# A Decentralized Federated Learning Using Reputation*

Olive Chakraborty[a] and Aymen Boudguiga[b]
*CEA LIST, France*

Keywords: Decentralized Federated Learning, Reputation, Secret-Sharing, Fully Homomorphic Encryption, Zero-Knowledge Proofs, Secure Shuffling.

Abstract: Nowadays Federated learning (FL) is established as one of the best techniques for collaborative machine learning. It allows a set of clients to train a common model without disclosing their sensitive and private dataset to a coordination server. The latter is in charge of the model aggregation. However, FL faces some problems, regarding the security of updates, integrity of computation and the availability of a server.
In this paper, we combine some new ideas like clients' reputation with techniques like secure aggregation using Homomorphic Encryption and verifiable secret sharing using Multi-Party Computation techniques to design a decentralized FL system that addresses the issues of incentives, security and availability amongst others. One of the original contributions of this work is the new leader election protocol which uses a secure shuffling and is based on a proof of reputation. Indeed, we propose to select an aggregator among the clients participating to the FL training using their reputations. That is, we estimate the reputation of each client at every FL iteration and then we select the next round aggregator from the set of clients with the best reputations. As such, we remove misbehaving clients (e.g., byzantines) from the list of clients eligible for the role of aggregation server.

## 1 INTRODUCTION

Federated Learning (FL, (McMahan et al., 2017)) is a machine learning training technique where multiple *clients* collaborate with a *server* to train a common model. Each client computes a local model update based on its training data and shares it with the server. The server, in turn, aggregates these local updates to generate a *global* model update.

FL allows clients to contribute to training without revealing their private data, providing a degree of privacy. However, these updates can still disclose information about the client's data (Bhowmick et al., 2018). FL addresses this issue by employing *Secure Aggregation* (Bonawitz et al., 2017), which utilizes techniques such as Multi-party computation (MPC) (Evans et al., 2018), Homomorphic Encryption (Sébert et al., 2021), and Differential privacy (Dwork et al., 2014). FL generally assumes that participating clients and servers fulfill their roles reliably and honestly, though this is often not the case in real-world scenarios (Bhagoji et al., 2019; Fang et al., 2020). Clients deviating from expected behavior are termed *Byzantine*, and various resilient aggregation techniques have been proposed, including (Blanchard et al., 2017; Yin et al., 2018).

However, FL's privacy-preserving and Byzantine-resilient nature is insufficient on its own. Two commonly overlooked issues are the *availability* and *liveliness* of an aggregator[1]. A single aggregator introduces a single point of failure in an FL system, making it susceptible to Denial-of-Service (DoS) attacks by malicious clients in a centralized setting. Additionally, there may be scenarios where a trusted third-party server is unachievable. Decentralized federated learning models address these issues by using the resources of participating clients as aggregators while relying on mutual trust. The second issue is the lack of verifiability of computations in the presence of Byzantine clients and aggregators.

In this work, we propose a theoretical construction of a decentralized peer-to-peer federated learning algorithm that ensures secure Byzantine resilience, secure aggregation, and verifiability. We introduce the concept of *reputation* and utilize it to elect aggregators in various rounds of FL training. In the state-of-the-art, there exists few works (Rehman et al., 2020; Moudoud et al., 2021) which have started to use the concept of *incentives* but all of them utilize an existing blockchain architecture. Our work is based on the

[a] https://orcid.org/0000-0003-0396-908X
[b] https://orcid.org/0000-0001-6717-8848

*This work is submitted as a position paper as it is still in progress.

[1] In this article, we use server, leader, and aggregator interchangeably.



Chakraborty, O. and Boudguiga, A.
A Decentralized Federated Learning Using Reputation.
DOI: 10.5220/0012428300003648

In *Proceedings of the 10th International Conference on Information Systems Security and Privacy (ICISSP 2024)*, pages 799-806
ISBN: 978-989-758-683-5; ISSN: 2184-4356

use of cryptographic tools that can work efficiently on any system without the need of computationally expensive systems like blockchains.

The remainder of the paper is organized as follows: Section 2 describes the addressed problem and provides the required background for our construction, presenting the different cryptography building blocks we will use. Section 3 outlines our construction of a decentralized federated learning algorithm. Finally, Section 4 concludes the paper and provides perspectives.

# 2 BACKGROUND AND SOLUTION OVERVIEW

## 2.1 Peer-2-Peer Building Blocks

### 2.1.1 Leader Election in Peer-2-Peer Network

In this work, we consider a mesh network topology, where every client communicates with the others. In such a network, usually, one client acts as an aggregator. One method to select such a client is *leader election*. From a group of registered users participating in the election, the leader election chooses exactly one leader. In this work, we utilize the method of *secure shuffling* for leader election in a fair and unique manner. The method is similar to (Boneh et al., 2020), however we don't utilize the same framework.

### 2.1.2 Reputation

In certain scenarios, a Byzantine user may engage in malicious activities, attempting to manipulate the aggregation process in Federated Learning (FL) by providing inaccurate updates or improperly aggregating updates. Consequently, it becomes essential to prevent certain users in the network from participating in the training or leader election. To address this concern, various solutions, such as those based on Proof-of-Work (Jakobsson and Juels, 1999) and Proof-of-Stake (Shayan et al., 2020), have been proposed. However, when malicious users hold substantial stakes, they could potentially become leaders and act maliciously. Drawing inspiration from such scenarios, this work introduces the concept of *reputation*. The trustworthiness of a client, as perceived by other clients, is generally known as reputation. It serves as a reliable indicator of a client's future behavior within a network of interacting nodes. A positive reputation tends to correlate with expected good behavior, while a negative reputation correlates with anticipated misconduct. Reputation is determined based on a client's actions during a specific interaction, which can be either positive or negative. We utilize this reputation metric as an input to identify the leader among the parties with the highest reputation.

## 2.2 Cryptographic Building Blocks

### 2.2.1 Homomorphic Encryption

Homomorphic encryption (HE) allows the evaluation of arbitrary functions on encrypted data. It allows performing operations on a ciphertext, whose decryption corresponds to algebraic operations on the plaintext. An HE is characterized by mainly four operations: KeyGen, Enc, Dec and Eval. KeyGen generates a secret and public key pair for the asymmetric variants and a symmetric key for the symmetric variants of HE and an evaluation key. The Enc and Dec are similar to their classical tasks in conventional encryption schemes. Eval is a HE-specific operations, which takes ciphertexts as input and outputs a ciphertext corresponding to a functioned plaintext. The most important point in homomorphic encryption is that the format of the ciphertexts after an evaluation process must preserve the form a readable ciphertext in order for it to be decrypted correctly. In the state-of-the-art, HE can be broadly classified into Partially HE (PHE), Somewhat SHE (SHE) and Fully HE (FHE). PHE (Rivest et al., 1978; ElGamal, 1985; Paillier, 1999) supports the Eval function for only either addition or multiplication, while SHE (Boneh et al., 2005; Sander et al., 1999; Ishai and Paskin, 2007) supports for only limited number of operations or limited circuit. FHE (Gentry, 2009; Brakerski and Vaikuntanathan, 2011; Fan and Vercauteren, 2012; Brakerski et al., 2014; Chillotti et al., 2016a; Cheon et al., 2017) supports evaluation of any arbitrary function for unlimited number of times over ciphertexts. In this work we use the TFHE (Chillotti et al., 2016a) cryptosystem on the aggregator side.

### 2.2.2 Shamir Secret Sharing

Adi Shamir in (Shamir, 1979) proposed an idea of sharing a secret $s$ amongst $n$ parties such that the complete secret can be reconstructed from any combination of $t \leq n$ shares and any $t-1$ or less shares reveals no information about the secret $s$. This $t$ is called the *threshold* of the secret sharing scheme. The scheme is defined over a finite field $\mathbb{F}$ and has two algorithms:

- $\{(i, s_i)\}_{i \in P} \xleftarrow{\$} \mathsf{Share}(s, P, t)$. For a secret $s \in \mathbb{F}$, a set of $n$ unique field elements $P \in \mathbb{F}^n$ and a threshold $t$, this algorithm chooses a random polynomial $p \in \mathbb{F}[X]$ such that $p(0) = s$ and generates shares

as $(i, p(i))$ $i \in P$.

- $s \leftarrow \mathsf{Rec}((i,s_i)_{i\in Q})$. Given shares of a secret from a subset $Q \subset P, |Q| \geq t$, this algorithm reconstruct the secret $s$.

To tolerate cases of user dropouts or message errors, a more robust reconstruction technique is used in Secret Sharing. One can leverage Reed-Solomon decoding (Blahut, 1983) for robust construction of Shamir's Secret Shares (Roy Chowdhury et al., 2022).

- $s \leftarrow \mathsf{RobustRec}((i,s_i)_{i\in Q})$. The previous scheme results in a $[n,t,n-t+1]$ Reed-Solomon code that can tolerate up to $q$ errors and $e$ message dropouts such that $2q+e < n-t+1$. Given any subset of $n-e$ shares, with upto $q$ errors, any standard Reed Solomon decoding algorithm can robustly reconstruct $s$.

For achieving a Verifiable Secret Sharing (VSS) from Shamir's secret sharing scheme, one can use of the Feldman's technique (Feldman, 1987). For a share of a secret, a party must be able to check its validity. If the share is valid, there exists a unique secret which will be the output of the reconstruction algorithm when run on any $t$ distinct valid shares. More formally, we have:

- $1/0 \leftarrow \mathsf{Verify}((i,v),\Psi)$. For an input of a share and a check string $\Psi_s$ such that:

$$\forall\, V \subset \mathbb{F}\times\mathbb{F}.\ \text{where } |V| = t, \exists s \in \mathbb{F}\ \text{s.t}$$
$$(\forall (i,v) \in V, \mathsf{Verify}((i,v),\Psi) = 1)$$
$$\implies \mathsf{Rec}(V) = s.$$

The check string are the commitments to the coefficients given by:

$$\psi_i = g^{c_i}, i \in \{0,\ldots,t-1\},$$

where $g$ denotes a generator of $\mathbb{F}$. All arithmetic is taken modulo $q$ such that:

$$(p|q-1)$$

where p is the prime of $\mathbb{F}$. For verifying a share $(j,s_j)$, a party needs to check whether $g^{s_j} = \prod_{i=0}^{t-1} \psi_i^{j^i}$. The privacy of the secret $s = c_0$ is implied by the the intractability of computing discrete logarithms.

#### 2.2.3 Dynamic Threshold Homomorphic Encryption

A threshold fully homomorphic cryptosystem brings together three main algorithms: a distributed key generation protocol, an FHE scheme, and a threshold decryption protocol. A threshold decryption protocol makes use of a secret sharing scheme, such as Shamir Secret Sharing (Shamir, 1979) for a distributed decryption such that $t$ out of $n$ parties ($t \leq n \in \mathbb{Z}_+$) can decrypt the secret, however, no combination of $t-1$ parties can get any information about the secret. Threshold HE can be either *static* or *dynamic*. In this work, we consider the case of a dynamic threshold system, where a new user can join the training process without the need to regenerate the keys for all existing parties along with the new party.

#### 2.2.4 Zero-Knowledge Proofs, Secret-Shared Non Interactive Proofs and Commitments

Non-interactive zero-knowledge (NIZK) proofs are cryptographic primitives that allow a prover $\mathcal{P}$, to convince a verifier $\mathcal{V}$ that a statement is true using a single message (the proof) without the verifier learning anything more. NIZKs are particularly useful as a single proof can be re-utilized to convince multiple verifiers. A secret-shared non-interactive proof (SNIP) (Corrigan-Gibbs and Boneh, 2017) is an information-theoretic zero-knowledge proof (ZKP) system for distributed data. When a secret data is distributed amongst multiple parties, receivers act as verifiers, verifying the validity of their shares. SNIP, relies specifically on a additive secret sharing scheme over a finite field. A commitment is a cryptographic primitive that allows a user to publish a value, *commitment*, which binds the user to its message without revealing it. Any commitment scheme satisfies the properties of correctness, binding and hiding. Commitments have been used quite a lot as a part of verifiable secret sharing, secure MPCs and ZKPs (Damgård and Nielsen, 2008).

#### 2.2.5 Secure Shuffling

The leader election process employs a consensus protocol to securely and provably select a leader in a random manner, gaining the consensus of participating clients. The definition of the single secret leader election SSLE protocol was formalized by (Boneh et al., 2020), where the leader's identity remains unknown to clients except for the leader themselves. Boneh et al., proposed an intriguing idea in this article by introducing the use of shuffling, a well-known technique for randomly selecting members from a list, commonly applied in tasks like choosing a leader or a group of validators in blockchains.

However, our work necessitates a lightweight and non-secret leader election process. In this endeavor, we adopt a secure and lightweight 'swap-or-not' shuffling technique (Hoang et al., 2012), notably forming the foundation of the oblivious shuffling-based secure single leader election protocol utilized in Ethereum

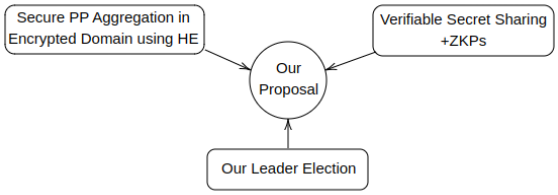


Figure 1: An illustration of the scheme : Secure Privacy Preserving aggregation provides privacy, Verifiable Secret Sharing provides verifiability and data3 integrity, while our Leader election protocol ensures availability.

2.0 (Asn et al., 2022; Sanso, 2022). This shuffling technique is a modified version of (Boneh et al., 2020)'s proposal. It employs deterministic functions to generate pseudo-random outputs from inputs, ensuring verifiability by any client when required. The function takes two inputs: a random value `seed`, and the `size` of the list to be shuffled. In the subsequent sections, we denote this shuffling process using the function SECURESHUFFLE(`seed`,`size`).

#### 2.2.6 Public-Key Infrastructure and Public Bulletin

In this work we require the presence of a public-key infrastructure PKI that allows clients to register their identities, and sign messages which other clients can verify, but cannot impersonate them. This prevents the aggregator from simulating an arbitrary number of clients. One can assume that a public bulletin $\mathcal{B}$ is available to every client. Every client can read/write access to $\mathcal{B}$ which will be used as a medium for broadcasting information (Bonawitz et al., 2017). The bulletin contains information about each round of the training along with the public keys of all the participating clients. The details on the content of $\mathcal{B}$ are given in the subsequent sections.

## 3 OUR PROPOSAL

In this section we present a peer-2-peer decentralized federated learning model, which combines secure aggregation, verifiability and a new leader election process (see Figure 1). Figure 2 shows an high-level overview of the proposal. We assume that participating clients have enough computational capabilities. We also assume that the proposed protocol works under the Common Reference String (CRS) model.

### 3.1 Threat Model

We consider a threat model where we can have:

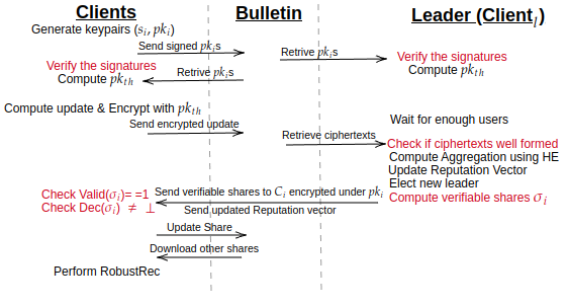


Figure 2: High-level view of our protocol. Red parts are required to guarantee of robustness and verifiabilty.

- **Malicious Clients:** We consider a set of $m$ malicious client, $C_M$. These malicious clients can deviate from the protocol, arbitrarily by either 1) sending malformed inputs, 2) sending inputs with the intention of leading the aggregation further away from the actual convergence, 3) by, violating the privacy of an honest client, by colluding with other clients or the aggregator.
- **Malicious Aggregator:** We consider that the aggregator can deviate from the protocol arbitrarily by either 1) providing incorrect aggregation results acting independently or via collusion, 2) aiming at recovering individual updates of an honest client.

### 3.2 Setup

Let $\mathcal{T} = \{C_1, \ldots, C_n\}$ be a set of $n$ clients. In the setup phase, all parties are initialized with system parameters, including the security parameter $\lambda$, the number of clients $n$, the threshold parameter $t$ (representing the maximum number of potentially malicious clients), and a field $\mathbb{F}$ where $|\mathbb{F}| \geq 2^{\lambda}$. A standard PKI is employed for clients to register on $\mathcal{B}$ with specific user IDs. For simplicity, we assume that each client $C_i$ is assigned a user ID $i \in \mathbb{Z}_n$. In each round $j$ of the training process, each client $C_i$ maintains a local update $\mathbf{w}_{i,j} \in \mathbb{F}^d$. All clients have read and write access to $\mathcal{B}$ through authenticated channels.

Prior to the first round, under the CRS model, each client $C_i$ has access to the public parameter $A$ and independently generates the key pair $(\mathbf{s}_i, \mathbf{pk}_i) \xleftarrow{\$} \mathsf{FHE.KeyGen}()$, where $\mathbf{pk}_i = A\mathbf{s}_i + \mathbf{e}_i$ , for some $e_i \xleftarrow{\$} \mathcal{D}$, where $\mathcal{D}$ is some error distribution (Fan and Vercauteren, 2012). Clients then update their public keys and user IDs on the bulletin $\mathcal{B}$. Additionally, each client uploads a random value $r_i \in \mathbb{Z}$. Once all public keys have been broadcasted on $\mathcal{B}$, each client can construct the the global public key as $\mathbf{pk_{th}} = \sum_i \mathbf{pk_i}$. This public key is used by each client to send their en-

crypted weights. This one-time communication overhead does not need to be repeated when a new client joins the training. Upon a new client joining, she independently creates her public key and uploads it to $\mathcal{B}$.

Additionally, on the public bulletin $\mathcal{B}$, a reputation vector $\mathcal{R} \in \mathbb{Z}^n$ is maintained. This vector tracks the *reputation* values of each participating client to the FL training. Initially set to 1 during setup, this vector is appended with the reputation of each new user upon registration, i.e., 1. Finally, each client also maintains a list, $\mathcal{C}^*$, of identified malicious clients.

**Leader Selection.** Along with computing the collective public key $\mathbf{pk_{th}}$, a leader is also selected. Using the random values $r_i \in \mathbb{Z}$ uploaded during registration, every client computes a common random seed value $r_{beg} = \sum_i r_i$. Each client then calls the function SECURESHUFFLE($r_{beg}, 1, n$) to receive the index of the leader for the first round.

**Note 1.** *For ensuring the availability of a leader/aggregator, after the selection of a leader, the clients wait for a certain time threshold, after which the leader election process is repeated.*

## 3.3 Send Secure Updates

The aggregation of $d$-dimension gradients from $n$ clients requires a communication complexity of $O(d)$ in terms of communication traffics, which generally limits the system scalability. Gradient sparsification (Ergun et al., 2021) is a promising technique for distributed stochastic gradient, which can significantly reduce the communication traffic while preserving the model convergence. In gradient sparsification, a compressor $\mathsf{Comp}_k$ is applied on each update to locally select $k$, $k \leq d$, gradients for aggregation and $\mathsf{Comp}_k$ $\in$ {Top-k, Rand-k} (Ergun et al., 2021). $\mathsf{Comp}_k$ zeros out $(d-k)$ elements of the update and keeps $k$ elements unchanged. For every round $j$, each participating client encrypts their sparsified local updates $\overline{\mathbf{w}_{i,j}} = \mathsf{Comp}_k(\mathbf{w}_{i,j})$ using $\mathbf{pk_{th}}$. Each client $C_i$ uploads the encrypted weight, $\mathsf{Enc}(\overline{\mathbf{w}_{i,j}}, \mathbf{pk_{th}})$ on $\mathcal{B}$. Since the public key has been constructed in a threshold manner, the leader is unable to learn anything about the local updates sent from the other clients. The client also uploads a commitment $\chi_{i,j}$ to the weight $\overline{\mathbf{w}_{i,j}}$. This can be later utilized if a malicious client tries to deny sending a malformed or 'malicious' input. In order to prove that the encrypted value is a well-formed input, the client also sends a SNIP $\Psi_{i,j}$. For the rest of the document, we shall denote any encrypted entity as $[\cdot]$.

## 3.4 Perform Secure Aggregation and Update Reputation

Before initiating the aggregation, the leader awaits a time threshold to receive the updates $[\overline{\mathbf{w}i,j}]\mathbf{pk_{th}}$. Utilizing the proofs $\pi_{i,j}$, the leader conducts a validity check on the encrypted inputs. Updates failing the validity check are rejected, and the corresponding clients are marked as malicious, and are added to $\mathcal{C}^*$. In this study, we opt for SABLE, (Choffrut et al., 2023), a novel Byzantine-resilient secure aggregation method. SABLE efficiently implements the coordinate-wise Trimmed mean over the encrypted domain using Homomorphic Encryption and can tolerate up to $t$-out-of-$n$ Byzantines. While our selection of this secure aggregation method is specific to this paper, any other privacy-preserving aggregation method capable of handling Byzantine clients and aggregators in the encrypted domain can be employed. It is crucial to note that the choice of the aggregation method impacts both the accuracy and efficiency of the overall scheme. Additionally, the chosen leader may still be in the process of completing their own training. In such cases, clients await a time counter $\mathsf{time_{limit}}$ for the leader to return the aggregated weight. If there's no response, the leader is added to the malicious clients' list $\mathcal{C}^*$, prompting clients to restart the leader selection process by uploading new random values $r_i$s to $\mathcal{B}$.

### 3.4.1 Compute Reputation

As already stated earlier, each participating user in the scheme is associated with a reputation, that is updated at the end of each round. In each round the leader computes the reputation change of each client based on their round performance. The reputation update is based on the distance of the individual weights $[\overline{\mathbf{w}_{i,j}}]$ from the aggregated weight $[\mathbf{w}_{agg}]$. To compute this, we make use of the Homomorphic encryption. In particular we use functional bootstrapping (Clet et al., 2022) over TFHE (Chillotti et al., 2016b). The first step is to compute the distance between the two weights homomorphically : $[d_i] = ||[\mathbf{w}_{agg}] - [\overline{\mathbf{w}_{i,j}}]||_2$ and this is mapped onto the top half of the $\mathbb{T} = \mathcal{R}/\mathbb{Z}$. $\mathbb{T}$ is the additive group of real numbers modulo 1 ($\mathcal{R}$ mod [1]) and it is a $\mathbb{Z}$ module. Given a base $b$ with which we encode the reputation changes 1 and 2, such that $\frac{1}{b}, \frac{2}{b} \in \mathbb{T}$ and proceed to create the test vector as:

$$testv = \sum_{i=0}^{\frac{N}{4}-1} \frac{2}{b} X^l + \sum_{i=\frac{N}{4}}^{\frac{N}{2}-1} \frac{1}{b} X^l - \sum_{i=\frac{N}{2}}^{\frac{3N}{4}-1} \frac{1}{b} X^l - \sum_{i=\frac{3N}{4}}^{N-1} \frac{2}{b} X^l$$

For a random distance of $d_i \in [0, \frac{1}{2}[$, the outputs of the bootstrapping algorithm (Algorithm 9, in (Chillotti

et al., 2016b)) are:

$$\texttt{Rep}=\begin{cases}[\frac{2}{b}] & \text{if } [d_i]\in[0,1/8[\\ [\frac{1}{b}] & \text{if } [d_i]\in[1/8,1/4[\\ [\frac{-1}{b}] & \text{if } [d_i]\in[1/4,3/8[\\ [\frac{-2}{b}] & \text{if } [d_i]\in[3/8,1/2[\end{cases} \quad (1)$$

The leader performs this bootstrapping procedure and decrypts the reputation update (scaled by a factor of $b$) for each of the participating clients in the training. The clients that perform well, i.e., whose weights are closer to the aggregated weights, are incentivized with respect to their distance. While, the users whose weights are further away, and hence more likely to have performed maliciously are de-incentivized. The leader then decrypts the reputation changes and updates the global reputation vector $\mathcal{R}$.

We introduce a concept of *maximum reputation*, denoted as `maxrep`, beyond which a client's reputation is reset to 1. This ensures *fairness* by providing every client an equal opportunity to become a leader. Additionally, the leader must also reset its reputation to 1, preventing the current leader from immediate re-election and promoting fairness in the leader selection process. Both the reputation resets are performed by the current leader.

Due to the lack of a trusted setup, to attest to the honesty of the computation, the leader furnishes a zero-knowledge proof using zk-SNARKS (e.g., Aurora (Ben-Sasson et al., 2019)). While zk-SNARKS are generally faster for both prover and verifier (Viand et al., 2023), in a decentralized setting like this, achieving a trusted environment is challenging. Therefore, we choose to prioritize larger times for creating proofs. If however, the generations of the proofs are somehow done in a Trusted Execution Environments (Damgård et al., 2008), zk-SNARKS can be used guaranteeing faster executing times.

Any participating client engaged in the aggregation process can verify the authenticity of updates, ensuring they were sent by the correct leader and that no other malicious client attempted to upload faulty updates. Furthermore, after each round, any client can verify whether the previous leader accurately updated their reputation and refrained from maliciously attempting to become the leader again, thereby influencing the aggregation process. In cases where a leader maliciously attempts to decrease another client's reputation, the affected client can dispute this action by making its weight public. Other clients can then use this information to compute a bootstrapped reputation update. If successful, all clients collectively decide to abort the protocol, concluding that the leader has acted maliciously. The misbehaving leader is subsequently added to the malicious set $\mathcal{C}^*$ and is prohibited from further participation in the protocol.

In situations where a leader colludes with another client to increase its weight and to become the leader in the subsequent round, the colluding client is still obligated to perform an honest aggregation. Failure to adhere to this results in the client being banned from further participation.

#### 3.4.2 Leader Election for Next Round

Before, sending the shares of the global update, the leader must also elect the leader for the next round. Firstly, the reputation vector needs to be sorted to shortlist all the indices with the maximum reputation. Recall that every client also sends a random value $r_{i,j} \in \mathbb{Z}$. The leader then computes $\sum_i r_{i,j}$ and determines the leader for the next round using $\texttt{SecureShuffle}(\sum_i r_{i,j}, k)$, where $k$ is the size of the shortlist with the maximum reputation. The leader also provides a SNIP $\pi_{\ell,j}$ to attest to the honesty of the shuffling for the $j^{th}$ round.

### 3.5 Send Updated Weight Using VSS

After the aggregation has been computed, the aggregator now sends $[\mathbf{w}_{agg}]$ for collaborative decryption. The leader generates the secret shares of aggregated weight $\{(1,\mathbf{u}_1),\ldots,(n,\mathbf{u}_n),\pi\} \xleftarrow{\$} \mathsf{Share}(\mathbf{w}_{agg},\mathcal{T},t)$. The $\pi$ is a SNIP that proves the consistency of the computation done by the leader. The leader $C_\ell$ also generates the proof $\Psi_i$ of validity of the share $\mathbf{u}_i$, i.e. $\mathbf{u}_i$ is indeed a share of $[\mathbf{w}_{agg}]$. This will allow the receiving client to verify the validity of the their corresponding share. Finally, it encrypts the updates' shares and the proof's shares $(i,\mathbf{u}_i)||(\Psi_i)$ for each individual client $C_i$ using the public key $\mathbf{pk}_i$, signs them with his signing key and publishes it on the public bulletin $\mathcal{B}$.

### 3.6 Distributed Decryption, Reconstruction of Update & Verification

Once, the leader has uploaded the shares of the new aggregate weights and the proofs on $\mathcal{B}$, each client downloads and computes the shares $\sigma_i = \mathsf{Dec}(([i,\mathbf{u}_i],\mathbf{sk}_i)$. He also downloads the check string $\Psi_i$ and verifies the validity of the share using SNIP. The clients must also validate the signatures of the sender and verifies if it has been sent by the correct leader. If not, then the sender whose signatures

Table 1: Cost summary of the protocol.

| | Client | Leader |
|---|---|---|
| **computation** | $O(n)$ | $O(mn^2)$ |
| **communication** | $O(n+m)$ | $O(n^2)$ |
| **storage** | $O(n+m)$ | $O(n+m)$ |

matches that of the received share, must be added to $C^*$. Also, since we have the presence of $t$ malicious clients, this allows any cohort of $n-t$ clients to reconstruct the weight and instantiate a SNIP protocol. If any of the shares are not valid, the client flags the leader and also adds the leader to the malicious list $C^*$. If the share is valid, the client broadcasts its share $(i, \sigma_i)$ to $\mathcal{B}$. This is necessary for all the other honest clients to recover the aggregated weight for the new round. After downloading all the shares of the other clients, $\sigma_{j\neq i}$, $C_i$ finally performs a robust reconstruction on its side to recover the aggregated weight for the round as $\mathbf{w}_{agg} = \mathsf{RobustRec}(i, \sigma_i)$.

**Note 2.** *For the new round, it might happen that the elected leader is still in the process of completing its own training and therefore unavailable to complete the secure aggregation. In such a case, the leader election process might be appended to select a deputy leader which takes up the job of the leader. This would of course result in de-incentivizing the originally elected leader.*

# 4 CONCLUSION AND PERSPECTIVES

Modern practical FL models have introduced the need of ensuring the privacy, integrity and availability of model aggregators, with no single point of failure. In this work, we provide a new theoretical construction which addresses all of them. We use the secure byzantine resilient model aggregation technique of SABLE, which protects the system from byzantine attacks as well providing privacy to private updates. We use VSS as well as SNIPs which allows verifiable data integrity in secure aggregation. Also to the best of our knowledge, this article provides the first leader election protocol which is based on reputation unlike the state-of-the-art. We consider that the use of reputation opens up a new avenue for future research. We also provide an initial cost analysis of the scheme in Table 1.

We remind that this is a theoretical work and it is still in progress. As for future work, we aim to provide an empirical evaluation of the FL scheme as well as an implementation of this construction.

# REFERENCES


Asn, B., Killari, B., and Genya-Z (2022). Whisk: A practical shuffle-based ssle protocol for ethereum.

Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., and Ward, N. P. (2019). Aurora: Transparent succinct arguments for r1cs. In *EUROCRYPT 2019*, pages 103–128. Springer.

Bhagoji, A. N., Chakraborty, S., Mittal, P., and Calo, S. (2019). Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*, pages 634–643. PMLR.

Bhowmick, A., Duchi, J., Freudiger, J., Kapoor, G., and Rogers, R. (2018). Protection against reconstruction and its applications in private federated learning. *arXiv preprint arXiv:1812.00984*.

Blahut, R. E. (1983). Theory and practice of error control codes. *(No Title)*.

Blanchard, P., El Mhamdi, E. M., Guerraoui, R., and Stainer, J. (2017). Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in neural information processing systems*, 30.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. (2017). Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191.

Boneh, D., Eskandarian, S., Hanzlik, L., and Greco, N. (2020). Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 12–24.

Boneh, D., Goh, E.-J., and Nissim, K. (2005). Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*, pages 325–341. Springer.

Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (2014). (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36.

Brakerski, Z. and Vaikuntanathan, V. (2011). Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer.

Cheon, J. H., Kim, A., Kim, M., and Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer.

Chillotti, I., Gama, N., Georgieva, M., and Izabachene, M. (2016a). Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*, pages 3–33. Springer.

Chillotti, I., Gama, N., Georgieva, M., and Izabachène, M. (2016b). Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *ASIACRYPT 2016*, pages 3–33. Springer Berlin Heidelberg.

Choffrut, A., Guerraoui, R., Pinot, R., Sirdey, R., Stephan, J., and Zuber, M. (2023). Practical homomorphic aggregation for byzantine ml. *arXiv preprint arXiv:2309.05395*.

Clet, P.-E., Zuber, M., Boudguiga, A., Sirdey, R., and Gouy-Pailler, C. (2022). Putting up the swiss army knife of homomorphic calculations by means of tfhe functional bootstrapping. *Cryptology ePrint Archive*.

Corrigan-Gibbs, H. and Boneh, D. (2017). Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*, pages 259–282.

Damgård, I. and Nielsen, J. (2008). Commitment schemes and zero-knowledge protocols (2007). In *LNCS'08: Lecture Notes in Computer Science*.

Damgård, I., Nielsen, J. B., and Wichs, D. (2008). Isolated proofs of knowledge and isolated zero knowledge. In *Advances in Cryptology–EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings 27*, pages 509–526. Springer.

Dwork, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407.

ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472.

Ergun, I., Sami, H. U., and Guler, B. (2021). Sparsified secure aggregation for privacy-preserving federated learning. *arXiv preprint arXiv:2112.12872*.

Evans, D., Kolesnikov, V., Rosulek, M., et al. (2018). A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246.

Fan, J. and Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption.

Fang, M., Cao, X., Jia, J., and Gong, N. (2020). Local model poisoning attacks to {Byzantine-Robust} federated learning. In *29th USENIX security symposium (USENIX Security 20)*, pages 1605–1622.

Feldman, P. (1987). A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178.

Hoang, V. T., Morris, B., and Rogaway, P. (2012). An enciphering scheme based on a card shuffle. In *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 1–13. Springer.

Ishai, Y. and Paskin, A. (2007). Evaluating branching programs on encrypted data. In *Theory of Cryptography Conference*, pages 575–594. Springer.

Jakobsson, M. and Juels, A. (1999). Proofs of work and bread pudding protocols (extended abstract). secure information networks (s. 258-272).

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.

Moudoud, H., Cherkaoui, S., and Khoukhi, L. (2021). Towards a secure and reliable federated learning using blockchain. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 01–06. IEEE.

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

Rehman, M. H., Salah, K., Damiani, E., and Svetinovic, D. (2020). Towards blockchain-based reputation-aware federated learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 183–188. IEEE.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Roy Chowdhury, A., Guo, C., Jha, S., and van der Maaten, L. (2022). Eiffel: Ensuring integrity for federated learning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2535–2549.

Sander, T., Young, A., and Yung, M. (1999). Non-interactive cryptocomputing for nc/sup 1. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 554–566. IEEE.

Sanso, A. (2022). Towards practical post quantum single secret leader election (ssle).

Sébert, A. G., Pinot, R., Zuber, M., Gouy-Pailler, C., and Sirdey, R. (2021). SPEED: secure, PrivatE, and efficient deep learning. *Machine Learning*, 110(4):675–694.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

Shayan, M., Fung, C., Yoon, C. J., and Beschastnikh, I. (2020). Biscotti: A blockchain system for private and secure federated learning. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1513–1525.

Viand, A., Knabenhans, C., and Hithnawi, A. (2023). Verifiable fully homomorphic encryption. *arXiv preprint arXiv:2301.07041*.

Yin, D., Chen, Y., Kannan, R., and Bartlett, P. (2018). Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*, pages 5650–5659. PMLR.