for implementing Byzantine-robust and privacy-preserving sign-based FL. Among existing research, FLOD [7] is the most relevant work, as it also constructs a private and robust FL scheme by leveraging a sign-based Hamming metric. However, our approach fundamentally differs in its underlying cryptographic techniques. More precisely, FLOD relies on computationally heavy cryptographic tools, namely garbled circuits for boolean operations and homomorphic encryption for arithmetic operations. As a result, these choices inevitably incur significant communication and computational costs. In contrast, our experimental benchmarks demonstrate that our protocol design achieves a substantial efficiency improvement, i.e., a $19 \sim 23\times$ speedup over FLOD.

It is crucial to emphasize that our FL scheme relying on 2PC protocols, is fundamentally distinct from 2PC-based secure neural network training solutions, such as [39], [40], and [70]. Generally speaking, the latter aims to perform secure model training between two servers on secret-shared samples from data owners. We demonstrate two key differences, concerning both Byzantine robustness and efficiency. (1) *Lack of Byzantine robustness:* All existing secure 2PC training works do not ensure robustness against Byzantine adversaries, i.e., malicious corrupted data owners. In fact, these corrupted entities may provide poisoned samples to destroy the usability of the 2PC-trained model [71]. Even worse, it may be difficult to integrate existing defenses [71], [72], [73] into 2PC frameworks due to complex optimization-based strategies. *(2) Prohibitively high overhead:* Even the state-of-the-art 2PC-based training schemes incur prohibitively high overhead. For example, Quotient [40] requires 1901 hours (about 79 days) for training a 3-layer fully-connected neural network on MNIST to achieve desirable accuracy. Rather, our FL scheme is efficient and gains several orders of magnitude improvement over 2PC-based training. The main reason is that we only evaluate private and robust aggregation with secure protocols, while the complex training process is performed by individual participant in a plaintext context.

VIII. Conclusion

In this paper, we have presented Sanitizer, a sign-based FL framework that achieves efficiency, privacy protection, and Byzantine robustness at the same time. At the heart of our constructions lie two new privacy-preserving building blocks, i.e., high-dimensional boolean summation and weighted boolean majority vote, which may be of independent interest. We have performed extensive evaluations, and the results show that Sanitizer outperforms the state-of-the-art by up to $23\times$ while achieving superior Byzantine robustness. Our scheme focuses on secure and robust FL for traditional models, which faces scalability constraints when extended to Large Language Models (LLMs). Due to the massive parameter scale of LLMs, the computational and communication overhead required for our cryptographic primitives and gradient filtering becomes prohibitive. Furthermore, adapting our robust aggregation to parameter-efficient fine-tuning methods (e.g., LoRA) remains a non-trivial challenge for future work.

Appendix

A. Supporting Protocols

For completeness, we give the detailed protocols of the functionalities introduced in Section II-E3.

Algorithm 4 Secure ReLU Protocol, Π_{ReLU}

Input: S_0 and S_1 hold $\langle x\rangle_0^\ell$ and $\langle x\rangle_1^\ell$, respectively.

Output: S_0 and S_1 learn $\langle y\rangle_0^\ell$ and $\langle y\rangle_1^\ell$, respectively, s.t. $y = \mathsf{ReLU}(x)$.

1: S_0 and S_1 parse $\langle x\rangle_0^\ell$ and $\langle x\rangle_1^\ell$ as $\mathrm{msb}_0\|x_0$ and $\mathrm{msb}_1\|x_1$, respectively, where $\mathrm{msb}_b \in \mathbb{Z}_2$ and $x_b \in \mathbb{Z}_{2^{\ell-1}}$ for $b \in \{0,1\}$.
2: S_0 and S_1 invoke an instance of $\mathcal{F}_{\mathsf{Mill}}$ with input $2^{\ell-1} - 1 - x_0$ and x_1, and learn $\langle a\rangle_0^B$ and $\langle a\rangle_1^B$.
3: S_0 and S_1 set $\langle b\rangle_0^B = \mathrm{msb}_0 \oplus \langle a\rangle_0^B$ and $\langle b\rangle_1^B = 1 \oplus \mathrm{msb}_1 \oplus \langle a\rangle_1^B$.
4: S_0 and S_1 invoke an instance of $\mathcal{F}_{\mathsf{BmulA}}$ with inputs $\langle x\rangle_0^\ell, \langle b\rangle_0^B$ from S_0 and $\langle x\rangle_1^\ell, \langle b\rangle_1^B$ from S_1, learn $\langle y\rangle_0^\ell$ and $\langle y\rangle_1^\ell$.
5: S_0 and S_1 output $\langle y\rangle_0^\ell$ and $\langle y\rangle_1^\ell$, respectively.

Algorithm 5 Secure MSB-Known Extension Protocol, Π_{Ext}

Input: S_0 and S_1 hold $\langle x\rangle_0^\ell$ and $\langle x\rangle_1^\ell$, respectively, where $\mathsf{MSB}(x) = 0$.

Output: S_0 and S_1 learn $\langle y\rangle_0^{\ell'}$ and $\langle y\rangle_1^{\ell'}$, respectively, s.t. $y = x$.

1: S_0 and S_1 invoke $\mathcal{F}_{\mathsf{AND}}(\neg\mathsf{MSB}(\langle x\rangle_0^\ell), \neg\mathsf{MSB}(\langle x\rangle_1^\ell))$, and learn $\langle z\rangle^B$.
2: S_0 and S_1 set $\langle w\rangle_0^B = \langle z\rangle_0^B \oplus 1$ and $\langle w\rangle_1^B = \langle z\rangle_1^B$, respectively, where $w = \mathsf{MSB}(\langle x\rangle_0^\ell) \vee \mathsf{MSB}(\langle x\rangle_1^\ell)$.
3: S_0 and S_1 invoke $\mathcal{F}_{\mathsf{B2A}}\left(\langle w\rangle^B\right)$ and learn $\langle w\rangle^{\ell'-\ell}$.
4: For $b \in \{0,1\}, S_b$ outputs $\langle y\rangle_b^{\ell'} = \langle x\rangle_b^\ell - 2^\ell \cdot \langle w\rangle_b^{\ell'-\ell} \bmod 2^{\ell'}$.

1) ReLU: Assuming the functionalities $\mathcal{F}_{\mathsf{Mill}}$ and $\mathcal{F}_{\mathsf{BmulA}}$ exist, we present the ReLU protocol in Algorithm 4, which is proposed in [36] and further improved by [53] using the silent OT extension primitive. To elaborate, the functionality $\mathcal{F}_{\mathsf{Mill}}$ takes as input $x \in \{0,1\}^\ell$ from S_0 and $y \in \{0,1\}^\ell$ from S_1, and returns $\langle z\rangle^B$ such that $z = 1\{x < y\}$. The functionality $\mathcal{F}_{\mathsf{BmulA}}$ takes as input $\langle x\rangle_0^\ell \in \mathbb{Z}_{2^\ell}, \langle y\rangle_0^B \in \{0,1\}$ from S_0 and $\langle x\rangle_1^\ell \in \mathbb{Z}_{2^\ell}, \langle y\rangle_1^B \in \{0,1\}$ from S_1, and returns $\langle z\rangle^\ell$ such that $z = x \cdot y$.

2) MSB-Known Extension: Assuming the functionalities $\mathcal{F}_{\mathsf{AND}}$ and $\mathcal{F}_{\mathsf{B2A}}$ exist, we present the MSB-known extension protocol in Algorithm 5, which is proposed in [37]. We further improve it using the silent OT extension primitive, rather than the IKNP OT extension in [37]. Our optimized protocol only requires $\ell' - \ell + 3$ bits of communication. Here, the functionality $\mathcal{F}_{\mathsf{AND}}$ takes as input $x \in \{0,1\}$ from S_0 and $y \in \{0,1\}$ from S_1, and returns $\langle z\rangle^B$ such that $z = x \cdot y$. The functionality $\mathcal{F}_{\mathsf{B2A}}$ takes as input boolean shares $\langle x\rangle^B$ and outputs arithmetic shares of the same value, i.e., $\langle x\rangle^\ell$.

B. Security Proofs

We give a security proof of Theorem 2.

Proof: We first construct a simulator Sim to simulate the view of corrupted S_0 (the simulator for S_1 should be the same). Sim proceeds as follows: It calls the $\mathcal{F}_{\mathsf{COT}}$ simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{COT}}}(\langle \mathbf{a}[i]\rangle^B)$ for $i \in [d]$, and simulates the bitlength extension by calling the $\mathcal{F}_{\mathsf{Ext}}$ simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Ext}}}(\langle \mathbf{z}[i]\rangle^{\ell'})$ for $i \in [k]$, and appends its output to the view.

Then, we argue that the simulated view is computationally indistinguishable to the real one. We formally show the simulation by proceeding the sequence of hybrid arguments H_0, H_1, H_2, where H_0 is the real world view and H_2 is the simulated view generated by Sim.

- Hybrid_0: The first hybrid is the real interaction described in Π_{BoolSum}. Let H_0 denote the real view.
- Hybrid_1: Let H_1 be the same as H_0, except the $\mathcal{F}_{\mathsf{COT}}$ execution is replaced with running the $\mathsf{Sim}_{\mathcal{F}_{\mathsf{COT}}}(\langle \mathbf{a}[i]\rangle^B)$ for $i \in [d]$. Because $\mathsf{Sim}_{\mathcal{F}_{\mathsf{COT}}}$ is guaranteed to produce output computationally indistinguishable from real, H_1 and H_0 are indistinguishable.
- Hybrid_2: Let H_2 be the same as H_1, except the $\mathcal{F}_{\mathsf{Ext}}$ execution is replaced with running the $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Ext}}}(\langle \mathbf{Z}[i]\rangle^{\ell'})$ for $i \in [k]$. The indistinguishability of the underlying simulator $\mathsf{Sim}_{\mathcal{F}_{\mathsf{Ext}}}$ guarantees that H_2 and H_1 are indistinguishable.

In summary, H_2 is identically distributed to the simulator's output, completing the proof.■

REFERENCES

[1] S. Pichai, "Privacy should not be a luxury good," The New York Times, New York, NY, USA, Tech. Rep., 2019.
[2] B. Li, Y. Wu, J. Song, R. Lu, T. Li, and L. Zhao, "DeepFed: Federated deep learning for intrusion detection in industrial cyber–physical systems," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5615–5624, Aug. 2021.
[3] P. Kairouz et al., "Advances and open problems in federated learning," 2019, *arXiv:1912.04977*.
[4] *Google's Tensorflow Federated*. Accessed: May 3, 2026. [Online]. Available: https://www.tensorflow.org/federated
[5] *Openmined's Pysyft*. Accessed: May 3, 2026. [Online]. Available: https://blog.openmined.org/tag/pysyft
[6] *Webank's Fate*. Accessed: May 3, 2026. [Online]. Available: https://fate.fedai.org
[7] Y. Dong, X. Chen, K. Li, D. Wang, and S. Zeng, "FLOD: Oblivious defender for private Byzantine-robust federated learning with dishonest-majority," in *Proc. ESORICS*, 2021, p. 993.
[8] J. Bernstein, J. Zhao, K. Azizzadenesheli, and A. Anandkumar, "SignSGD with majority vote is communication efficient and fault tolerant," in *Proc. ICLR*, 2018.
[9] F. Tramèr et al., "Truth serum: Poisoning machine learning models to reveal their secrets," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 2779–2792.
[10] M. Hao, H. Li, G. Xu, H. Chen, and T. Zhang, "Efficient, private and robust federated learning," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2021, pp. 45–60.
[11] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 603–618.
[12] M. Fang, X. Cao, J. Jia, and N. Z. Gong, "Local model poisoning attacks to Byzantine-robust federated learning," in *Proc. USENIX Secur.*, 2019, pp. 1605–1622.
[13] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *Proc. ICML*, 2018, pp. 634–643.
[14] L. He, S. Praneeth Karimireddy, and M. Jaggi, "Secure Byzantine-robust machine learning," 2020, *arXiv:2006.04747*.
[15] J. So, B. Guler, and A. S. Avestimehr, "Byzantine-resilient secure federated learning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 2168–2181, Jul. 2021.
[16] H. Hashemi, Y. Wang, C. Guo, and M. Annavaram, "Byzantine-robust and privacy-preserving framework for FedML," in *Proc. ICLR Workshop Secur. Saf. Mach. Learn. Syst.*, 2021.
[17] Y. Khazbak, T. Tan, and G. Cao, "MLGuard: Mitigating poisoning attacks in privacy preserving distributed collaborative learning," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2020, pp. 1–9.
[18] T. Duc Nguyen et al., "FLAME: Taming backdoors in federated learning (Extended version 1)," 2021, *arXiv:2101.02281*.
[19] H. Lycklama, L. Burkhalter, A. Viand, N. Küchler, and A. Hithnawi, "RoFL: Robustness of secure federated learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2023, pp. 453–476.
[20] J. Bell et al., "ACORN: Input validation for secure aggregation," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 4805–4822.

[21] M. Rathee, C. Shen, S. Wagh, and R. A. Popa, "ELSA: Secure aggregation for federated learning with malicious actors," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2023, pp. 1961–1979.

[22] M. Hao, H. Li, X. Luo, G. Xu, H. Yang, and S. Liu, "Efficient and privacy-enhanced federated learning for industrial artificial intelligence," *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6532–6542, Oct. 2020.

[23] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1333–1345, May 2018.

[24] K. Bonawitz et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1175–1191.

[25] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "SignSGD: Compressed optimisation for non-convex problems," in *Proc. ICML*, 2018, pp. 559–568.

[26] S. P. Karimireddy, Q. Rebjock, S. U. Stich, and M. Jaggi, "Error feedback fixes SignSGD and other gradient compression schemes," in *Proc. ICML*, vol. 97, 2019, pp. 3252–3261.

[27] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "FLTrust: Byzantine-robust federated learning via trust bootstrapping," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021.

[28] T. Gehlhar, F. Marx, T. Schneider, A. Suresh, T. Wehrle, and H. Yalame, "SafeFL: MPC-friendly framework for private and robust federated learning," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2023, pp. 69–76.

[29] G. N. Rothblum, E. Omri, J. Chen, and K. Talwar, "Pine: Efficient verification of a Euclidean norm bound of a secret-shared vector," in *Proc. USENIX Secur.*, 2024, pp. 6975–6992.

[30] A. Roy Chowdhury, C. Guo, S. Jha, and L. van der Maaten, "EIFFeL: Ensuring integrity for federated learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 2535–2549.

[31] O. Franzese et al., "Robust and actively secure serverless collaborative learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 39504–39528.

[32] *Federated Learning: Collaborative Machine Learning Without Centralized Training Data*. Accessed: May 3, 2026. [Online]. Available: https://research.google/blog/federated-learning-collaborative-machine-learning-without- centralized-training-data/

[33] N. Chandran, D. Gupta, S. L. B. Obbattu, and A. Shah, "Simc: ML inference secure against malicious clients at semi-honest cost," in *Proc. USENIX Secur.*, 2022, pp. 1361–1378.

[34] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "MUSE: Secure inference resilient to malicious clients," in *Proc. USENIX Secur.*, vol. 2021, 2021, pp. 1040–2218.

[35] Y. Lindell, "How to simulate it—A tutorial on the simulation proof technique," in *Tutorials on the Foundations of Cryptography*. Cham, Switzerland: Springer, 2017, pp. 277–346.

[36] D. Rathee et al., "CrypTFlow2: Practical 2-party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 325–342.

[37] D. Rathee et al., "SiRnn: A math library for secure RNN inference," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 1003–1020.

[38] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *Proc. USENIX NSDI*, 2017, pp. 259–282.

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

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

[41] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou, "Prio+: Privacy preserving aggregate statistics via Boolean shares," in *Proc. SCN*, 2022, pp. 516–539.

[42] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight techniques for private heavy hitters," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 762–776.

[43] E. Dauterman, M. Rathee, R. A. Popa, and I. Stoica, "Waldo: A private time-series database from function secret sharing," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 2450–2468.

[44] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh, "Express: Lowering the cost of metadata-hiding communication with cryptographic privacy," in *Proc. USENIX Secur.*, 2019, pp. 1775–1792.

[45] *Privacy-Preserving Collection of Aggregated Statistics in the Mozilla's Firefox Browser*. Accessed: May 3, 2026. [Online]. Available: https://blog.mozilla.org/security/

[46] *COVID-19 Exposure Notification Analytics*. Accessed: May 3, 2026. [Online]. Available: https://github.com/google/exposure-notifications-android

[47] *Digital Asset Wallets*. Accessed: May 3, 2026. [Online]. Available: https://sepior.com/products/advanced-mpc-wallet

[48] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (Poly)Logarithmic overhead," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 1253–1269.

[49] D. Demmler, T. Schneider, and M. Zohner, "ABY-A framework for efficient mixed-protocol secure two-party computation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015.

[50] M. O. Rabin, "How to exchange secrets with oblivious transfer," Cryptol. ePrint Arch., Tech. Rep. 2005/187, Jun. 1981, p. 187.

[51] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Proc. CRYPTO*, 2003, pp. 145–161.

[52] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated OT with small communication," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 1607–1626.

[53] Z. Huang, W.-J. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proc. USENIX Secur.*, 2022, pp. 809–826.

[54] L. Yang et al., "Enhanced model poisoning attack and multi-strategy defense in federated learning," *IEEE Trans. Inf. Forensics Security*, vol. 20, pp. 3877–3892, 2025.

[55] L. Sperling and S. Kulkarni, "SONNI: Secure oblivious neural network inference," in *Proc. 22nd Int. Conf. Secur. Cryptography*, 2025, pp. 515–522.

[56] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "Aby2. 0: Improved mixed-protocol secure two-party computation," in *Proc. USENIX Secur.*, 2021, pp. 2165–2182.

[57] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proc. ACM CCS*, 2018, pp. 35–52.

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[59] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proc. CRYPTO*, 2007, pp. 420–432.

[60] R. Cramer and V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack," in *Proc. Annu. Int. Cryptol. Conf.*, 1998, pp. 13–25.

[61] M. Dworkin, "Recommendation for block cipher modes of operation: The CMAC mode for authentication," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. NIST SP 800-38B, 2005.

[62] H. Chen, M. Hao, H. Li, P. Xing, G. Xu, and T. Zhang, "Iron: Private inference on transformers," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 15718–15731.

[63] J. E. Hu et al., "LoRA: Low-rank adaptation of large language models," in *Proc. ICLR*, 2021, p. 3.

[64] *Emp-Tookit*. Accessed: May 3, 2026. [Online]. Available: https://github.com/emp-toolkit

[65] Y. LeCun et al., "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[66] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. NeurIPS*, vol. 30, 2017, pp. 118–128.

[67] D. Yin, Y. Chen, K. Ramchandran, and P. L. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. ICML*, 2018, pp. 5650–5659.

[68] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proc. AISTATS*, 2018, pp. 2938–2948.

[69] M. Hao et al., "Scalable zero-knowledge proofs for non-linear functions in machine learning," in *Proc. USENIX Secur. Symp.*, 2024, pp. 3819–3836.

[70] M. Keller and K. Sun, "Secure quantized training for deep learning," in *Proc. ICML*, 2021, pp. 10912–10938.

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

[72] A. Mehra, B. Kailkhura, P.-Y. Chen, and J. Hamm, "How robust are randomized smoothing based defenses to data poisoning?," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 13239–13248.

[73] J. Steinhardt, P. W. Koh, and P. Liang, "Certified defenses for data poisoning attacks," in *Proc. NeurIPS*, 2017.