Blockchain-based Secure Client Selection in Federated Learning

Truc Nguyen* – Phuc Thai†, Tre' R. Jeter*, Thang N. Dinh† and My T. Thai*
*Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL, 32611
†Department of Computer Science, Virginia Commonwealth University, Richmond, VA, 23284
Email: truc.nguyen@ufl.edu, thaipd@vcu.edu, t.jeter@ufl.edu, tndinh@vcu.edu, mythai@cise.ufl.edu

2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC) | 978-1-6654-9538-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICBC54727.2022.9805521

***Abstract*—Despite the great potential of Federated Learning (FL) in large-scale distributed learning, the current system is still subject to several privacy issues due to the fact that local models trained by clients are exposed to the central server. Consequently, secure aggregation protocols for FL have been developed to conceal the local models from the server. However, we show that, by manipulating the client selection process, the server can circumvent the secure aggregation to learn the local models of a victim client, indicating that secure aggregation alone is inadequate for privacy protection. To tackle this issue, we leverage blockchain technology to propose a verifiable client selection protocol. Owing to the immutability and transparency of blockchain, our proposed protocol enforces a random selection of clients, making the server unable to control the selection process at its discretion. We present security proofs showing that our protocol is secure against this attack. Additionally, we conduct several experiments on an Ethereum-like blockchain to demonstrate the feasibility and practicality of our solution.**

I. Introduction

In recent years, Federated Learning (FL) has emerged as an auspicious large-scale distributed learning framework that simultaneously offers both high performance in training models and privacy protection for clients. FL, by design, allows millions of clients to collaboratively train a global model without the need of disclosing their private training data. In each training round, a central server distributes the current global model to a random subset of clients who will train locally and upload model updates to the server. Then, the server averages the updates into a new global model. FL has inspired many applications in various domains, including training mobile apps [14], [28], self-driving cars [16], [24], digital health [6], [25], and smart manufacturing [13], [18].

Although training data never leaves clients' devices, data privacy can still be leaked by observing the local model updates and conducting some attacks such as membership inference [26], [27]. Thus, FL is not particularly secure against an honest-but-curious server. To address this issue, recent research has focused on developing a privacy-preserving FL framework by devising secure aggregation on the local models [2], [5], [29]. Specifically, it enables the server to privately combine the local models in order to update the global model without learning any information about each individual local model. As a result, the local model updates are concealed from the server, thereby preventing the server from exploiting the updates of any client to infer their private training data.

However, in this paper, we exploit a gap in the existing secure aggregation and show that they are inadequate to protect the data privacy. Particularly, we demonstrate that a *semi-malicious* server can circumvent a secure aggregation to learn the local model updates of a victim client via our proposed *biased selection attack*. Intuitively, our attack leverages the fact that the central server in FL has a freedom to select any pool of clients to participate in each training round. Hence, it can manipulate the client selection process to target the victim and extract their update from the output of the secure aggregation protocol. We present two different strategies to conduct the biased selection attack, and show experimentally that the server can successfully infer some information about the victim's private training data without making any additional security assumptions about the capabilities of the server.

To counter this attack, we focus on strictly enforcing a random selection of clients on the central server, thereby preventing it from manipulating the selection process at its discretion. To this end, we propose using blockchain as a public trust entity and devise a verifiable random selection protocol for the server to randomly select a pool of clients in each training round. Specifically, we utilize the blockchain as a source of randomness that is used to determine the pool of clients that will participate in a training round. Via the immutability of blockchain, the clients can verify the correctness of the random selection protocol, i.e., ensuring that they are indeed randomly selected. To demonstrate the feasibility of our solution, we concretely prove that our protocol is secure against the biased selection attack. We also benchmark the performance of the proposed protocol with an Ehtereum-like blockchain and show that it imposes minimal overhead on FL.

Contributions. Our contributions are summarized as follows:

- We propose the biased selection attack where the server learns the local model updates of a victim in spite of secure aggregation. We describe two strategies to perform this attack without making extra security assumptions on the server. Then, we conduct some experiments to demonstrate its viability with respect to inferring some information about the victim's training data.

The first two authors contribute equally to this paper.

978-1-6654-9538-7/22/$31.00 ©2022 IEEE

- As a countermeasure, we devise a verifiable random selection protocol for the server to randomly select clients in each training round. Our protocol leverages blockchain as a source of randomness so that the clients can verify whether the server correctly follows the selection protocol. Therefore, it enforces a random selection of clients, making the biased selection attack infeasible.
- We present concrete security proofs to show that the proposed protocol is secure against the attack. We also analyze the communication and computation cost of the protocol, together with some benchmarks to show that its overhead on FL is minimal.

Organization. The rest of the manuscript is structured in the following manner. Section II establishes the preliminaries for our paper. We present the biased selection attack in Section III. Section IV describes our proposed client selection protocol. We then provide security and performance analysis in Section V. Experiments to evaluate our solution are given in Section VI. We discuss some related work in Section VII and finally provide concluding remarks in Section VIII.

II. Preliminaries

A. Federated Learning and Secure Aggregation

Depending on how training data is distributed among the participants, there are two main versions of federated learning: horizontal and vertical. In this paper, we focus on a horizontal setting in which different data owners hold the same set of features but different sets of samples.

Typically, an FL process follows the FedAvg framework [22] which comprises multiple rounds. In this setting, a server and a set $\mathcal{U}$ of $n = |\mathcal{U}|$ clients participate in a collaborative learning process. Each client $u \in \mathcal{U}$ holds a training dataset D_u and agrees on a single deep learning task and model architecture to train a global model. A central server $\mathcal{S}$ keeps the parameters G^t of the global model at round t. Let x_u^t be a vector representing the parameters of the local model of client u at round t. Each training round includes the following phases:

1) *Client selection*: $\mathcal{S}$ samples a subset of m clients $\mathcal{U}' \subseteq \mathcal{U}$ and sends them the current global model G^t.
2) *Client computation*: each selected client $u \in \mathcal{U}'$ updates G^t to a new local model x_u^t by training on their private data D_u, and uploads x_u^t to the central server $\mathcal{S}$.
3) *Aggregation*: the central server $\mathcal{S}$ averages the received local models to generate a new global model as follows:

$$G^{t+1} = \frac{1}{m} \sum_{u \in \mathcal{U}'} x_u^t \tag{1}$$

The training continues until the global model converges.

To counter several attacks conducted based on the local model updates of clients, such as inference attacks by the server [1], [11], the Aggregation phase can be replaced by a *secure aggregation* protocol such that each x_u^t is not exposed to the server [2], [5], [29]. By leveraging cryptographic secure multiparty computation (SMC), the secure aggregation protocols can guarantee that the server cannot learn any information about each local model update, but still be able to construct the sum of all updates. Specifically, with secure aggregation, the equation (1) is replaced by:

$$G^{t+1} = \frac{1}{m} \prod_{\{x_u^t | u \in \mathcal{U}'\}} \left[\sum_{u \in \mathcal{U}'} x_u^t \right] \tag{2}$$

where $\prod_X [f(X, \cdot)]$ denotes an abstract secure computation protocol on some function $f(X, \cdot)$ and X is a private input. The protocol $\prod_X [f(X, \cdot)]$ is: (1) *correct* if it outputs the same value as $f(X, \cdot)$, and is: (2) *secure* if it does not reveal X during the execution of the protocol.

B. Blockchain

Blockchain, introduced in [23], is a type of distributed ledger, jointly maintained by a set of nodes in a network, called miners. Blockchain can provide guarantees on the correctness (i.e. tamper-resistance) and security of the ledger without the need of trust on a central trusted party.

A consensus protocol for maintaining blockchain is called secure if it satisfies the following two security properties: 1) *persistence*: all honest miners have the same view of the ledger; and 2) *liveness*: the valid transactions will eventually be added to the ledger.

In this work, we consider a proof-of-work (PoW) blockchain, in which miners compete to solve a PoW puzzle. The miner who solves the puzzle can append a new block into a blockchain data structure. The PoW blockchain is shown to be secure under the assumption that the honest miners hold the majority of mining power [12]. The security of the protocol is parameterized by the length of the hash function $\kappa \in \mathbb{N}$ [12], called *security parameter*.

The blockchain is used in our client selection protocol to ensure 1) all clients in FL have the same views on the selected clients, and 2) a provably random selection of the client.

C. Verifiable random function

To implement the provable random client selection, we use a cryptographic tool called *verifiable random functions* (VRF) [10]. VRF is a public-key pseudorandom function that provides proofs showing that its outputs were calculated correctly and randomly, i.e., hard to predict. Consider a user with secret and public keys sk and pk. The user can use VRF to generate a function output σ and a proof π for any input value x by running a function $\mathsf{VRFprove}_{\mathsf{sk}}(x)$. Everyone else, using the proof π and the public key pk, can check that the output σ was calculated correctly by calling a function $\mathsf{VRFverify}(\mathsf{pk}, \sigma, \pi)$. Yet, the proof π and the output σ does not reveal any information on the secret key sk.

In our protocol, the input value x in the VRF is a randomness rnd, extracted from the blockchain. Each client $i \in \mathcal{U}$ independently computes an VRF output σ_i on the input value rnd to determine whether or not i is selected into the pool.

III. Biased selection attack and Secure client selection problem

This section describes a simple yet effective biased selection attack and defines necessary properties of a secure client selection. First, we establish the threat model as follows.

Threat model. Our threat model extends that of previous work on secure aggregation in FL [2], [5], [29]. Instead of an honest-but-curious server, we consider a *semi-malicious* server that honestly follows the training protocol of FL, except that it tries to manipulate the selection process to its advantage. We assume that a secure aggregation protocol is used such that the server learns nothing other than the sum of the model updates in each training round as in equation (2). The server can collude with a subset of clients. We denote by $\beta \in (0, 1)$ an upper-bound on the fraction of colluding clients. The goal of the server is to learn the parameters of the victim's local model updates, from which it can infer some properties about the victim's training data.

A. Biased selection attacks

We present two different strategies to conduct this attack. First, we show that the server can viably collude with some clients $\bar{\mathcal{U}} \subset \mathcal{U}$ to learn the local models of a victim $v \in \mathcal{U} \setminus \bar{\mathcal{U}}$. Second, we demonstrate that the server can still learn some information about the victim's local model x_v and conduct inference attacks even without colluding with some clients.

Colluding attack. Let $\bar{\mathcal{U}} \subset \mathcal{U}$ be the set of clients with which the server can collude. At a particular round t, the server can extract the victim's local model x_v^t as follows:

1) The central server $\mathcal{S}$ selects the victim v and a subset of colluding clients $\mathcal{U}' \subseteq \bar{\mathcal{U}}$ and sends them the current global model G^t.
2) The selected clients compute their local model updates as normal. However, each selected colluding client $u \in \mathcal{U}'$ secretly shares their x_u^t with the server.
3) The server, via a secure aggregation protocol, obtains $S = \prod_{\{x_u^t | u \in \mathcal{U}' \cup v\}} \left[\sum_{u \in \mathcal{U}' \cup v} x_u^t\right]$, which is the sum of the clients' local models as in equation (2). Since the server knows the local models of the colluding clients, i.e., x_u^t for $u \in \mathcal{U}'$, it can extract the victim's model as $x_v^t = S - \sum_{u \in \mathcal{U}'} x_u^t$.

Note that the server cannot solely select the victim as the only client in the training round since certain secure aggregation protocols require some form of communication between the selected clients [5]. Hence, the server has to select some clients that it can collude with, i.e., $\bar{\mathcal{U}} \neq \emptyset$. In fact, those colluding clients are not necessarily real devices in the system, but can be some Sybil clients created by the server. Therefore, it is viable for the server to have some clients to collude with and conduct this attack to extract the victim's model.

Non-colluding attack. In this strategy, even if we restrict the threat model to forbid collusion between the server and the clients, the server can still learn some information about the victim's model only by manipulating the client selection process. This attack requires at least two training rounds as illustrated in Fig. 1. The attack procedure is shown below:

1) At round t, the server selects the victim v and a subset of other clients $\mathcal{U}' \subseteq \mathcal{U} \setminus v$ to conduct the training round with G^t.

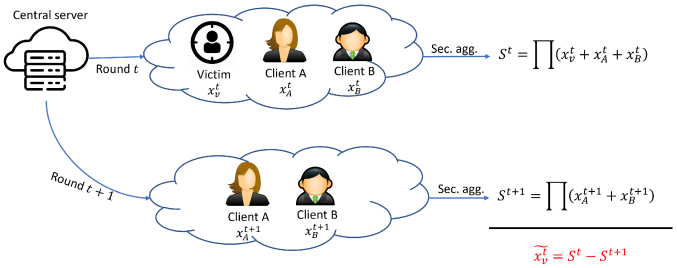


Fig. 1: Overview of the non-colluding attack. The attack works in two rounds where the victim is selected in the first round, but not in the second round. The sum of the local models' parameters of each round is obtained via secure aggregation (Sec. agg.). The difference between the sums of two rounds give an approximation $\tilde{x_v^t}$ of the victim's model.

2) Through secure aggregation, the server obtains $S^t = \prod_{\{x_u^t | u \in \mathcal{U}' \cup v\}} \left[\sum_{u \in \mathcal{U}' \cup v} x_u^t\right]$ which is the sum of the clients' models including the victim's.
3) At round $t+1$, the server re-selects the subset $\mathcal{U}'$ (without selecting v) and conducts the training round with G^{t+1}.
4) The server obtains $S^{t+1} = \prod_{\{x_u^{t+1} | u \in \mathcal{U}'\}} \left[\sum_{u \in \mathcal{U}'} x_u^{t+1}\right]$ which is the sum of the clients' models excluding the victim's.
5) The server then extracts an *approximation* of the victim's model x_v^t by $\tilde{x_v^t} = S^{t+1} - S^t$

The intuition behind this strategy is that, suppose $G^{t+1} = G^t$, for each client $u \in \mathcal{U}'$, the local model parameters in the two rounds x_u^t and x_u^{t+1} are trained with the same initialization, same algorithm and on the same training data D_u. Therefore, we can expect that $x_u^{t+1} \approx x_u^t$ for $u \in \mathcal{U}'$. As such, $S^{t+1} - S^t$ should give a good approximation of the victim's local model.

Regarding the assumption that $G^{t+1} = G^t$, the server can simply reuse G^t at round $t+1$. Moreover, even if the server chooses not to reuse G^t to avoid being suspicious, it can still honestly update G^{t+1} according to the protocol and send G^{t+1} to the clients at round $t+1$. However, this should be done only when the global model has already converged, thus $G^{t+1} \approx G^t$, so that the attack is still effective.

Membership inference attack on $\tilde{x_v^t}$. With the non-colluding attack, we show how the server is able to obtain an approximation of the victim's model $\tilde{x_v^t}$, the question remains whether the server can conduct any kind of privacy attacks on $\tilde{x_v^t}$, in other words, does $\tilde{x_v^t}$ leak any information about D_v? We investigate this issue by conducting some experiments with membership inference attacks [26], [27] on $\tilde{x_v^t}$.

In our experiments, we use the CIFAR-10 dataset [19] and partition it among 50 clients, where each one holds about 1000 training samples. For the classification task, we use a convolutional neural network composed of two convolutional layers and two pooling layers, together with one fully connected layer, and a Softmax layer at the end. ReLU is used as the activation function. In each training round, the server randomly selects 6 clients to locally train their models using

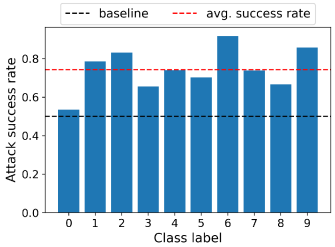


Fig. 2: The success rate of the membership inference attack [27] on $\tilde{x_v^t}$ per class label. The black dashed line shows the baseline success rate of random guessing, which is 0.5. The red dashed line shows the average attack success rate across all class labels, which is 0.743.

an Adam optimizer with 5 epochs and a batch size of 32. The global model converges after 100 training rounds. Then, at $t = 101$, the server conducts the non-colluding attack in two rounds to obtain the approximation $\tilde{x_v^t}$ of the victim's model.

Next, we conduct the membership inference attack from Shokri et al. [27] on $\tilde{x_v^t}$. This attack builds shadow models that mimic the behaviour of the victim's model (i.e., same model architecture, training data comes from the same distribution), and then uses the posteriors of those shadow models to train an attack model that determines whether a data sample is a member of the training dataset or not. In our experiments, we train the attack model as a Support Vector Machine (SVM) classifier. The attack success rate is determined by the portion of data samples that the attack model correctly predicts their membership, and it should be greater than 0.5, which is the baseline for random guessing.

Figure 2 shows the attack success rate of the membership inference attack per class label. We can see that the average success rate is about 0.743 which is much higher than the baseline. In particular, the attack can attain 0.92 success rate on the class label 6. From this result, we can conclude that even without colluding with clients, the server can still learn some information about the victim's training dataset D_v only by manipulating the client selection process.

B. Secure client selection problem

We define a new problem, called *secure client selection (SCC)* problem that asks for a protocol Π, executed by a server $\mathcal{S}$ and a set of clients $\mathcal{U}$, to select subset of clients in each training round in FL. At the end of the execution of the protocol Π, for each client j, the server $\mathcal{S}$ sends a collections of proofs $\{\omega_j^{(i)}\}_{i\in\mathcal{U}}$, in which $\omega_j^{(i)}$ is either empty or *a proof on whether or not the client i is selected*. The designed protocol is required to have three security properties, namely, *pool consistency, pool quality, and anti-targeting*.

Definition III.1 (Secure client selection problem)**.** *Let st_j be the local state of the client $j \in \mathcal{U}$ at the end of a training round. We say Π is secure iff there exists a predicate PVer_Π that takes the state st_j of a client j, a proof $\omega_j^{(i)}$ (provided by server $\mathcal{S}$) as input and output*

$$\mathsf{PVer}_\Pi(\mathsf{st}_j, \omega_j^{(i)}) = \begin{cases} 1 & \text{if } i \text{ is selected in the view of } j, \\ 0 & \text{if } i \text{ is not selected in the view of } j, \\ \perp & \text{if } \omega_j^{(i)} \text{ is empty or invalid.} \end{cases}$$

with the following properties:

- Pool consistency*: $\forall i \in \mathcal{U}$ and $\forall j_1, j_2 \in \mathcal{H}$,*

$$\Pr\left[\ \exists\ \omega_{j_1}^{(i)}, \omega_{j_2}^{(i)} \ \middle|\ \begin{array}{l} \mathsf{PVer}_\Pi(\mathsf{st}_j, \omega_{j_1}^{(i)}) = 1 \wedge \\ \mathsf{PVer}_\Pi(\mathsf{st}_{j'}, \omega_{j_2}^{(i)}) = 0 \end{array}\ \right] \leq e^{-\Omega(\kappa)},$$

  where $\mathcal{H}$ denotes the set of honest clients and κ is the security parameter.
- γ-pool quality for $\gamma \in (0, 1)$*: Let $\mathcal{P}$ be the set of selected clients, defined as:*

$$\mathcal{P} = \{i \in U : \exists j \in \mathcal{H} \text{ s.t. } \mathsf{PVer}_\Pi(\mathsf{st}_j, \omega_j^{(i)}) = 1\}.$$

  We have:

$$\Pr\left[\frac{\mathcal{H} \cap \mathcal{P}}{\mathcal{P}} \geq \gamma\right] \geq 1 - e^{-\Omega(\kappa)}.$$

- Anti-targeting*: Let $c = \frac{m}{n}$, termed the* selection probability*. We have:*

$$|\Pr[i \in \mathcal{P}] - c| \leq e^{-\Omega(\kappa)},\ \forall i \in \mathcal{U}.$$

The pool consistency ensures that the server cannot prove that a client is selected to one client while proving that it is not selected to another client. The pool quality enforces a minimum fraction of selected honest clients. Finally, the anti-targeting guarantees that all honest clients are selected with the almost the same probability.

Baseline protocol. Initially, the clients register their public keys on the blockchain. In each training round, each client computes a set of selected clients using a pre-arranged function of the registered information and the round number. The function can be implemented using blockchain's smart contracts so that all the clients agree on the same list of selected clients.

The above protocol provides pool consistency. However, it could not guarantee either pool quality or anti-targeting properties. Jumping a head, in Section V, we will show that our protocol can achieve all three security properties of SCC problem (see Table I).

Protocols	PC	PQ	AT
Baseline	yes	no	no
This work	yes	yes	yes

TABLE I: The security of the baseline protocol and our protocol. PC, PQ, and AT stand for pool consistency, pool quality, and anti-targeting, respectively.

Grinding attack on the baseline protocol. The clients, who colluded with the server, can wait til all other clients complete their registration. Then they can probe for different public keys to bias the selection of clients. Since the round number and the registration information of honest clients are known, the search

can be done to either give more chance for colluding clients to be selected (breaking the pool quality) or more chance towards a targeted honest client (breaking the anti-targeting).

Completeness of the security properties. We show that if a protocol satisfies the above three security properties in Definition III.1, the adversary cannot perform biased selection attacks as discussed in the previous subsection.

Lemma III.2. *Consider a pool selection protocol that can achieve pool consistency, pool quality, and anti-targeting properties. The probability that the server can perform colluding/non-colluding attacks is at most* $e^{-\Omega(\min\{h,\kappa\})}$, *where* $h = |\mathcal{H}|$ *and* κ *is the security parameter.*

Due to the space limit, we provide an outline of our proof. By pool consistency, all nodes have the same view on $\mathcal{P}$, the set of selected clients with a probability at least $1 - e^{-\Omega(\kappa)}$.

From the anti-targeting property, the probability that an honest client is selected concentrate around c, the selection probability. We have

$$\Pr[\text{Colluding attack}] \leq \Pr[\mathcal{H} \cap \mathcal{P} = 1] \approx \binom{h}{1} c(1-c)^{h-1} = e^{-\Omega(h)}.$$

For non-colluding attacks, let S_i, S_{i+1} be the sets of selected clients in two consecutive rounds and $s = |S_i|$.

$$\Pr[\text{Non-colluding attack}] \leq \Pr[S_{i+1} = S_i \cup \{x\}, x \in \mathcal{H}] \approx \binom{s}{1} c^{s-1}(1-c)^{h-s-1} = e^{-\Omega(h)}.$$

Combining all the probabilities of the bad events yields the bound $e^{-\Omega(\min\{h,\kappa\})}$.

IV. Client Selection Protocol

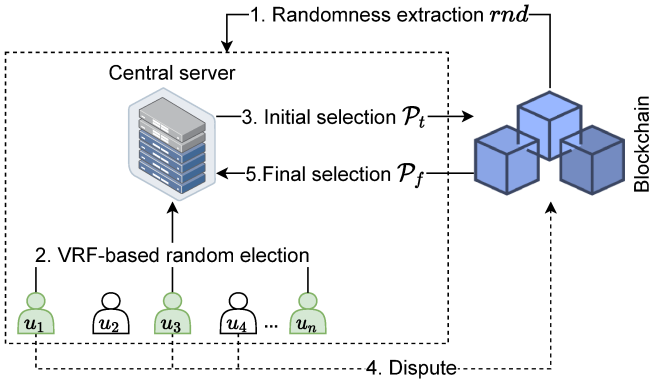


Fig. 3: The 5 steps of client selection in each training round: (1) Randomness extraction, (2) Random election, (3) Initial commitment, (4) Dispute, and (5) Final selection.

Our protocol consists of a one-time registration phase and multiple training rounds. In the registration phase, clients registered their public keys with the server. At the beginning of each training round, a random subset of clients will be selected using our client selection protocol.

Registration phase. At the beginning of the FL process, each client registers its public key with the server. The server composes a list of all public keys, submits a succinct commitment of the list to the blockchain, and provides each client with a membership proof using Merkle proof [7], a lightweight way to prove the membership in large sets.

Membership proof with sorted Merkle tree. Given a set of l values $X = \{d_1, d_2, \ldots, d_l\}$, a Merkle tree is a binary tree constructed over the hash values of d_i. The root of the Merkle tree, denoted by $\mathsf{MRoot}(X)$, can be used as a succinct representation of all the values. Knowing $\mathsf{MRoot}(X)$, we can construct a membership/non-membership proof of size $O(\log l)$ to prove whether a value x appears in X. Such a proof, denoted by $\mathsf{MProof}(x \stackrel{?}{\in} X)$, can also be verified in a time $O(\log l)$.

Let $\mathcal{U}$ be the set of registered clients, the server submits a registration transaction containing the Merkle root $\mathsf{MRoot}(\mathcal{U})$ to blockchain and sends $\mathsf{MProof}(\mathsf{pk}_i \stackrel{?}{\in} \mathcal{U})$ to each client i.

Client selection phase. As shown in Fig. 3, the selection consists of: randomness extraction, VRF-based random election, initial selection, dispute, and final selection. In randomness extraction, all clients and the server compute locally a random token rnd by hashing together the block headers in the previous round. Verifiable random functions (VRFs) [10], taking the client's public key and rnd as inputs, are employed to determine which clients are selected. In the initial selection, the server composes a list of selected clients and commits the list to the blockchain. A selected client can submit a dispute transaction if s/he is not properly included in the initial list, forcing the server to include her/him in the final selection.

We provide the details for the steps in the client selection protocol in Alg. 1. We use the height of the blockchain, or *block height* (BH), to measure time. We select a parameter $\tau = \Omega(\kappa)$ so that sent messages are received and submitted transactions are finalized within τ blocks. For a training round started at BH ℓ, the client selection protocol is executed between block heights ℓ and $\ell + 2\tau$.

Algorithm 1: Client selection protocol.

One trainning round: Consider a training round that starts at block height (BH) ℓ.

1. BH ℓ: *Randomness extraction* The server and clients extract the randomness rnd from the blockchain.
2. BH ℓ: *VRF-based random election*: Each client i computes $(\sigma_i, \pi_i) \leftarrow \mathsf{VRFprove}_{\mathsf{sk}_i}(\mathsf{rnd})$. If $\sigma_i < c2^\kappa$, the client i is qualified and sends the proof $(\sigma_i, \pi_i, \mathsf{pk}_i)$ to the server.
3. BH $\ell + \tau$: *Initial selection*: The server submits the Merkle tree root on the set of qualified clients $\mathcal{P}_t$ and sends the Merkle proof to each client.
4. BH $\ell + \tau$: *Dispute*: If a qualified client i does not receive the proof from the server, it submits a dispute transaction that consists of the proof $(\sigma_i, \pi_i, \mathsf{pk}_i)$ to blockchain.
5. BH $\ell + 2\tau$: *Final selection*: The server submits the Merkle tree root on the set of dispute clients $\mathcal{P}_f$.

1. Randomness extraction. We follow the scheme to extract the randomness in [9]. At block height ℓ, the server and all

clients compute a randomness rnd by hashing together the block headers of κ blocks created during the previous training round. The chain quality of the blockchain means that, with high probability, at least one of those blocks must be from an honest miner [12]. Thus, rnd includes at least one unbiased random source.

2. VRF-based random election. After extracting the randomness, each client i uses the VRF to check whether or not she/he is selected in this round. The client i computes the output σ_i and the proof π_i of the VRF based on the randomness rnd, i.e., $(\sigma_i, \pi_i) \leftarrow \mathsf{VRFprove}_{\mathsf{sk}_i}(\mathsf{rnd})$. If the VRF output σ_i is smaller than a given threshold, i.e., $\sigma_i < c2^{\kappa}$, the client i is *qualified* to be selected. Here, $c = \frac{m}{n}$ is the *selection probability*, i.e., the fraction of selected clients per round. If the client i is qualified, she/he sends a message $(\sigma_i, \pi_i, \mathsf{pk}_i)$ to the server.

3. Initial selection. Let $\mathcal{P}_t$ be the set of public keys of qualified clients that are verified by the server. The server submits an initial selection transaction that consists of the Merkle tree root $\mathsf{MRoot}(\mathcal{P}_t)$ to the blockchain. After the transaction is included to the blockchain, the server provides a Merkle proof $\mathsf{MProof}(\mathsf{pk}_i \overset{?}{\in} \mathcal{P}_t)$ for each client $i \in \mathcal{U}$.

4. Dispute. If a qualified client $i \in \mathcal{U}$ does not receive any Merkle proof from the server, or finds any discrepancy between the Merkle root obtained from the server to the one that the server submitted to the blockchain, it will start a dispute process. The client will submit proof of qualification directly to blockchain to force the inclusion of itself into the pool. More concretely, at block height $\ell + \tau$, the client can submit a transaction containing the tuple $(\sigma_i, \pi_i, \mathsf{pk}_i)$ to the blockchain. The client i also includes the Merkle proof $\mathsf{MProof}(\mathsf{pk}_i \overset{?}{\in} \mathcal{U})$ to show that its public key is registered.

5. Final selection. At block height $\ell + 2\tau$, the server submits a final selection transaction that contains the information of *all dispute transactions*. Let $\mathcal{P}_t$ be the set of the public keys of *dispute clients*, i.e., the clients who submitted dispute transactions. Then, similar to the initial selection, the server constructs a Merkle tree Merkle_f based on $\mathcal{P}_f$. The server submits a final selection transaction that consists of the Merkle tree root $\mathsf{MRoot}(\mathcal{P}_f)$ and sends a Merkle proof $\mathsf{MProof}(\mathsf{pk}_i \overset{?}{\in} \mathcal{P}_f)$ to each client $i \in \mathcal{U}$. Here, before adding the final selection transaction to the blockchain, the miners verify that all public keys of the dispute clients are included in the MRoot_f. The *correctness will be enforced* through smart contracts, executed by all miners in the blockchain.

V. Security Analysis

In this section, we analyze the security of our protocol in Algo. 1. We start with the construction of our $\mathsf{PVer}(\cdot)$ function, followed by the proof sketches on the three security properties, defined in Section II.

Pool membership verification function. We describe the function $\mathsf{PVer}(\mathsf{st}_j, \omega_j^{(i)})$ that verifies if the client i is selected in the view of the client j.

For each client j with the state st_j, the function $\mathsf{PVer}(\mathsf{st}_j, \omega_j^{(i)})$ extracts the blockchain C_j from the local state st_j and then proceeds as follows.

- The function verifies whether or not (1) $\mathsf{VRFverify}(\mathsf{pk}_i, \sigma_i, \pi_i) = 1$. (2) the initial selection transaction and the final commitment transaction are included in the header blockchain C_j. If those conditions do not hold, it returns $\perp$.
- If all conditions hold, i.e., the proof $\omega_j^{(i)}$ is valid, the function verifies (1) $\sigma_i < c2^{\kappa}$, and (2) pk_i is included in MRoot or in MRoot_f. If those conditions hold, it returns 1, i.e., the client i is selected.
- Otherwise, the function returns 0, i.e., the client i is not selected.

Recall that in our protocol, for each qualified client j, the server provides only $\omega_j^{(j)}$, the proof of membership of j. The proof consists of (1) the VRF output and the public key of j $(\sigma_j, \pi_j, \mathsf{pk}_j)$, (2) the initial selection transaction that consists of MRoot_t, (3) the Merkle proof $\mathsf{MProof}_t(\mathsf{pk}_j)$, (4) the final selection transaction that consists of MRoot_f, and (5) the Merkle proof $\mathsf{MProof}_f(\mathsf{pk}_j)$.

Pool from all clients' views. We say a client i is selected if there exists an honest client j and a proof $\omega_j^{(i)}$ such that $\mathsf{PVer}(\mathsf{st}_j, \omega_j^{(i)}) = 1$. Let $\mathcal{P}$ be the set of selected clients, i.e.,

$$\mathcal{P} = \{i : \exists \text{ honest client } j, \omega_j^{(i)}, s.t., \mathsf{PVer}(\mathsf{st}_{j_1}, \omega_j^{(i)}) = 1\}.$$

We first prove that all honest clients have the same view on the set $\mathcal{P}$ of selected clients. Intuitive, as the blockchain maintains an immutable ledger, all honest clients have the same view on the commitment transactions. Thus, they can extract the same list of selected clients.

Lemma V.1 (Pool consistency)**.** *For any client $i \in \mathcal{U}$, and any honest clients j_1, j_2, we have,*

$$\Pr\left[\exists\, \omega_{j_1}^{(i)}, \omega_{j_2}^{(i)} \;\middle|\; \begin{array}{l} \mathit{PVer}(\mathsf{st}_{j_1}, \omega_{j_1}^{(i)}) = 1 \wedge \\ \mathit{PVer}(\mathsf{st}_{j_2}, \omega_{j_2}^{(i)}) = 0 \end{array} \right] \leq e^{-\Omega(\kappa)}$$

We omit the proof due to the space limit and outline the main intuition. As all the honest clients have the same view on the blockchain, the valid proofs $\omega_{j_1}^{(i)}, \omega_{j_2}^{(i)}$ must have the same Merkle tree roots MRoot_t and MRoot_f. Recall that, the client i is considered to be selected if it is included in MRoot_t and MRoot_f. Thus, the honest clients have the same view on whether or not the client i is selected.

Next, we prove that the fraction of honest selected clients is proportional to the fraction of honest clients. Intuitively, the VRFs guarantee the randomness in selecting the qualified clients, i.e., the fraction of honest qualified clients is proportional to the fraction of honest clients. Plus, the dispute ensures that all honest qualified clients are selected.

Lemma V.2 (Pool quality)**.** *Let $\mathcal{H}$ be the set of honest clients in the set of selected clients $\mathcal{P}$. For $\epsilon > 0$, we have,*

$$\Pr[\frac{\mathcal{H} \cap \mathcal{P}}{\mathcal{P}} \geq \alpha(1-\epsilon)] \geq 1 - e^{-\Omega(nc - \log \kappa)}$$

where n is the number of clients, $\alpha = 1 - \beta$ is the fraction of honest clients, and c is the selection probability.

Proof. Let $\mathcal{P}' \supseteq \mathcal{P}$ be the set of qualified clients, i.e., the clients having VRF outputs smaller than $c2^\kappa$. Let $\mathcal{H}'$ and $\mathcal{M}'$ be the set of honest and colluding clients in $\mathcal{P}'$, respectively.

We prove by bounding the number of qualified colluding clients. By the chain quality property of the blockchain [12], the adversary can create at most κ blocks among the last blocks used for creating the randomness. Thus, it has at most κ randomness values to choose from. Using the Chernoff bound and union bound, for any $\epsilon' > 0$, we have

$$\Pr[|\mathcal{M}'| \geq (1+\epsilon')n(1-\alpha)c] \leq \kappa e^{\Omega(nc)} = e^{-\Omega(nc - \log\kappa)}$$

For the honest clients, since the server cannot predict the outputs of the VRFs, thus, changing the randomness will not affect the probability that honest clients are selected. Using the Chernoff bound, for any $\epsilon' > 0$, we have,

$$\Pr[|\mathcal{H}'| \leq (1-\epsilon')n\alpha c] \leq e^{-\Omega(nc)}$$

By choosing ϵ' such that $\epsilon = \frac{1-\epsilon'}{1+\epsilon'}$, we have

$$\Pr\left[\frac{|\mathcal{H}'|}{|\mathcal{P}'|} \leq \alpha(1-\epsilon)\right] \leq e^{-\Omega(nc - \log\kappa)}$$

Recall that, the honest qualified clients are included either in the initial selection transaction or the final selection transaction (through dispute). Thus, we have $\mathcal{H} = \mathcal{H}'$. Further, the selected clients must be qualified, i.e., $|\mathcal{P}| \leq |\mathcal{P}'|$. Hence, we have $\frac{|\mathcal{H}'|}{|\mathcal{P}'|} \geq \frac{|\mathcal{H}|}{|\mathcal{P}|}$. Therefore,

$$\Pr\left[\frac{|\mathcal{H}|}{|\mathcal{P}|} \leq \alpha(1-\epsilon)\right] \leq \Pr\left[\frac{|\mathcal{H}'|}{|\mathcal{P}'|} \leq \alpha(1-\epsilon)\right] \leq e^{-\Omega(nc - \log\kappa)}$$

□

Finally, we show that even when the server can choose among up to κ different randomess values, it has little chance to select a target client.

Lemma V.3 (Anti-targeting)**.** *Considering an honest client i, we have*

$$|\Pr[i \in \mathcal{P}] - c| = e^{-\Omega(\kappa)},$$

where c is the selection probability.

Proof. As we have shown in the proof of Lemma V.2, an honest client is selected if the output of its VRF is smaller than a threshold. As the adversary cannot predict the output of the VRF of the client i, for any randomness rnd, the outputs of the VRF of i cannot be distinguished with a random number. Thus, with probability c, the client i is qualified. If the blockchain is secure (with probability $1 - e^{-\Omega(\kappa)}$), the qualified client i is selected. Thus, the probability that the client i is selected is at most $c + e^{-\Omega(\kappa)}$ and at least $c - e^{-\Omega(\kappa)}$. □

Together, lemmas V.1, V.2, V.3 yield the security proof of our protocol.

Theorem V.4 (Secure pool selection)**.** *The pool selection protocol in Algorithm 1 achieves pool quality, pool consistency, and anti-targeting properties.*

VI. Experiments

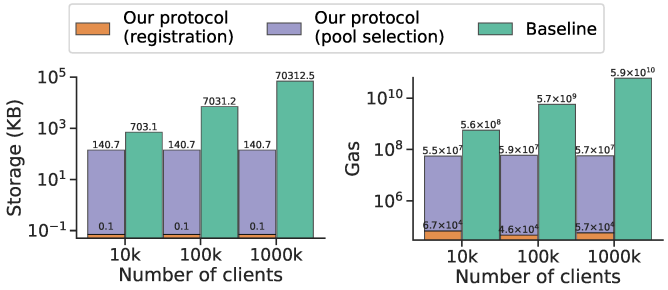
We evaluate the performance of our protocol and the (insecure) baseline protocol (section III). Further, we analyze the dispute cost associated with the server and the clients.

Setup. We assume a public blockchain, e.g., Avalanche or Solana, with Solidity smart contracts. We use the VRF in the Libsodium cryptographic library[1] and the VRF verification in Solidity at https://github.com/witnet/vrf-solidity.

We conducted all experiments on a CentOS machine Intel(R) Xeon(R) CPU E7-8894 v4 2.40GHz. We report the performance of the protocols in terms of blockchain storage cost (in KB), blockchain computation cost (in gas), and CPU time for the server and clients.

We choose the number of clients among $10k$, $100k$, and $1000k$ and set the selection probability $c = 1\%$ of that. The number of training rounds in the FL process is assumed to be $1,000$.

A. Performance



(a) Storage (b) Computation cost (gas)

Fig. 4: The storage and computation costs on the blockchain for the registration and the pool selection in $1,000$ training rounds.

Storage and computation on blockchain. As shown in Fig. 4, the storage and computation costs on blockchain for our protocol is significantly lower than those in the baseline protocol. Further, both the storage cost and the computation cost of our protocol remain constants when the number of clients increases. This contradicts the linear increase in the costs for the baseline protocol. For example, when the number of clients $n = 1000k$, the total size of transactions in the baseline protocol and in our protocol are $70.3MB$ and $0.1KB$, respectively. Similarly, the total gas cost is $5.9 \cdot 10^{10}$ in the baseline protocol and $5.7 \cdot 10^7$ in our protocol. Thus, our protocol is up to several orders of magnitude more efficient than the (insecure) baseline protocol.

CPU time. Only short computation times are needed for both the server and the clients in our protocol. For $n = 1000k$ clients, the server in our protocol takes a $7.5s$ time to construct the Merkle tree during the registration, and a $5.8s$ time per round to verify the VRF proofs of the clients. All the computation is done using a single core. We note that this

[1] https://github.com/algorand/libsodium/tree/draft-irtf-cfrg-vrf-03

time can be reduced by several folds using parallel computing (not shown in here). The computation for each client is also very short with a negligible time in the registration, and a $0.031s$ time per training round.

The baseline protocol incurs negligible computing times for both the server and the clients.

B. Dispute cost

We now measure the dispute cost of the server and the average dispute cost of each client. We consider a scenario in which the number of clients is $n = 1000k$, the selection probability $c = 1\%$, the probability that qualified clients submit a dispute transaction is 1%. We report the cost of the server and the average cost of each client in $1,000$ training round.

	Storage (KB)	Computation cost (gas)
The server	70.35	4.7×10^9
Each client	0.01	2.1×10^5

TABLE II: The storage and computation costs for dispute.

As shown in Table. II, the average dispute cost of each client is much smaller than that, paid by the server. The storage costs for the server and each client are $140.7KB$ and $0.01KB$, respectively. Similarly, the gas cost for the server and each client are 4.7×10^9 and 2.1×10^5, respectively.

VII. Related work

Secure aggregation in FL. Leveraging secret sharing and random masking, Bonawitz et al. [5] propose a secure aggregation method and apply it to deep neural networks to aggregate client-provided model updates. In [2] and [29], the authors utilize homomorphic encryption to blindly aggregate the model updates into global models. These secure aggregation protocols can scale up to millions of devices, and are robust to clients dropping out. Generic secure MPC based on secret sharing that securely computes any function among multiple parties [4], [8], [20] can also be used as secure aggregation in FL. However, they are not scalable enough due to the high complexity in both computation and communication.

Although these protocols provide strong security guarantees with respect to concealing the local model updates from the server, they are only applicable to an honest-but-curious adversary. They assume that the server honestly follows the protocol, including the random client selection. We show that the server can easily manipulate the selection process to bypass the secure aggregation and learn the local model update of a victim. We also devise a verifiable random selection protocol as a countermeasure to prevent the server from manipulating the selection of participating clients, thereby maintaining the security guarantees of secure aggregation protocols.

Integration of Blockchain and FL. Recently, there have been multiple studies focusing on integrating the immutability and transparency properties of blockchain into FL. For instance, Bao et al. [3] propose FLChain which is an auditable and decentralized FL system that can reward the honest clients and detect the malicious ones. Zhang et al. [30] propose a blockchain-based federated learning approach for IoT device failure detection. Kang et al. [15] develop a reputation management scheme using blockchain to manage and select reliable clients, thereby avoiding unreliable model updates. In [17], [21], the authors utilize blockchain for the exchange and aggregation of local model updates without a central server.

The above-mentioned systems cannot be employed directly to address the biased selection attack because they are not designed specifically for protecting client model updates. Additionally, they are not compatible to be used with a secure aggregation protocol. Our approach is different in a way that we use blockchain as a source of randomness for the client selection protocol, such that it enforces the random selection of clients, making the biased selection attack infeasible.

VIII. Conclusion

In this paper, we have shown that using the secure aggregation protocols alone is not adequate to protect the local model updates from the server. Via our proposed biased selection attack, we have demonstrated that the server can manipulate the client selection process to learn the local model update of a victim, effectively circumventing the security guarantees of the secure aggregation protocols. To counter this attack and ensure privacy protection for the local model updates, we have proposed a verifiable client selection protocol using blockchain as a source of randomness. As a result, it enforces a random selection of clients in each training round, thereby preventing the server from manipulating the client selection process. We have proven its security against the proposed attack and analyzed its computation cost with Ethereum Solidity to show that it imposes negligible overhead on FL.

Acknowledgement

This work was supported in part by the National Science Foundation under grants CNS-2140477 and CNS-2140411.

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, 2016.

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

[3] Xianglin Bao, Cheng Su, Yan Xiong, Wenchao Huang, and Yifei Hu. Flchain: A blockchain for auditable federated learning with trust and incentive. In *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)*, pages 151–159. IEEE, 2019.

[4] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 351–371. 2019.

[5] 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, 2017.

[6] Travers Ching, Daniel S Himmelstein, Brett K Beaulieu-Jones, Alexandr A Kalinin, Brian T Do, Gregory P Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M Hoffman, et al. Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141):20170387, 2018.

[7] Rasmus Dahlberg, Tobias Pulls, and Roel Peeters. Efficient sparse merkle trees. In *Nordic Conference on Secure IT Systems*, pages 199–215. Springer, 2016.

[8] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[9] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[10] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

[11] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[12] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 281–310. Springer, 2015.

[13] Meng Hao, Hongwei Li, Xizhao Luo, Guowen Xu, Haomiao Yang, and Sen Liu. Efficient and privacy-enhanced federated learning for industrial artificial intelligence. *IEEE Transactions on Industrial Informatics*, 16(10):6532–6542, 2019.

[14] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[15] Jiawen Kang, Zehui Xiong, Dusit Niyato, Yuze Zou, Yang Zhang, and Mohsen Guizani. Reliable federated learning for mobile networks. *IEEE Wireless Communications*, 27(2):72–80, 2020.

[16] Latif U Khan, Yan Kyaw Tun, Madyan Alsenwi, Muhammad Imran, Zhu Han, and Choong Seon Hong. A dispersed federated learning framework for 6g-enabled autonomous driving cars. *arXiv preprint arXiv:2105.09641*, 2021.

[17] Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Blockchained on-device federated learning. *IEEE Communications Letters*, 24(6):1279–1283, 2019.

[18] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[19] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[20] Yehuda Lindell, Benny Pinkas, Nigel P Smart, and Avishay Yanai. Efficient constant round multi-party computation combining bmr and spdz. In *Annual Cryptology Conference*, pages 319–338. Springer, 2015.

[21] Chuan Ma, Jun Li, Ming Ding, Long Shi, Taotao Wang, Zhu Han, and H Vincent Poor. When federated learning meets blockchain: A new distributed learning paradigm. *arXiv preprint arXiv:2009.09338*, 2020.

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

[23] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[24] Jason Posner, Lewis Tseng, Moayad Aloqaily, and Yaser Jararweh. Federated learning in vehicular networks: opportunities and solutions. *IEEE Network*, 35(2):152–159, 2021.

[25] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):1–7, 2020.

[26] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246*, 2018.

[27] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[28] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[29] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, pages 493–506, 2020.

[30] Weishan Zhang, Qinghua Lu, Qiuyu Yu, Zhaotong Li, Yue Liu, Sin Kit Lo, Shiping Chen, Xiwei Xu, and Liming Zhu. Blockchain-based federated learning for device failure detection in industrial iot. *IEEE Internet of Things Journal*, 8(7):5926–5937, 2020.