# Secure Byzantine-Robust Machine Learning

**Lie He**
MLO, EPFL
`lie.he@epfl.ch`

**Sai Praneeth Karimireddy**
MLO, EPFL
`sai.karimireddy@epfl.ch`

**Martin Jaggi**
MLO, EPFL
`martin.jaggi@epfl.ch`

## Abstract

Increasingly machine learning systems are being deployed to edge servers and devices (e.g. mobile phones) and trained in a collaborative manner. Such distributed/federated/decentralized training raises a number of concerns about the robustness, privacy, and security of the procedure. While extensive work has been done in tackling with robustness, privacy, or security individually, their combination has rarely been studied. In this paper, we propose a secure two-server protocol that offers both input privacy and Byzantine-robustness. In addition, this protocol is communication-efficient, fault-tolerant and enjoys local differential privacy.

## 1 Introduction

Recent years have witnessed fast growth of successful machine learning applications based on data collected from decentralized user devices. Unfortunately, however, currently most of the important machine learning models on a societal level do not have their utility, control, and privacy aligned with the data ownership of the participants. This issue can be partially attributed to a fundamental conflict between the two leading paradigms of traditional centralized training of models on one hand, and decentralized/collaborative training schemes on the other hand.

When applied to datasets containing personal data, the use of privacy-preserving techniques is currently required under regulations such as the GDPR or HIPAA. The idea of training models on decentralized datasets and incrementally aggregating model updates via a central server motivates the federated learning paradigm [18]. However, the averaging in federated learning, when viewed as a *multi-party computation (MPC)*, does not preserve the *input privacy* because the server observes the models directly. The *input privacy* requires each party learns nothing more than the output of computation which in this paradigm means the aggregated model updates. To solve this problem, *secure* aggregation rules as proposed in [8] achieve guaranteed input privacy. Such secure aggregation rules have found wider industry adoption recently e.g. by Google on Android phones [7, 23] where input privacy guarantees can offer e.g. efficiency and exactness benefits compared to differential privacy, but both can also be combined.

While centralized training violates the privacy rights of participating users, existing alternative training schemes are typically not Byzantine-robust. The concept of Byzantine robustness has received considerable attention in the past few years for practical applications, as a way to make the training process robust to malicious actors. A Byzantine participant or worker can behave arbitrarily malicious, e.g. sending arbitrary updates to the server. This poses great challenge to the most widely used aggregation rules, e.g. simple average, since a single Byzantine worker can compromise the results of aggregation. A number of Byzantine-robust aggregation rules have been proposed recently [5, 20, 2, 19, 29, 21] and can be used as a building block for our proposed technique.

Combining input privacy and Byzantine robustness, however, has rarely been studied. The work closest to our approach is [22] which tolerates data poisoning but does not offer Byzantine robustness. Prio [10] is a private and robust aggregation system relying on secret-shared non-interactive proofs (SNIP). While their setting is similar to ours, the robustness they offer is limited to check the range of

Presented at SpicyFL Workshop at 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

the input. Besides, the encoding for SNIP has to be affine-aggregable and is expensive for clients to compute.

In this paper, we propose a secure aggregation framework with the help of two non-colluding honest-but-curious servers. This framework also tolerates server-worker collusion. In addition, we combine robustness and privacy at the cost of leaking only worker similarity information which is marginal for high-dimensional neural networks. Note that our focus is not to develop new defenses against state-of-the-art attacks, e.g. [3, 27]. Instead, we focus on making *arbitary* current and future distance-based robust aggregation rules (e.g. (e.g. [19, 22])) compatible with secure aggregation.

**Main contributions.** We propose a novel distributed training framework which is

- **Privacy-preserving:** our method keeps the input data of each user secure against any other user, and against our honest-but-curious servers.
- **Byzantine robust:** our method offers Byzantine robustness and allows to incorporate existing robust aggregation rules, e.g. [5, 2]. The results are exact, i.e. identical to the non-private robust methods.
- **Fault tolerant and easy to use:** our method natively supports workers dropping out or newly joining the training process. It is also easy to implement and to understand for users.
- **Efficient and scalable:** the computation and communication overhead of our method is negligible (less than a factor of 2) compared to non-private methods. Scalability in terms of cost including setup and communication is linear in the number of workers.

## 2 Problem setup, privacy, and robustness

We consider the distributed setup of $n$ user devices, which we call workers, with the help of two additional servers. Each worker $i$ has its own private part of the training dataset. The workers want to collaboratively train a public model benefitting from the joint training data of all participants.

In every training step, each worker computes its own private model update (e.g. a gradient based on its own data) denoted by the vector $\boldsymbol{x}_i$. The aggregation protocol aims to compute the sum $\boldsymbol{z} = \sum_{i=1}^{n} \boldsymbol{x}_i$ (or a robust version of this aggregation), which is then used to update a public model. While the result $\boldsymbol{z}$ is public in all cases, the protocol must keep each $\boldsymbol{x}_i$ private from any adversary or other workers.

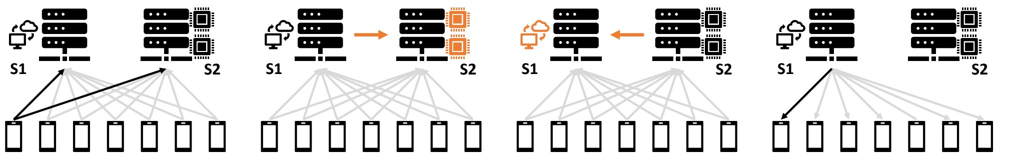
**Security model.** We consider honest-but-curious servers which do not collude with each other but may collude with malicious workers. An honest-but-curious server follows the protocol but may try to inspect all messages. We also assume that all communication channels are secure. We guarantee the strong notion of *input privacy*.

**Byzantine robustness model.** We allow the standard Byzantine worker model which assumes that workers can send arbitrary adversarial messages trying to compromise the process. We assume that a fraction of up to $\alpha$ ($< 0.5$) of the workers is Byzantine, i.e. are *malicious* and not follow the protocol.

**Additive secret sharing.** Secret sharing is a way to split any secret into multiple parts such that no part leaks the secret. Formally, suppose a scalar $a$ is a *secret* and the secret holder shares it with $k$ parties through *secret-shared values* $\langle a \rangle$. In this paper, we only consider additive secret-sharing where $\langle a \rangle$ is a notation for the set $\{a_i\}_{i=1}^{k}$ which satisfy $a = \sum_{p=1}^{k} a_p$, with $a_p$ held by party $p$. Crucially, it must not be possible to reconstruct $a$ from any $a_p$. For vectors like $\boldsymbol{x}$, their secret-shared values $\langle \boldsymbol{x} \rangle$ are simply component-wise scalar secret-shared values.

**Two-server setting.** We assume there are two non-colluding servers: model server (S1) and worker server (S2). S1 holds the output of each aggregation and thus also the machine learning model which is public to all workers. S2 holds intermediate values to perform Byzantine aggregation. Another key assumption is that the servers have no incentive to collude with workers, perhaps enforced via a potential huge penalty if exposed. It is realistic to assume that the communication link between the two servers S1 and S2 is faster than the individual links to the workers. To perform robust aggregation, the servers will need access to a sufficient number of *Beaver's triples*. These are data-independent values required to implement secure multiplication in MPC on both servers, and can be precomputed beforehand. For completeness, the classic algorithm for multiplication is given in in Appendix B.1.

**Byzantine-robust aggregation oracles.** Most of existing robust aggregation algorithms rely on distance measures to identity potential adversarial behavior [5, 29, 19, 17, 12]. All such distance-



(a) **WorkerSecretSharing**: each worker $i$ secret-shares its update $\boldsymbol{x}_i$ locally and uploads them to **S1** and **S2** separately.

(b) **RobustWeightSelection**: Compute and reveal $\{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\}_{i<j}$ on **S2** and select a robust set of indices represented by $\boldsymbol{p} = \{p_i\}_i$ by calling the Byzantine-robust oracle.

(c) **AggregationAndUpdate**: Compute and reveal aggregation $\boldsymbol{z} = \sum_{i=1}^n p_i \boldsymbol{x}_i$ on **S1**. **S1** updates the public model.

(d) **WorkerPullModel**: Each worker $i$ pulls model from **S1**.

Figure 1: Illustration of Algorithm 2. The orange components on servers represent the computation-intensive operations at low communication cost between servers.

based aggregation rules can be directly incorporated into our proposed scheme, making them secure. In addition, our protocol also works with schemes designed for non-iid data such as [17, 12, 13].

## 3 Secure aggregation protocol: two-server model

In this section, we present our protocol for non-robust aggregation and robust aggregation.

### 3.1 Non-robust secure aggregation

In each round, Algorithm 1 consists of two stages:

- **WorkerSecretSharing** (Figure 1a): each worker $i$ randomly splits its private input $\boldsymbol{x}_i$ into two additive secret shares $\boldsymbol{x}_i = \boldsymbol{x}_i^{(1)} + \boldsymbol{x}_i^{(2)}$. This can be done e.g. by sampling a large noise $\xi_i$ and then using $\boldsymbol{x}_i \pm \xi_i$ as the shares. Worker $i$ sends $\boldsymbol{x}_i^{(1)}$ to **S1** and $\boldsymbol{x}_i^{(2)}$ to **S2**. We write $\langle \boldsymbol{x}_i \rangle$ for the secret-shared values distributed over the two servers.
- **AggregationAndUpdate** (Figure 1c): Given some weights $\{p_i\}_{i=1}^n$, each server locally computes $\langle \sum_{i=1}^n p_i \boldsymbol{x}_i \rangle$. Then **S2** sends its share $\langle \sum_{i=1}^n p_i \boldsymbol{x}_i \rangle^{(2)}$ to **S1** so that **S1** can then compute $\boldsymbol{z} = \sum_{i=1}^n p_i \boldsymbol{x}_i$. **S1** updates the public model with $\boldsymbol{z}$.

We now argue about correctness and privacy. It is clear that the output $\boldsymbol{z}$ of the above protocol satisfies $\boldsymbol{z} = \sum_{i=1}^n p_i \boldsymbol{x}_i$, ensuring that all workers compute the right update. For privacy, we track the values stored by each of the servers and workers:

- **S1**: The secret share $\{\boldsymbol{x}_i^{(1)}\}_{i=1}^n$ and the sum of other share $\sum_{i=1}^n \boldsymbol{x}_i^{(2)}$.
- **S2**: The secret share $\{\boldsymbol{x}_i^{(2)}\}_{i=1}^n$.
- Worker $i$: $\boldsymbol{x}_i$ and $\boldsymbol{z} = \sum_{i=1}^n p_i \boldsymbol{x}_i$.

Clearly, the workers have no information other than the aggregate $\boldsymbol{z}$ and their own data. **S2** only has the secret share which on their own leak no information about any data. Hence surprisingly, **S2** learns *no information* in this process. **S1** has its own secret share and also the sum of the other share. If $n = 1$, then $\boldsymbol{z} = \boldsymbol{x}_i$ and hence **S1** is allowed to learn everything. If $n > 1$, then **S1** cannot recover information about any individual secret share $\boldsymbol{x}_i^{(2)}$ from the sum. Thus, **S1** learns $\boldsymbol{z}$ and nothing else.

### 3.2 Robust secure aggregation

We now describe how Algorithm 2 replaces the simple aggregation with any distance-based robust aggregation rule **Aggr**, e.g. Multi-Krum [5]. The key idea is to use two-party MPC to securely compute multiplication.

- **WorkerSecretSharing** (Figure 1a): Same as before.
- **RobustWeightSelection** (Figure 1b): After collecting all secret-shared values $\{\langle \boldsymbol{x}_i \rangle\}_i$, the servers compute pairwise difference $\{\langle \boldsymbol{x}_i - \boldsymbol{x}_j \rangle\}_{i<j}$ locally. **S2** then reveals—to itself exclusively—in plain text all of the pairwise Euclidean distances between workers $\{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\}_{i<j}$ with the help of precomputed Beaver's triples and Algorithm 3. The distances are kept private from **S1** and workers. **S2** then feeds these distances to the distance-based robust aggregation rule **Aggr**, returning (on **S2**) a weight vector $\boldsymbol{p} = \{p_i\}_{i=1}^n$ (a

**Algorithm 1** Two-Server Secure Aggregation (Non-robust variant)

```
Setup: $n$ workers (non-Byzantine) with private vectors $\boldsymbol{x}_i$. Two non-colluding servers S1 and S2.
Workers: (WorkerSecretSharing)
    1. split private $\boldsymbol{x}_i$ into additive secret shares $\langle \boldsymbol{x}_i \rangle = \{\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_i^{(2)}\}$ (such that $\boldsymbol{x}_i = \boldsymbol{x}_i^{(1)} + \boldsymbol{x}_i^{(2)}$)
    2. send $\boldsymbol{x}_i^{(1)}$ to S1 and $\boldsymbol{x}_i^{(2)}$ to S2
Servers:
    1. $\forall\, i$, S1 collects $\boldsymbol{x}_i^{(1)}$ and S2 collects $\boldsymbol{x}_i^{(2)}$
    2. (AggregationAndUpdate):
        (a) On S1 and S2, compute $\langle \sum_{i=1}^{n} \boldsymbol{x}_i \rangle$ locally
        (b) S2 sends its share of $\langle \sum_{i=1}^{n} \boldsymbol{x}_i \rangle$ to S1
        (c) S1 reveals $\boldsymbol{z} = \sum_{i=1}^{n} \boldsymbol{x}_i$ to everyone
```

seleced subset indices can be converted to a vector of binary values), and secret-sharing them with **S1** for aggregation.

- **AggregationAndUpdate** (Figure 1c): Given weight vector $\boldsymbol{p}$ from previous step, we would like S1 to compute $\sum_{i=1}^{n} p_i \boldsymbol{x}_i$. **S2** secret shares with **S1** the values of $\{\langle p_i \rangle\}$ instead of sending in plain-text since they may be private. Then, **S1** reveals to itself, but not to **S2**, the value of $\boldsymbol{z} = \sum_{i=1}^{n} p_i \boldsymbol{x}_i$ using secret-shared multiplication and updates the public model.
- **WorkerPullModel** (Figure 1d): Workers pull and update the public model on **S1**.

The key difference between the robust and the non-robust aggregation scheme is the weight selection phase where **S2** computes all pairwise distances and uses this to run a robust-aggregation rule in a black-box manner. **S2** computes these distances i) without leaking any information to **S1**, and ii) without itself learning anything other than the pair-wise distances (and in particular none of the actual values of $\boldsymbol{x}_i$). To perform such a computation, **S1** and **S2** use precomputed *Beaver's triplets* (Algorithm 3 in the Appendix), which can be made available in a scalable way [25].

### 3.3 Salient features

Overall, our protocols are very resource-light and straightforward from the perspective of the workers. It enjoys advantages in the following aspects:

**Communication overhead.** In applications, individual uplink speed from worker to servers is typically much slower than speed between servers. In each round, our protocols only require two uplink and one downlink which is very small comparing to interactive proofs.

**Fault tolerance.** The workers in Algorithm 1 and Algorithm 2 are completely stateless across multiple rounds and there is no *offline* phase required. This means that workers can start participating in the protocols simply by pulling the latest public model. Further, our protocols are unaffected if some workers drop out in the middle of a round. Unlike in [8], there is no entanglement between workers and we don't face unbounded recovery issues.

**Compatibility with local differential privacy.** One byproduct of our protocol can be used to convert differentially private mechanisms, such as [1] which only of the aggregate model which guarantees privacy, into the stronger *locally* differentially private mechanisms which guarantee user-level privacy.

**Other Byzantine-robust oracles.** We can also use some robust-aggregation rules which are not based on pair-wise distances such as Byzantine SGD [2]. Since the basic structures are very similar to Algorithm 2, we put Algorithm 8 in the appendix.

**Security.** The security of Algorithm 1 is straightforward as we previously discussed. The security of Algorithm 2 again relies on the separation of information between **S1** and **S2** with neither the workers nor **S1** learning anything other than the aggregate $\boldsymbol{z}$. We will next formally prove that this is true even in the presence of malicious workers.

## 4 Theoretical guarantees

### 4.1 Exactness

In the following lemma we show that Algorithm 2 gives the exact same result as non-privacy-preserving version.

**Algorithm 2** Two-Server Secure Robust Aggregation (Distance-Based)

```
Setup: $n$ workers, $\alpha n$ of which are Byzantine. Two non-colluding servers S1 and S2.
Workers: (WorkerSecretSharing)
    1. split private $\boldsymbol{x}_i$ into additive secret shares $\langle \boldsymbol{x}_i \rangle = \{\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_i^{(2)}\}$ (such that $\boldsymbol{x}_i = \boldsymbol{x}_i^{(1)} + \boldsymbol{x}_i^{(2)}$)
    2. send $\boldsymbol{x}_i^{(1)}$ to S1 and $\boldsymbol{x}_i^{(2)}$ to S2
Servers:
    1. $\forall\, i$, S1 collects gradient $\boldsymbol{x}_i^{(1)}$ and S2 collects $\boldsymbol{x}_i^{(2)}$
    2. (RobustWeightSelection):
        (a) For each pair $(\boldsymbol{x}_i,\ \boldsymbol{x}_j)$ compute their Euclidean distance $(i < j)$:
            • On S1 and S2, compute $\langle \boldsymbol{x}_i - \boldsymbol{x}_j \rangle = \langle \boldsymbol{x}_i \rangle - \langle \boldsymbol{x}_j \rangle$ locally
            • Use precomputed Beaver's triples (see Algorithm 3) to compute the
              distance $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$
        (b) S2 perform robust aggregation rule $\boldsymbol{p} =$ Aggr$(\{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\}_{i<j})$
        (c) S2 secret-shares $\langle \boldsymbol{p} \rangle$ with S1
    3. (AggregationAndUpdate):
        (a) On S1 and S2, use MPC multiplication to compute $\langle \sum_{i=1}^n p_i \boldsymbol{x}_i \rangle$ locally
        (b) S2 sends its share of $\langle \sum_{i=1}^n p_i \boldsymbol{x}_i \rangle^{(2)}$ to S1
        (c) S1 reveals $\boldsymbol{z} = \sum_{i=1}^n p_i \boldsymbol{x}_i$ to all workers.
Workers:
    1. (WorkerPullModel): Collect $\boldsymbol{z}$ and update model locally
```

**Lemma 1** (Exactness of Algorithm 2)**.** *The resulting $\boldsymbol{z}$ in Algorithm 2 is identical to the output of the non-privacy-preserving version of the used robust aggregation rule.*

With the exactness of the protocol established, we next focus on the privacy guarantee.

## 4.2 Privacy

We prove information-theoretic notion of privacy which gives the strongest guarantee possible. Formally, we will show that the distribution of the secret does not change even after being conditioned on all observations made by all participants, i.e. each worker $i$, **S1** and **S2**. This implies that the observations carry absolutely no information about the secret. Our results rely on the existence of simple additive secret-sharing protocols as discussed in the Appendix.

Each worker $i$ only receives the final aggregated $\boldsymbol{z}$ at the end of the protocol and is not involved in any other manner. Hence no information can be leaked to them. We will now examine **S1**. The proofs below rely on Beaver's triples which we summarize in the following lemma.

**Lemma 2** (Beaver's triples)**.** *Suppose we secret share $\langle x \rangle$ and $\langle y \rangle$ between **S1** and **S2** and want to compute $xy$ on **S2**. There exists a protocol which enables such computation which uses precomputed shares $BV = (\langle a \rangle, \langle b \rangle, \langle c \rangle)$ such that **S1** does not learn anything and **S2** only learns $xy$.*

Due to the page limit, we put the details about Beaver's triples, multiplying secret shares, as well as the proofs for the next two theorems to the Appendix.

**Theorem I** (Privacy for **S1**)**.** *Let $\boldsymbol{z} = \sum_{i=1}^n p_i \boldsymbol{x}_i$ where $\{p_i\}_{i=1}^n$ is the output of byzantine oracle or a vetor of 1s (non-private). Let $BV_{ij} = \langle \boldsymbol{a}_{ij}, \boldsymbol{b}_{ij}, \boldsymbol{c}_{ij} \rangle$ and $BVp_i = \langle \boldsymbol{a}_i^p, \boldsymbol{b}_i^p, \boldsymbol{c}_i^p \rangle$ be the Beaver's triple used in the multiplications. Let $\langle \cdot \rangle^{(1)}$ be the share of the secret-shared values $\langle \cdot \rangle$ on **S1**. Then for all workers i*

$$\mathbb{P}(\boldsymbol{x}_i = x_i \mid \{\langle \boldsymbol{x}_i \rangle^{(1)}, \langle p_i \rangle^{(1)}\}_{i=1}^n,\ \{BV_{i,j}^{(1)}, \boldsymbol{x}_i - \boldsymbol{x}_j - \boldsymbol{a}_{ij}, \boldsymbol{x}_i - \boldsymbol{x}_j - \boldsymbol{b}_{ij}\}_{i<j},$$
$$\{\langle \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 \rangle^{(1)}\}_{i<j},\ \{BVp_i^{(1)}, p_i - \boldsymbol{a}_i^p, p_i - \boldsymbol{b}_i^p\}_{i=1}^n, \boldsymbol{z}) = \mathbb{P}(\boldsymbol{x}_i = x_i | \boldsymbol{z})$$

*Note that the conditioned values are what **S1** observes throughout the algorithm. $\{BV_{ij}^{(1)}, \boldsymbol{x}_i - \boldsymbol{x}_j - \boldsymbol{a}_{ij}, \boldsymbol{x}_i - \boldsymbol{x}_j - \boldsymbol{b}_{ij}\}_{i<j}$ and $\{BVp_i^{(1)}, p_i - \boldsymbol{a}_i^p, p_i - \boldsymbol{b}_i^p\}_{i=1}^n$ are intermediate values during shared values multiplication.*

On the other hand, **S2** doesn't know the output of aggregation $\boldsymbol{z}$which is more similar to an independent system knowing little about the underlying tasks, model weights, etc. While **S2** has observed many intermediate values, it can only learn no more than the model distances.

**Theorem II** (Privacy for **S2**). *Using same notations and assumptions as Theorem I, for all workers i*

$$\begin{aligned}&\mathbb{P}(\boldsymbol{x}_i = x_i \mid \{\langle \boldsymbol{x}_i \rangle^{(2)}, \langle p_i \rangle^{(2)}, p_i\}_{i=1}^n, \{BV_{i,j}^{(2)}, \boldsymbol{x}_i - \boldsymbol{x}_j - \boldsymbol{a}_{ij}, \boldsymbol{x}_i - \boldsymbol{x}_j - \boldsymbol{b}_{ij}\}_{i<j}, \\ &\quad \{\langle \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 \rangle^{(2)}, \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\}_{i<j}, \{BVp_i^{(2)}, p_i - \boldsymbol{a}_i^p, p_i - \boldsymbol{b}_i^p\}_{i=1}^n) \\ &\quad = \mathbb{P}(\boldsymbol{x}_i = x_i \mid \{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\}_{i<j})\end{aligned} \tag{1}$$

The model distances indeed only leaks similarity among the workers. Such similarity, however, does not tell **S2** information about the parameters.

### 4.3 Combining with differential privacy

While input privacy is our main goal, our approach is naturally compatible with other orthogonal notions of privacy. Global differential privacy (DP) [24, 1, 9] is mainly concerned about the privacy of the *aggregated* model, and whether it leaks information about the training data. On the other hand, local differential privacy (LDP) [11, 14] is stronger notions which is also concerned with the training process itself. It requires that every communication transmitted by the worker does not leak information about their data. In general, it is hard to learn deep learning models satisfying LDP using iterate perturbation (which is the standard mechanism for DP) [8].

Our non-robust protocol *is naturally compatible* with local differential privacy. Consider the usual iterative optimization algorithm which in each round $t$ performs

$$\boldsymbol{p}_t \leftarrow \boldsymbol{p}_{t-1} - \eta(\boldsymbol{x}_t + \nu_t), \text{ where } \boldsymbol{x}_t = \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_{t,i}\,. \tag{2}$$

Here $\boldsymbol{x}_t$ is the aggregate update, $\boldsymbol{p}_t$ is the model parameters, and $\nu_t$ is the noise added for DP [1].

**Theorem III** (from DP to LDP). *Suppose that the noise $\nu_t$ in* (2) *is sufficient to ensure that the set of model parameters $\{\boldsymbol{p}_t\}_{t\in[T]}$ satisfy $(\varepsilon, \delta)$-DP for $\varepsilon \geq 1$. Then, running* (2) *with using Alg. 1 to compute $(\boldsymbol{x}_t + \eta_t)$ by securely aggregating $\{\boldsymbol{x}_{1,t} + n\eta_t, \boldsymbol{x}_{2,t}, \ldots, \boldsymbol{x}_{n,t}\}$ satisfies $(\varepsilon, \delta)$-LDP.*

Unlike existing approaches, we do not face a tension between differential privacy which relies on real-valued vectors and cryptographic tools which operate solely on discrete/quantized objects because our protocols do not rely on primitives like Diffie-Hellman key agreement. In particular, the vectors $\boldsymbol{x}_i$ can be full-precision at the cost of adding marginal rounding error tolerated by robust aggregation rule and stochastic gradient descent algorithms. Thus, our protocol can be integrated with a mechanism with global DP properties e.g. [1], and prove *local* DP guarantees.

## 5 Empirical analysis of overhead

We present an illustrative simulation on a local machine to demonstrate the overhead of our scheme. We use PyTorch with MPI to train a neural network of 1.2 million parameters on the MNIST dataset. We compare the following three settings: simple aggregation with 1 server, secure aggregation with 2 servers, robust secure aggregation with 2 servers (with Krum [5]). The number of workers is always 5.

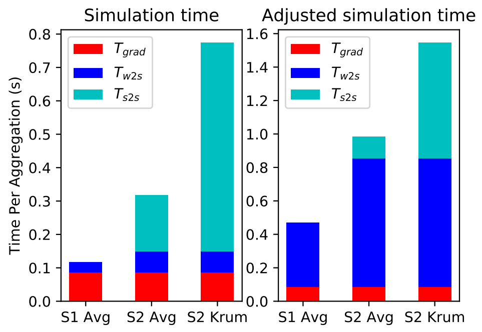


Figure 2: Left: Actual time; Right: Adjusted time.

In Figure 2, $T_{grad}$, $T_{w2s}$, $T_{s2s}$ stand for the time spent on gradient computation, worker to server, server to server communication. We adjust the bandwidth of the worker-to-server link to 100Mbps and server-to-server link to 1Gbps. From the right figure, we can see the overhea is small. As a comparison, a zero-knowledge-proof-based approach require 0.03 seconds to encode a submission of 100 integers [10].

## 6 Conclusion

In this paper, we propose a novel secure and Byzantine-robust aggregation framework. To our knowledge, this is the first work to address these two key properties jointly. Our algorithm is simple and fault tolerant and scales well with the number of workers. In addition, our framework holds for any existing distance-based robust rule. Besides, the communication overhead of our algorithm is roughly bounded by a factor of 2 and the computation overhead, as shown in Algorithm 3, is marginal and can even be computed prior to training.

## References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[2] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. Byzantine stochastic gradient descent. In *NeurIPS - Advances in Neural Information Processing Systems*, pages 4613–4623, 2018.

[3] Moran Baruch, Gilad Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. *arXiv preprint arXiv:1902.06156*, 2019.

[4] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

[5] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In *NeurIPS - Advances in Neural Information Processing Systems 30*, pages 119–129, 2017.

[6] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM journal on Computing*, 13(4):850–864, 1984.

[7] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. In *SysML - Proceedings of the 2nd SysML Conference, Palo Alto, CA, USA*, 2019.

[8] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[9] Melissa Chase, Ran Gilad-Bachrach, Kim Laine, Kristin E Lauter, and Peter Rindal. Private collaborative neural network learning. *IACR Cryptology ePrint Archive*, 2017:762, 2017.

[10] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 259–282, 2017.

[11] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 211–222, 2003.

[12] Avishek Ghosh, Justin Hong, Dong Yin, and Kannan Ramchandran. Robust federated learning in a heterogeneous environment. *arXiv preprint arXiv:1906.06629*, 2019.

[13] Lie He, Sai Praneeth Karimireddy, and Martin Jaggi. Byzantine-robust learning on heterogeneous datasets via resampling. *arXiv preprint arXiv:2006.09365*, 2020.

[14] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016.

[16] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: making SPDZ great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 158–189. Springer, 2018.

[17] Liping Li, Wei Xu, Tianyi Chen, Georgios B Giannakis, and Qing Ling. RSA: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1544–1551, 2019.

[18] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[19] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in byzantium. *arXiv preprint arXiv:1802.07927*, 2018.

[20] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. *arXiv preprint arXiv:1708.08689*, 2017.

[21] Luis Muñoz-González, Kenneth T. Co, and Emil C. Lupu. Byzantine-robust federated machine learning through adaptive model averaging. *arXiv preprint arXiv:1909.05125*, 2019.

[22] Krishna Pillutla, Sham M. Kakade, and Zaid Harchaoui. Robust Aggregation for Federated Learning. *arXiv preprint arXiv:1912.13445*, 2019.

[23] Daniel Ramage and Stefano Mazzocchi. Federated analytics: Collaborative data science without data collection. `https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html`, May 27 2020.

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

[25] Nigel P. Smart and Titouan Tanguy. TaaS: Commodity MPC via Triples-as-a-Service. In *CCSW'19 - Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, CCSW'19, page 105–116, 2019.

[26] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26–49, 2019.

[27] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Fall of Empires: Breaking Byzantine-tolerant SGD by Inner Product Manipulation. *arXiv preprint arXiv:1903.03936*, 2019.

[28] Andrew C Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*, pages 80–91. IEEE, 1982.

[29] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. *arXiv preprint arXiv:1803.01498*, 2018.