BlockFLA: Accountable Federated Learning via Hybrid Blockchain Architecture

Harsh Bimal Desai
The University of Texas at Dallas
Richardson, Texas, USA
hbd140030@utdallas.edu

Mustafa Safa Ozdayi
The University of Texas at Dallas
Richardson, Texas, USA
mustafa.ozdayi@utdallas.edu

Murat Kantarcioglu
The University of Texas at Dallas
Richardson, Texas, USA
muratk@utdallas.edu

ABSTRACT

Federated Learning (FL) is a distributed, and decentralized machine learning protocol. By executing FL, a set of agents can jointly train a model without sharing their datasets with each other, or a third-party. This makes FL particularly suitable for settings where data privacy is desired.

At the same time, concealing training data gives attackers an opportunity to inject backdoors into the trained model. It has been shown that an attacker can inject backdoors to the trained model during FL, and then can leverage the backdoor to make the model misclassify later. Several works tried to alleviate this threat by designing robust aggregation functions. However, given more sophisticated attacks are developed over time, which by-pass the existing defenses, we approach this problem from a complementary angle in this work. Particularly, we aim to discourage backdoor attacks by detecting, and punishing the attackers, possibly after the end of training phase.

To this end, we develop a hybrid blockchain-based FL framework that uses smart contracts to automatically detect, and punish the attackers via monetary penalties. Our framework is general in the sense that, any aggregation function, and any attacker detection algorithm can be plugged into it. We conduct experiments to demonstrate that our framework preserves the communication-efficient nature of FL, and provide empirical results to illustrate that it can successfully penalize attackers by leveraging our novel attacker detection algorithm.

CCS CONCEPTS

• **Computer systems organization → Peer-to-peer architectures**; • **Security and privacy → Malware and its mitigation**; • **Computing methodologies → Supervised learning**.

KEYWORDS

Hybrid Blockchain; Hyperledger; Ethereum; Machine Learning; Backdoor attacks; Federated Learning; Federated Averaging

CODASPY '21, April 26–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8143-7/21/04…$15.00
https://doi.org/10.1145/3422337.3447837

ACM Reference Format:
Harsh Bimal Desai, Mustafa Safa Ozdayi, and Murat Kantarcioglu. 2021. BlockFLA: Accountable Federated Learning via Hybrid Blockchain Architecture. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy (CODASPY '21), April 26–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3422337.3447837

1 INTRODUCTION

Federated Learning [22] (FL) is a multi-round machine learning protocol that is run between an aggregation server and a set of agents. FL allows the participating agents to collaboratively train a model without sharing their data with each other, or with a third-party. At a high level, each agent first locally trains a model on his dataset, and then send his model to the server for aggregation. In return, the server aggregates the received models, and returns the aggregated model back to agents for the next round of training. The rounds can simply go on until the trained model reaches some desired performance metric (e.g., accuracy) on a validation dataset maintained by the server. Since the data does not leave its owner, FL is particularly suitable for settings where privacy-sensitive data is involved. A vast range of organizations can collaborate on training a model via FL, and obtain a better performing model with respect to a model that is only trained on locally available data, while maintaining privacy of the data.

However, since the data of agents is unvetted, FL is susceptible to a wide range of attacks. We particularly consider *backdoor attacks* [3, 5] in this work as they are the biggest threat for FL to the best of our knowledge. In a backdoor attack, an adversary disturbs the training process to make the model learn a *targeted misclassification functionality* [9, 19, 30]. In centralized setting, this is typically done by *data poisoning*. For example, in a classification task involving dogs and birds, the adversary could label all blue birds in the training data as dogs in an attempt to make the model to classify blue birds as dogs at the inference/test phase. In FL, since the data is decentralized, it is unlikely that an adversary could access all the training data. Thus, backdoor attacks are typically carried through *model poisoning* in the FL context [3, 5, 32]. That is, the adversary tries to constructs an update that encodes the backdoor in a way such that, when it is aggregated with other agents' updates, the aggregated model exhibits the backdoor.

Several works try to prevent such attacks by designing robust aggregation functions [4, 6, 11, 23, 26, 27, 32, 34]. In our work, we approach this problem from a complementary angle, considering the fact that some of the proposed defenses are broken [3, 5], and there is no guarantee that existing defenses will succeed in defending against any type of adversary. Concretely, we design a framework that incorporates *accountability* to the FL framework

to discourage attackers. That is, *even if an attack is not prevented during training, our framework allows one to detect and penalize the adversarial agents later at a time when the backdoor is found in the trained model.*

To make FL accountable, we leverage *blockchains* since they are compatible with the decentralized nature of FL, provide practical immutability, and Turing-complete computation on the logged data via smart contracts. The challenge in this context is to design a blockchain architecture with a low communication and latency overhead such that it can be seamlessly incorporated to the FL data flow. We address this problem by designing a hybrid architecture consisting of both a public, and a private blockchain. This is because latency in public blockchains is too high to run any computation intensive algorithms, and the data on the public blockchains can be accessed by anyone. On the other hand, even though private blockchains are communication efficient, and address privacy challenges by allowing sensitive data to be seen only by an approved set of participants, they do not allow for public accountability since transactions are approved by a predetermined set of users, and cannot be accessed publicly. Thus, by combining public, and private blockchains in a novel architecture, we alleviate the weakness of each, and have an architecture that meets the needs of FL (cf. Figure 1 for an overview of our framework).

To our knowledge, this is the first work that implements FL over a hybrid blockchain architecture to discourage attacks by providing accountability, and penalty mechanisms. We also note that, our framework is general in the sense that, any aggregation method, such as FedAvg [29], signSGD [4], and any attacker detection mechanism can be plugged into it.

1.1 Overview of Our Contributions

The key contributions of our work are as follows:

- We propose BlockFLA: **Block**chain-based **F**ederated **L**earning with **A**ccountablity. BlockFLA is a general FL framework that aims to deter adversarial attacks by providing accountability. BlockFLA is general in the sense that, any aggregation function, and any attacker detection algorithm can be plugged into it.
- We provide a novel attacker detection algorithm for FL setting, particularly designed against pixel-pattern backdoor attacks and show its effectiveness empirically.
- We show extensive empirical evaluation of the BlockFLA on different settings.
- We analyze the security, and privacy provided by BlockFLA in detail.

The remainder of the paper is structured as follows: In Section 2, we provide the necessary background to the reader. Section 3 discusses the system architecture which illustrates a detailed outline of the on-chain aggregation, verification technique over the public chain, the penalty structure, log scheme and finally the trojan detection mechanism. In section 4 and section 5, we go over our implementation and some optimization techniques used to improve the performance of the system. In section 6 and section 7 we provide experimental evaluation results and subsequently analyze the security and privacy parameters of the system. Section 8 details the comparison of the BlockFLA system with other related blockchain based systems that integrate with Federated Learning. We conclude the paper with section 9 while providing some scope for the future work we intend to accomplish.

2 BACKGROUND

In this section, we provide the necessary background to the reader.

2.1 Federated Learning

At a high level, FL is a multi-round machine learning protocol between an aggregation server and a set of agents, in which agents jointly train a model. Formally, participating agents try to minimize the average of their loss functions,

$$\arg\min_{w \in R^d} f(w) = \frac{1}{K} \sum_{k=1}^{K} f_k(w),$$

where f_k is the loss function of k^{th} agent. For example, for neural networks, f_k is typically empirical risk minimization under a loss function L such as cross-entropy, i.e.,

$$f_k(w) = \frac{1}{n_k} \sum_{j=1}^{n_k} L(x_j, y_j; w),$$

with n_k being the total number of samples in agent's dataset and (x_j, y_j) being the j^{th} sample.

Concretely, FL protocol is executed as follows: at round t, server samples a subset of agents S_t, and sends them w_t, the model weights for the current round. Upon receiving w_t, k^{th} agent initializes his model with the received weight, and train for some number of iterations, e.g., via stochastic gradient descent (SGD), and ends up with weights w_t^k. The agent then computes his update as $\Delta_t^k = w_t^k - w_t$ and sends it back to the server. Upon receiving the update of every agent in S_t, server computes the weights for the next round by aggregating the updates with an aggregation function $\mathbf{g}\colon R^{|S_t| \times d} \rightarrow R^d$ and adding the result to w_t. That is, $w_{t+1} = w_t + \eta \cdot \mathbf{g}(\{\Delta_t\})$ where $\{\Delta_t\} = \bigcup_{k \in S_t} \Delta_t^k$, and η is the server's learning rate.

For example, original FL paper [21] and many subsequent papers on FL [3, 5, 7, 12, 32] consider weighted averaging to aggregate updates. In this context, this aggregation is referred as Federated Averaging (FedAvg), and yields the following update rule,

$$w_{t+1} = w_t + \eta \frac{\sum_{k \in S_t} n_k \cdot \Delta_t^k}{\sum_{k \in S_t} n_k}. \tag{1}$$

Another prominent aggregation method is presented in [4]. In this work, authors develop a communication efficient, distributed SGD protocol in which agents only communicate the signs of their gradients. In this case, server aggregates the received signs and returns the sign of aggregation to the agents who locally update their models using it. We refer their aggregation technique as *sign aggregation*, and in FL setting, it yields the following update rule,

$$w_{t+1} = w_t + \eta \big(\operatorname{sgn} \sum_{k \in S_t} \operatorname{sgn}(\Delta_t^k) \big), \tag{2}$$

where sgn is the element-wise sign operation.

In practice, rounds in FL can go on indefinitely, as new agents can keep joining the protocol, or until the model reaches some

desired performance metric (e.g., accuracy) on a validation dataset maintained by the server.

2.2 Backdoor Attacks and Model Poisoning

Training time attacks against machine learning models can roughly be classified into two categories: targeted [3, 5, 9, 19], and untargeted attacks [4, 6]. In untargeted attacks, the adversarial task is to make the model converge to a sub-optimal minima, or to make the model completely diverge. Such attacks have been also referred as *convergence attacks*, and to some extend, they are easily detectable by observing the model's accuracy on a validation data.

On the other hand, in targeted attacks, adversary wants the model to misclassify only a set of chosen samples while minimally affecting its performance on the main task. Such targeted attacks are also known as *backdoor attacks*. A prominent way of carrying backdoor attacks is through *trojans* [9, 19]. A trojan is a carefully crafted pattern that is leveraged to cause the desired misclassification. For example, consider a classification task over cars and planes and let the adversarial task be making the model classify blue cars as planes. Then, adversary could craft a brand logo, put it on *some* of the blue car samples in the training dataset, and only mislabel those as plane. Then, potentially, model would learn to classify blue cars with the brand logo as plane. At the inference time, adversary can present a blue car sample with the logo to the model to activate the backdoor. Ideally, since the model would behave correctly on blue cars that do not have the trojan, it would not be easy to detect the backdoor on a clean validation dataset.

In FL, the training data is decentralized and the aggregation server is only exposed to model updates. Given that, backdoor attacks are typically carried by constructing malicious updates. That is, adversary tries to create an update that encodes the backdoor in a way such that, when it is aggregated with other updates, the aggregated model exhibits the backdoor. This has been referred as *model poisoning* attack [3, 5, 32]. For example, an adversary could control some of the participating agents in a FL instance, and train their local models on trojaned datasets to construct malicious updates.

2.3 Blockchain

Blockchain was first introduced by Nakamato as the underlying ledger of the now famous Bitcoin cryptocurrency [25]. Briefly, a blockchain is an append-only, distributed and replicated database. It allows the participants of a network to collectively maintain a sequence of data in a tamper-resilient way. More importantly, it does so without a requirement for a trusted third party by invoking a consensus mechanism.

Informally, a blockchain network operates as follows: participants broadcast their data, and certain nodes called *miners* (or *validators*) gather, and store the data they receive in wrapper structures called *blocks*. Through a consensus mechanism, the network elects a leader miner in a decentralized fashion for a sequence of epochs. The epoch leader broadcast his block to the network and, having received the leaders block, other nodes store it in their local memory where each block maintains a hash-link to the previous block.

The consensus algorithm that the blockchain network deploys may depend on whether or not the network is *public*. For example, Bitcoin operates on a public network, where anyone is free to join and there is no uniform view of the network across participants. It utilizes a cryptographic puzzle called Proof-of-Work [15] to achieve consensus. This makes tampering with the order of blocks computationally infeasible when the majority of the network participants follow the protocol honestly. In *private* networks however, participants can employ more efficient consensus algorithms, such as PBFT [8]. This is because the identity and number of participants are known to every party, as access to the such networks can be arbitrarily restricted.

We provide examples for a private, and a public blockchain below, and note that there exists also hybrid architectures (as in this work), that combine both public, and private blockchains.

2.3.1 Private Blockchain: Hyperledger Fabric. Hyperledger [2] is the umbrella project for many open source blockchains. Hyperledger Fabric, a permissioned blockchain is one amongst many blockchains that holds properties like identifiable participants, high transaction throughput performance [14], low latency of transaction [28] confirmation alongside privacy and confidentiality of transactions. Hyperledger promotes the usage of smart contracts called chaincode and pluggable consensus models for the confirmation of the underlying transactions committed on the ledger. The transaction orders are maintained and are visible to all peers participating on the network.

2.3.2 Public Blockchain: Ethereum. Ethereum [33], also possess the capability to host smart contracts. However, the smart contracts published are public due to the permissionless nature of the blockchain making every transaction transparent. Each ethereum smart contract and participant have an account of its own. Ether, being the hosted cryptocurrency on the ethereum chain is required to publish contracts, call functions and send transactions over the chain. This currency is stored in a wallet possessed by every participant on the blockchain and is spent in the form of Gas to make smart contract calls. Ethereum, however, offers low transaction throughput and high latency on transaction confirmation.

3 SYSTEM ARCHITECTURE

In this paper, we propose a practical system architecture that allows any Federated Learning algorithm to run efficiently and securely while enabling auditability. Our solution maintains a multi-factor approach to securely detect the potential trojan introduced in the model over time and penalize the offending parties. There are many components to the system, each playing a critical role to accomplish the comprehensive goal.

3.1 Framework Setup

The overall BlockFLA framework assumes each participant trains the model on their local machine or on a separate Virtual Machine in the cloud. This assumption eliminates the expense of training the model on the chain and enhances data privacy. Alongside training the model locally, we consider the network to be an established TCP connection between the participants and the aggregation server, thus eliminating the overhead for establishing a connection every

time an event happens. Furthermore, the number of parameters sent by each mini-batch gradient epoch is an arbitrary number. There is no trusted server to perform any operations locally. All operations pertaining to inter-blockchain transactions are performed from any node that hosts the private blockchain and that could be either a worker node or an endorser node.

3.2 On-Chain Aggregation

In order to attain a true distributed nature of the Federated Learning algorithms, we allow the training of the model to happen off the chain by individual worker nodes participating in the convergence process. The primary advantage to execute the model training process off the chain is that individual worker nodes can provision their own computing power to dispatch the deep learning parameters to the server. We treat each worker node as individual entities sharing the common interest to get rewarded for training the model.

The server in the Federated Learning setting is represented by the Private Blockchain that performs the aggregation and sends the global updates back to the worker. As mentioned earlier, we perform all worker-blockchain communication with the help of a smart contract deployed on the private blockchain. The private blockchain is hosted on the same network as each worker node. Since this is a private blockchain, each account is issued by an authority. In this case, the node that sets up the private blockchain behaves as the ultimate authority providing membership to each worker node. Certificates are issued which are signed by the sever Certificate Authority (CA) for the worker nodes to send and receive parameters from the server by maintaining a TLS connection over SSL. The worker will then wait post upload of the parameter to the private blockchain until it receives a response from the chaincode in the form of aggregated parameters. On collection of the aggregated parameters, the worker moves to the next iteration to retrain its model and resend the updated parameters.

The server being the on chain entity will wait until each worker node has sent the model parameters. Once an update is received from every worker node, the server/private blockchain will confirm the integrity of each instance of the parameters received. The main factors checked would include the size of the update, the type of the update received and whether the sender has used the appropriate credentials to send the update.

This ends up in the server/private blockchain to conduct formation of the aggregated parameter and the blockchain will send the update to each worker node, triggering an event on the worker node to initiate the next iteration using the updated aggregate.

The system architecture illustrated in Figure 1 shows 4 worker nodes as an example participating in the Federated Learning process and generating updates in the form of parameters locally. Step 1A involves sending the local updates/parameters to the private chain and are also being logged to the secure cloud simultaneously. Logging of local updates to the secure cloud is discussed in section 3.5. Step 1B includes sending the corresponding generated SHA256 hashes of the local updates to the public blockchain for verification purposes and the motivation for this is discussed in section 3.3. Aggregation of all the local updates are being done in the private blockchain to generate a global model. Finally in Step 3, the global update in the form of aggregated parameters are being sent back to each worker node for the next epoch.

3.3 Verification over Public Ledger

The other crucial component of the BlockFLA system is the involvement of the public blockchain. Our verification system over the public blockchain is in the form of a smart contract visible to all. Every worker node on the private chain has an account on the public chain. The private blockchain maintains a one-to-one mapping of accounts on the private chain to their corresponding accounts on the public chain. Each worker must possess a wallet on the public chain with sufficient crypto-currency or transaction money to call the smart contract functions when sending updates.

The purpose of the smart contract deployed on the public ledger is three fold.

a) Storage of SHA256 Hashes of sent parameters
b) Verification function to validate whether a participant or a subset of participants have cheated in any fashion.
c) Deposit and penalty orchestration

The smart contract hosts an array per worker node to store the cryptographic (i.e., SHA256) hashes of *each parameter set* sent to the private blockchain based server. The hash value is calculated individually by each worker node off the chain on their own local machine and sent to the public smart contract as shown in Step 1B of Figure 1. The primary purpose served here is for any worker to hold any offending worker nodes accountable in the case a trojan is detected. As the public chain is transparent and immutable, the worker node can merely download the SHA256, retrieve a recreated SHA256 from the parameter store on the private chain and verify if the SHA256 stored on the public chain matches the private chain created SHA256.

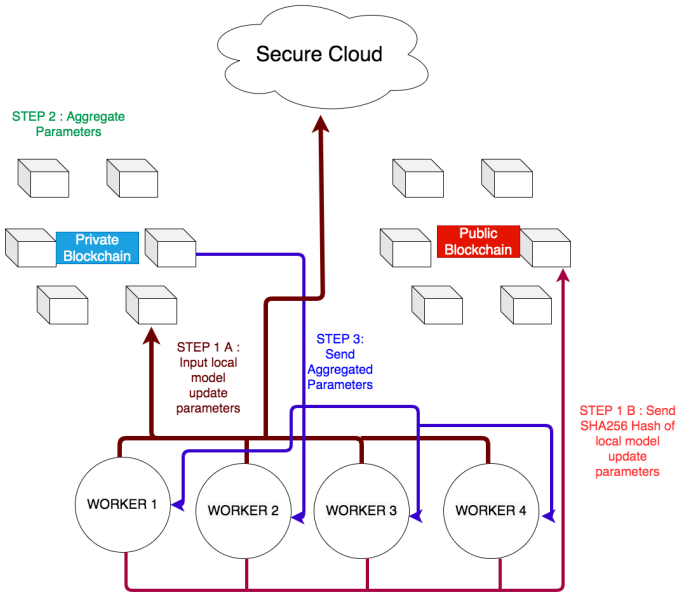


Figure 1: Federated Learning over Blockchain: Architecture Diagram

As shown in Figure 2, If in case, the SHA256 does not match, then a penalty is administered based on the penalty structure illustrated in the following section.

3.4 Penalty structure

The main incentive for worker nodes for participating as a model trainer in any Federated Learning algorithm is monetary [10]. This monetary benefit is fulfilled in the form of crypto-currency. For the purpose of participating as a worker in the Federated Learning process, each node must possess a crypto wallet and submit a deposit over the public chain. This deposit is returned to the worker at completion when convergence of the model is attained. An award is also dispatched to each worker for participating honestly. This award is coupled with a variable reimbursement to each worker that depends on the average time taken to send the model parameters to the server for aggregation. The faster the parameters are sent, the greater the award.

Although, if a worker is caught sending an update to introduce trojan by any other worker node as discussed in section 3.5.2, the deposit is lost and redistributed to the remaining worker nodes. For that reason, every worker node also has the opportunity to raise an alarm if a breach has occurred. On suspecting a breach over the public chain, a new verification contract is created and checks whether the accused actually has breached the terms of contract as shown in Figure 2. If the breach is confirmed, then the suspect loses their deposit. If no violation is confirmed, the accuser loses their deposit. This prevents unnecessary breach violation claims which may lock funds on the public chain indefinitely.

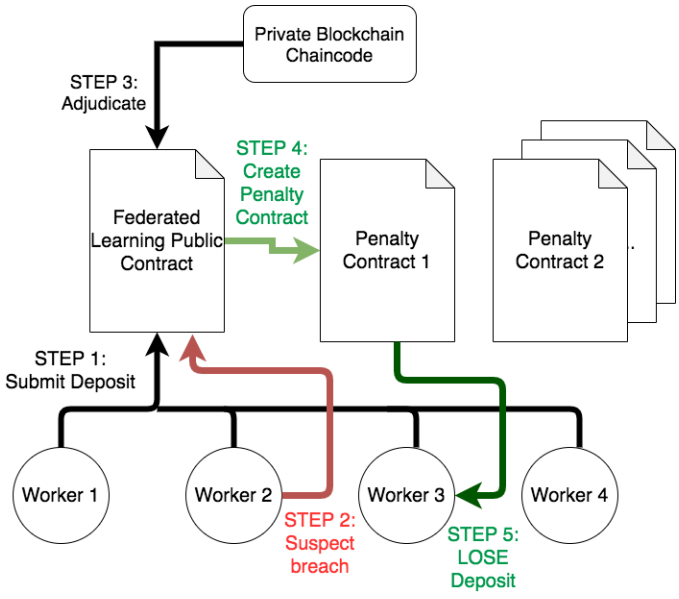


Figure 2: Breach Adjudications and Penalty Concept of Operations

3.5 Log Scheme in Secure Cloud

Logs are necessary to ensure / reinforce trojan detection. There are two important aspects with which the logging mechanism may be able to catch the perpetrator early in the process.

3.5.1 Log Publication. Logs are published to the secure cloud after every epoch by each worker node as shown in Figure 1. The logs are stored in a filesystem based on a secure cloud infrastructure where each worker has its own unique location to publish and after every epoch, the location will have the test classification results from the aggregated parameters excluding that worker node. The logs include the following data:

a) Uploaded parameters of each worker node
b) Test classification results at each epoch after aggregating parameters by excluding the parameters uploaded by that worker.

3.5.2 Anomaly alert policy. Whenever a participant suspects a breach, he/she will suspect by calling the public smart contract that holds the SHA256 Hashes of every local update made by every worker. Depending on the input of confirmation from the private blockchain, if the private blockchain is able to generate the SHA256 hashes from the local updates that match the one on the public smart contract, then the suspect was honest in uploading the parameters.

Now, for trojan detection, the private blockchain will provide a one-time link access to the location of the logs for the suspect on the secure cloud. The participant will be allowed to download the logs that contain the local parameter updates of the accused party and run the trojan detection algorithm as discussed in section 3.6. If the trojan is detected, then the suspect will be penalized according to the penalty structure as discussed in section 3.4.

3.6 An Attacker Detection Algorithm for Trojaning Attacks

We now describe a sample attacker detection algorithm, particularly designed for detecting adversaries who run pixel-pattern backdoor attacks, i.e., trojaning (cf. Figure 6). However, recall that, *our framework allows a developer to plug any kind of detection algorithm, or even multiple detection algorithms against different types of attacks, into his system. We developed this algorithm for sake of completeness, and to highlight what kind of detection algorithms can be used with our framework.* Recall that, in trojaning attacks, the adversarial task is to make the model misclassify instances from a *base class* as *target class* by using trojan patterns. To do so, an adversarial agent can simply corrupt his dataset by adding the trojan pattern to his base class instances, relabel them as the target class, and construct malicious updates by training on the corrupted dataset.

In what follows, we assume that the honest participants (i.e., verifiers) has access to *the backdoored model*, *the trojan pattern*, and the information of *base and the target class* used by the adversary. Also verifiers have access to the updates of agents that are sent during the execution of FL, as the updates are logged in the secure cloud (cf. Figure 1).

The key idea behind our detection algorithm is to do *parameter attribution* by computing the empirical Fisher Information Matrix (FIM) as done [26, 31]. At a high level, a verifier first creates a poisoned validation set using the trojan pattern of the adversary. That

is, he extracts base class instances from a clean validation dataset, adds them the trojan pattern and relabels them as the target class. He then replays the training process using the logged updates. After each round, he computes the backdoor loss on the poisoned validation dataset using the aggregated model. Then, for rounds in which backdoor loss decreases, the verifier does parameter attribution via FIM on the aggregated model using the poisoned validation dataset. This allows the verifier to list the parameters of the model in order of importance for the backdoor task, so she can identify the top-κ most important parameters for the backdoor task (where κ is a hyperparameter). Finally, the developer then measures and records the L_2 norm of each agent's update for these κ parameters.

The intuition is, *when backdoor loss decreases, we would expect the attackers contribution to be larger than contribution of honest agents for the most important backdoor task parameters.* Then, by looking at the average of recorded L_2 norms over the rounds, and making an assumption on the number of adversarial agents, the verifier can attempt to distinguish attackers. That is, the agents that contribute most on the top-κ important parameters for the backdoor task are likely to be adversarial. We illustrate the performance of our algorithm in Section 3.6 via experiments.

4 IMPLEMENTATION

Hyperledger Fabric is a permissioned blockchain infrastructure, providing a modular architecture with a delineation of roles between the nodes in the infrastructure, execution of Smart Contracts (called "chaincode" in Fabric) and configurable consensus and membership services [2]. Therefore, for our implementation, the choice of private blockchain is Hyperledger fabric. The fabric infrastructure is set up on docker containers hosted on a virtual machine inside Amazon Web Services(AWS).

We have one top level organization in the fabric implementation that is controlled by the node that sets up the private blockchain. The organization acts as the membership service provider to issue client certificates to the participants. The chaincode written in golang incorporates the logic to accept deep learning(DL) parameters in the form of batches and store them in a key-value store on the ledger. The data structure for each worker stores the deep learning parameters in a json-based tree structure. For example, in order to store the DL parameters of worker A, we divide the parameters into N parts and store them on the chain with keys A01, A02, ... AN etc.

Ethereum is one the most popular public blockchain, being the second largest cryptocurrency features a smart contract functionality formed around the principle of consensus thus eliminating the possibilities of fraud, corruption and makes the network tamperproof. Therefore, the choice of public blockchain is Ethereum Ropsten [16]. We use go-ethereum node to deploy our solidity based smart contract on the public chain. Our wallet is controlled by metamask and each participant has an account inside the wallet. The server deploys the contract making it the owner. The SHA256 hashes on the smart contract is stored in the form of arrays. Each worker has their own array and each element of the array constitutes the SHA256 hash of the DL parameters uploaded to the private chain in a particular epoch.

We set up the hyperledger fabric framework on EBS(Elastic Block Storage) backed m5.12xlarge EC2(Elastic Cloud Compute) instances. The specification of the EC2 instance is 24 cores, 192 GiB RAM and a network bandwidth of 10 Gbps. The number of threads per core being 2, the number of vCPUs allotted for processing amounts to 48.

We set up parallel loading of parameters into the private chain, and parallel aggregation of parameters on the chaincode. The SHA256 converter is set up locally on all the worker nodes.

In this work, we implemented two of the most commonly used federated learning averaging techniques: 1) *signSGD* 2) *FedAvg* (see section 2 for more details). The client application that computes the parameters for the *signSGD* or *FedAvg* alogrithm is implemented in Javascript with the help of Mxnet library. We create nodejs subroutines on the worker nodes to upload DL parameters in batches to the chaincode. A counter is implemented in the chaincode to increment its value whenever an the chaincode accepts the parameters from all the participants. No participant can upload the parameters twice in the same epoch. Once the counter's value becomes equal to the number of participants, the aggregation function is triggered calculating the aggregation based on all the participants' inputs. Another nodejs subroutine is implemented to upload the SHA256 hashes generated by the converter to the Ropsten.

In the case of *signSGD*, each worker node trains their model locally and sending the sign of each parameter in binary form to the smart contract. For the sake of simplicity, we map a negative sign to the bit 0 and the positive sign to bit 1. In the case of *FedAvg*, each worker node trains their model locally and sends the 32 bit float/real number of each parameter in float32 form to the smart contract.

For *signSGD* again, the Hyperledger Fabric chain code based server then computes the majority bit received from all the workers at each i^{th} position of the parameter. In the case of *FedAvg*, the chaincode computes the average of the parameters received from all the workers at each i^{th} parameter.

5 OPTIMIZATION TECHNIQUES

Maximizing throughput and minimizing communication overhead necessitates for the following optimization strategies we have implemented for a more reliable, practical and efficient execution of the averaging algorithm used for FL over blockchain to reach algorithmic convergence faster. To achieve this goal, we created Nodejs subroutines to convert binary parameters to base64 parameters as a compression technique. Also, we implemented a variable thread spawning mechanism to create new chaincode to perform the upload and aggregation of parameters in parallel for optimization purposes. We discuss the details of these optimizations below.

5.1 Binary compression using base64

Hyperledger Fabric does not accept inputs in a binary format when workers are sending binary based models in signSGD. The binary parameters have to be sent to the fabric chaincode in a text format. Therefore to send binary input as is, the DL parameters must be sent as characters in a string.

For example, in signSGD, for model updates involving a DL model that has 300 million parameters, if the binary representation

is converted to byte format and sent to the chaincode, we send 37.5 million bytes in the form of 75 million characters achieving a compression of 75%. This however could be improved by compressing the byte representation of the parameters into a base64 representation and decoding the base64 characters into binary on the chaincode. We use golang provided base64 decoding functions to decode the base64 inputted text into binary format. Base64 encodes 3 bytes (6 characters in a string) on 4 characters. Therefore, 37.5 million bytes (75 million characters) can be sent as 50 million base64 characters thus achieving a cumulative compression of 83.33%.

With this compression, we send fewer bytes to the Hyperledger Fabric chaincode, thus reducing communication and improving efficiency for signSGD.

5.2 Preventing key collisions

Hyperledger Fabric's underlying database uses Multiversion Concurrency Control(MVCC) to guarantee no double spending or inconsistency in the data occur. Therefore, an attempt to update the same state will result in a new version of the existing state being created to overwrite the old one.

Due to the size of the parameters being uploaded into the hyperledger fabric chaincode, we divide the input into smaller batches to avoid any throughput related errors. Hence, there is a need to parallelize sending input to the Fabric chaincode. If the uploader is not parallelized, then sequential sending of the batches will result in frequent update of the Hyperledger state. Therefore, roughly half of the transactions executed during the upload phase fail due to an MVCC related error and the Federated Learning algorithm implementation will be in an inconsistent state.

Hyperledger Fabric currently has the limitation of not being able to handle MVCC conflicts and will allow the transaction to happen resulting in an error being thrown when it tries to execute. Therefore, we design our chaincode data model so that MVCC conflicts are avoided. We separate the read operations from write operations so that we can implement them as queries and invokes respectively. We modified our data model so that every new transaction writes to a completely different key, thus mitigating the problem only to a certain extent. However, we have to record deltas and as a result our updates over key A (A being a participant) would be stored by transactions in independent keys such as A01, A02, A03. The updates implemented in this approach then get aggregated and the current state of A is reflected in the next query.

5.3 Architectural Improvements

5.3.1 Increase Endorsers and Channels. Due to usage of cloud based infrastructure and the ability to scale vCPUs, we increased the number of channels and endorsers to increase transaction throughput. We deployed multiple chaincodes on multiple channels to accept and aggregate parameters in parallel. Merely increasing the number of endorser peers do not improve efficiency. The peers are CPU-intensive and as a result have to be placed on separate VMs.

5.3.2 Usage of LevelDB. Hyperledger Fabric deploys with LevelDB in its default setting. However, in order to have a highly available setting, CouchDB is used to maintain the state database. Although, there are benefits in using CouchDB to store binary data modeled in a chaincode and having CouchDB as an external database for resiliency, the performance of LevelDB implementation of the chaincode is significantly better than the CouchDB implementation. LevelDB is a key-value store and practical implementations of it demonstrate significant performance improvement.

5.3.3 Deploying Peers on separate nodes. In Hyperledger Fabric, endorser peers are CPU intensive processes. If deployed in large numbers, they may significantly affect the transaction throughput performance. Orderers and Membership Service Providers / Organizations on the other hand are non-CPU intensive processes. Thus, the peers are deployed on separate Virtual Machines instead of all on one to improve the throughput. We set network bandwidth on the order of 10 Gbps in order for the transaction throughput to remain high.

5.4 Cache Chaincode

Cache chaincode are solely designed to terminate MVCC conflicts to avoid impairing the performance of the network. On uploading inputs to the chaincode, if an MVCC conflict occurs, the parameters can be cached onto this cache chaincode, so that it can be re-uploaded when the execution is complete. That way, if the database goes into an inconsistent state due to an error, the cache chaincode will be used to restore the state to maintain consistency. Thus, the worker only has to upload the parameters once and rely on the chaincode architecture to maintain persistence.

6 EXPERIMENTAL EVALUATION

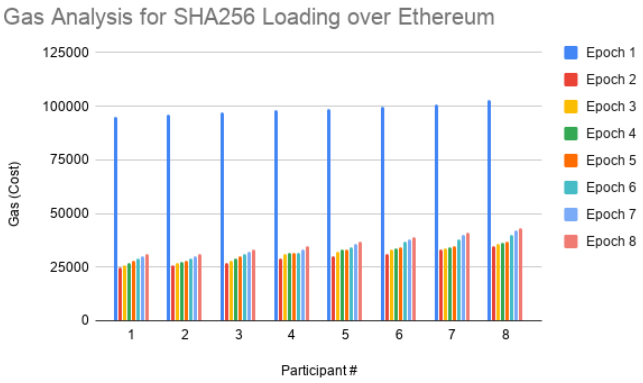
To assess the the BlockFLA system, we experiment multiple facets of the system under different scenarios. Our experimentation is pertinent to testing the impact of integrating a hybrid blockchain setting with the Federated Averaging algorithm and SignSGD algorithm, evaluating the penalty structure and security assessment of the system. We examine the effects of parallelism, deploying multiple contracts, using disparate data structures, swapping the underlying blockchain and fiscal impacts on implementing the system.

6.1 Experiment Setup

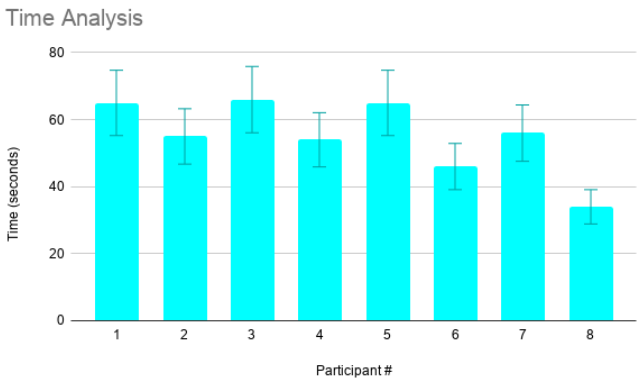
We use hyperledger provided fabric docker containers to be deployed on the m5.12xlarge EC2 instance. We have metamask installed on our browsers with the connection network pointing to the Ropsten Test Network. Our Ethereum version of smart contract is published to Ropsten at address:

0x304095B3Af015DB179CADD70dF4BC9D1fDc1aDbc

We use openssl to generate self-signed Certificate Authorities (CA) and issue client certificates to the peer nodes signed by the CAs. Nodejs scripts are written to control the deployment of chaincodes and creation of channels in a command-driven routine. A nodejs script is implemented to act as the liaison between the hyperledger fabric and ethereum networks. We use etheruem smart contract auto-generation technique for penalty payments on the public chain. Appropriate security groups are implemented for the endorsing peer based EC2 instances to communicate with each other. Each account on the Ropsten Ethereum chain are given 10 Ether to begin with and are requested from the Ropsten Ethereum faucet available on the internet. Binary to base64 converter is implemented for



(a) Gas Analysis for SHA256 Loading over Ethereum



(b) Time Analysis for SHA256 Loading over Ethereum

Figure 3: *Experimental results of Cost Analysis over Public Blockchain Ethereum.*

signSGD to convert participant provided parameters into base64 to upload to the chaincode. We use the peer binary provided by Hyperledger Fabric to upload parameters in a repeatable fashion.

Since our framework implements FL algorithms as it is, *our implementation does not result in any difference with respect to the accuracy of the generic FL models.* Hence we do not report any accuracy results here. On the other hand, at each epoch, depending on the DL model learned and the averaging scheme used, the performance may greatly change due to blockchain based smart contract characteristics. To understand the factors that impact the overall performance, we experimented with a single round of update under different aggregation schemes with varying DL model sizes. In order to set up our experiments, we adjusted the DL model sizes (i.e., the number of trainable parameters used by DL model) based on the commonly used DL models [1]. For example, classic DL architecture such as AlexNet has around 60 million trainable parameters. On the other hand, newer architectures such as VGG16 has around 160 million trainable parameters. In our experiments, we measure the performance by varying the number of parameters updated at each epoch ranging from 50 million to 300 million. This range covers most of the commonly used DL models [1].

6.2 Results

6.2.1 1 contract vs N contracts (SignSGD). Each client loads upto 10 KB (20,000 characters) of parameters in every iteration of uploading the DL parameters per epoch. That means per function call the clients are loading up to 13,300 base64 characters. With both, our local and AWS setup, we have compared the cases of using a single contract versus multiple contracts to upload the parameters and subsequently perform aggregation. Parameter loading to the hyperledger fabric framework is done in parallel in case there are multiple contracts. Each series of uploads to the smart contract will have its own thread running in the background. For a single thread / single contract system, the parameters are loaded in at the maximum 350 transactions with an event update post each update to give the client application the notification to send another update. Since the parameters are uploaded sequentially, the amount of time taken to upload the parameters is up to 5 minutes in both the local setup and AWS setup. This seems to be a reasonable time considering that we are using only a single contract for uploading the parameters sequentially. There is no monetary/gas cost analysis provided since it is a private blockchain and there are no wallets in a private blockchain. We then increased the number of contracts by a factor of 2 and compared the time taken for uploading the

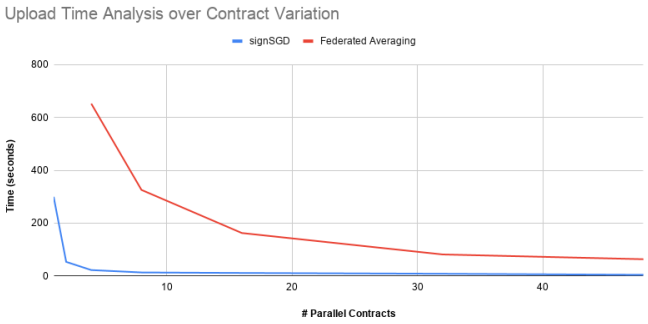


Figure 4: Upload Time Analysis over Contract Variation

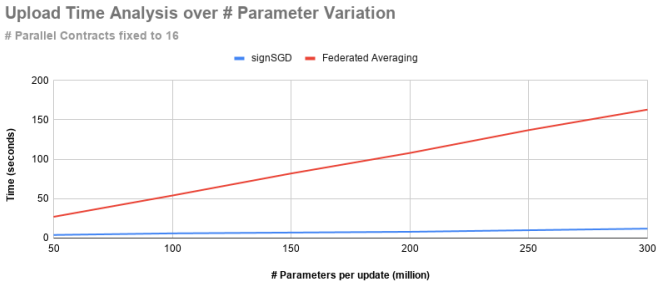


Figure 5: Upload Time Analysis over Parameter Variation

Table 1: Deploy and Aggregation Time Analysis over Contract Variation

# Contracts Deployed	Contract Deploy Time	Parameter Aggregate Time
1	45 seconds	4 seconds
2	71 seconds	3.5 seconds
4	93 seconds	3.5 seconds
8	144 seconds	3 seconds
16	243 seconds	2.8 seconds
32	443 seconds	2.5 seconds
48	503 seconds	2.3 seconds

(a) signSGD

# Contracts Deployed	Contract Deploy Time	Parameter Aggregate Time
1	35 seconds	6.6 seconds
2	48 seconds	6.5 seconds
4	70 seconds	6.5 seconds
8	132 seconds	7.3 seconds
16	258 seconds	6.8 seconds
32	420 seconds	6.5 seconds
48	510 seconds	6.5 seconds

(b) FedAvg

parameters to the framework. With our setup, we were able to upload all parameters within 60 seconds.

On further increasing the number of contracts, the time to upload improved significantly. With 32 parallel contracts, we were able to achieve upload time of 9 seconds for any signSGD setting. And if we push the system to its limits by uploading all the parameters in 48 parallel contracts, then the upload time is 5 seconds. This is the primary advantage of using a private blockchain instead of a public one simply because the transaction throughput is so high. Having achieved 20,000 transactions per second [13], in the event we have better processing power, we can potentially upload all the parameters within a fraction of a second.

The system we were using was a 48 vCPU system. As a result, even if we use more than 48 threads to upload all the parameters, it will have a negligible impact on the performance simply because all the vCPUs have been taken up by the 48 threads earlier.

The aggregation time as seen in Table 1a, for any Federated Learning signSGD setting is always less than 5 seconds. The results shown in Table 1a are recorded to demonstrate the worst case timing. Realistically, the parameters sent by each worker node would not be above 300 million parameters. Therefore, we have shown that even in the worst case scenario, aggregation happens within 5 seconds.

Thus, if we add up the results seen in Table 1a and Figure 4, for each worker node, with the current system, for 5-10 epochs, the total time taken for the aggregation and parameter loading to run takes less than 1.5 minute.

6.2.2 1 contract vs N contracts (FedAvg). Each client loads 130 KB (33000 float32) in every iteration of uploading all Federated Averaging parameters per epoch. We use the same experimental setup used for *signSGD*. For a sequential upload of parameters, the amount of time taken to upload the parameters is 25 minutes in both the local setup and AWS setup. Due to the usage of events, there is a drastic increase in time.

On increasing the number of contracts as seen in Figure 4, the time to upload improved significantly. On uploading all 330 million parameters in 48 parallel contracts, the upload time is under 1.5 minutes. Being a private blockchain instead of a public one implies the transaction throughput is so high.

The aggregation time as seen in Table 1b, for any Federated Averaging setting is less than 10 seconds constantly because it is being done on the ledger and will consume only one transaction to return the results. Similar to Table 1a, the results shown in Table 1b demonstrate the worst case timing for aggregating 300 million parameters.

Thus, if we add up the results seen in Table 1b and Figure 4, for each worker node, with the current system, for 5-10 epochs, the total time taken for the complete *FedAvg* algorithm to run takes less than 12 minutes.

6.2.3 Performance Analysis on Parameter Variation. As shown in Figure 5, on increasing the number of parameters to be uploaded by each Federated Learning algorithm by keeping the number of parallel contracts deployed constant(her 16 parallel contracts), the time required to upload all parameters into the private blockchain increases gradually. For signSGD, it takes about 4 seconds to upload 50 million parameters to the private chain and 12 seconds to upload 300 million parameters to the private chain. Similarly, for Federated Averaging, it takes 27 seconds to upload 50 million parameters to the private chain and 163 seconds to upload 300 million parameters to the public chain. This tells us that with the current optimizations in place, we are able to achieve communication efficient Federated Learning over the Hybrid Blockchain setting.

6.2.4 Hybrid chain vs all Public chain. We deployed a parallel solidity smart contract that performs the same operations as the golang hyperledger chaincode and deployed it on the ethereum private blockchain. There was a drastic slowness observed when performing Federated Averaging operations over Ethereum. The throughput attained when inputting parameters to the ethereum smart contract more than 1300 seconds for *FedAvg* versus 64 seconds per 300 million parameters for *FedAvg* for the hyperledger fabric framework. The time taken to perform aggregation was also of the order of 450 seconds for ethereum versus less than 10 seconds for hyperledger fabric. Since a private deployment of the Ethereum blockchain was set up, there are no incurred transaction costs incurred to any worker node.

6.2.5 Gas and Time Analysis over Public Chain. The price of 1 Ethereum Gas is 0.02 μ Ether. The amount of gas it takes to upload the SHA256 Hashes to the Ropsten Ethereum public chain is nearly 95000 Gas (.0019 Ether) for the first call of the first epoch. On subsequent epochs, each call takes nearly 25000 Gas(0.0005 Ether) to upload the SHA256 Hashes as shown in Figure 3a. Cost to upload

the Hash value is more for the first call because it initializes the data structure on the smart contract, thereby being more expensive than merely updating and appending values to the array that is already created. As seen in Figure 3b, the SHA256 Hash upload from the client is variant between 35 seconds to 75 seconds. This means each transaction confirmation over the public chain is independent of size of the hash and therefore unreliable to store the real parameters on the public chain if performance is taken into consideration.

6.3 Performance of our Attacker Detection Algorithm

We now illustrate the performance of the attacker detection algorithm we described in Section 3.6 via experiments. The general setting of our experiments are as follows: we simulate FL for R rounds among K agents where F fraction of them are corrupt. The backdoor task is to make the model misclassify instances from a *base class* as *target class* by using trojan patterns. That is, a model having the backdoor classifies instances from base class with trojan pattern as target class (see Figure 6). To do so, we assume an adversary who corrupts the local datasets of corrupt agents by adding a trojan pattern to base class instances, and relabeling them as target class. Other than that, adversary cannot view and modify updates of honest agents, or cannot influence the computation done by honest agents and the aggregation server. At each round, the server uniformly samples $C \cdot K$ agents for training where $C \leq 1$. Those agents locally train for E epochs with a batch size of B before sending their updates.

Figure 6: Samples from trojaned base class, and target class. The trojan pattern is a 5-by-5 plus pattern that is put to the top-left of base class instances. The goal of adversary is to make the model classify instances of dog class with the trojan pattern (left) as a horse (right).

We tested our detection algorithm on CIFAR10 [18] dataset by using a 5-layer convolutional neural network consisting of about 1.2M parameters with the following architecture: two layers of convolution, followed by a layer of max-pooling, followed by two fully-connected layers with dropout. Hyperparameters used in all experiments can be found in Appendix A. We were able to detect the adversarial agents perfectly in three out of these four settings. We briefly summarize our results below.

In the first two settings, we have a rather small setup of 10 agents, where one of them is adversary. We plot the L_2 of adversarial contribution for the rounds in which backdoor loss decreases in Figure 7. The attacker is Agent 0, and as can be seen, he stands out from the rest by having the largest L_2 contribution to most important backdoor parameters.

Our last two settings are somewhat more realistic for FL. We have a setup of 100 agents, where 10 of them are corrupt, and

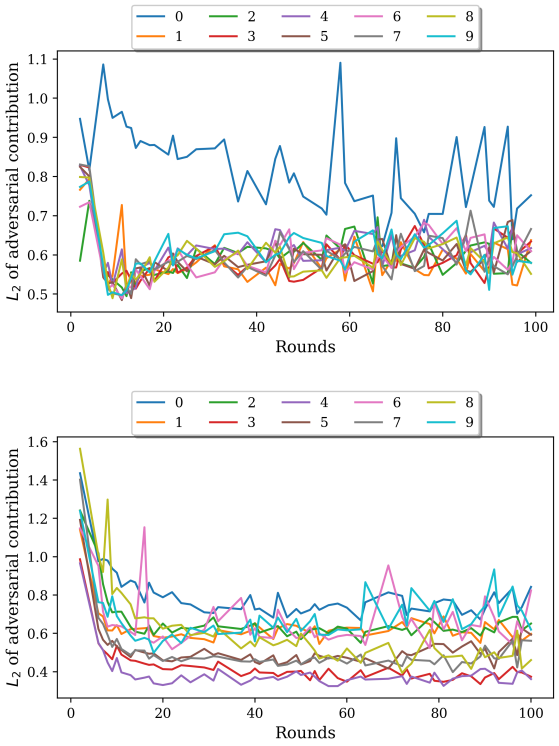


Figure 7: Results of trojan detection algorithm in iid (top), and non-iid (bottom) settings. In iid setting, the adversarial agent (Agent 0) stands out very clearly. In non-iid setting, the variance of contribution of agents' seem to be higher. Yet, if we compute the average L_2 contribution over the rounds, Agent 0 is at the top with a value of 0.78, followed by Agent 9 (0.67) and Agent 6 (0.65). In iid setting, data is distributed uniformly between agents, and in non-iid setting, we distribute each class by sampling from a Dirichlet distribution with a concentration of 0.5

where in each round, the aggregation server uniformly samples 10 agent out of 100. Our detection algorithm were able to perfectly distinguish adversarial agents for iid case. That is, *when we list agents by adversarial contribution, the top-10 consisted of solely adversarial agents.* However, for non-iid setting, it included some non-adversarial agents. For that case, in top-5, we had 4 adversarial agents, but the places between 6-10 were non-adversarial agents. We suspect this is because the variance of adversarial contributions are relatively higher in larger, and non-iid settings. Again,*we stress that our framework could be used with other detection algorithms.*

7 SECURITY AND PRIVACY ANALYSIS

In this section, we provide an overview of the security and privacy assurances of the proposed system. Especially, we discuss how the BlockFLA system handles attacks from the adversaries discussed in section 2.4.

7.0.1 Security Analysis. Since aggregation done on the private blockchain, as long as the 51% of the participants are honest (needed for convergence of the consensus algorithm used), the averaging needed for the FL will be correctly executed. Therefore, the aggregation will be done according to the specified protocol. Therefore, any single attacker cannot change the aggregation process.

On the other hand, a malicious worker node can try to attack the system by introducing a backdoor /trojan. Assuming the detection technique work with high probability, the participants may be held accountable for an attack. Since all participants commit the SHA256 Hashes of their parametric updates to the public smart contract, we can easily detect incorrect disclosure of updates. If the participant introduces a backdoor, then the federated learning algorithm can be re-simulated on the private chain using the updates as the test data using the appropriate off-chain detection technique.

7.0.2 Privacy Analysis. The privacy protection of parameters sent over the private blockchain is achieved with the help of channels. For access control over private chain, each worker is connected to the server by a separate channel. Since each participant can join the hyperledger framework only with authorization from the membership service provider, the data parameter updates sent between the worker nodes and the server node remain on the chain concealed from those who have no account on the private chain. Each participant's privacy over the public blockchain is achieved with the help of hashing parameters. Log files over the public cloud is encrypted and accessible only by the server.

8 RELATED WORK

A privacy-preserving solution for federated learning has been proposed in BlockFlow [24], they consider differential privacy utilizing Laplacian noise to avoid disclosing client information to other other clients. Still, this does not address the possibility of detection of an anomaly based trojan being introduced during the learning process. BlockFlow performs aggregation and evaluation of client models over the ethereum blockchain which is a public chain. However, implementing both the aggregation and penalty mechanism over the public chain drastically slows down the federated learning process. Sending models over the public chain results in expensive cost in terms of gas/ether incurred by both the client and the server which is not beneficial to the algorithm. This issue is addressed by our system. Since federated learning is eminently reliant on communication between the worker and server nodes, we have proposed and implemented a hybrid blockchain architecture to perform the aggregation over the private chain and the penalty mechanism over the public chain. Also having a separate smart contract between each client and server by the auto contract generation will keep the penalty mechanism more structured and robust.

The BC-FL framework proposed in [20] discusses the integration of a public blockchain with federated learning to prevent malicious worker nodes or UEs (User Equipment) as they call it from poisoning the learning process. They identify the issue that federated learning aggregation being conducted on a central server is a single point of failure in the federated learning process. The agenda of their system primarily involves model recording and publishing over the public chain and an incentive mechanism to motivate miners and discourage lazy nodes. To improve security while maintaining efficiency, they propose a Digital Signature system to recognize malicious clients by verifying learning results and a reputation system to reduce verification delay of a high-reputation miner. Although, the proposal makes an attempt, the system itself discourages efficiency since the federated learning aggregation is being conducted solely on the public chain. As we have seen in [17], Federated learning without a communication-efficient framework will take a considerable amount of time to reach convergence. Public blockchains like ethereum are inherently slow to commit transactions and this will cause a significant impact to Federated Learning performance. The BlockFLA framework, however, addresses this issue by implementing a hybrid blockchain framework. Secondly, the Digital Signature and the reputation system only prevents external attackers from introducing bad models. Moreover, if a backdoor is introduced, it is not possible to be detected by the participants themselves. As we discussed, we can detect a backdoor introduced with the help of our framework.

9 CONCLUSIONS AND FUTURE WORK

Federated Learning is an upcoming communication-efficient machine learning technique that trains a global model while maintaining data local. However, due to an existing risk of backdoor attacks, the poisoning of the global model can yield ineffective results in terms of classification and prediction which defeats the purpose of the overall goal of FL.

In order to address this aforementioned problem, we have proposed a general blockchain framework that integrates both public and private blockchains to achieve decentralization, immutability and transparency to discourage backdoor attacks on federated learning algorithms by holding the responsible parties accountable. We have shown that our framework facilitates the adaptation to any federated learning algorithm rendering a plug-and-play setting. We have presented the implementation of the Federated Averaging and signSGD algorithm over our general blockchain framework and based on our empirical results, conclude that our proposed approach maintains the communication-efficient nature of these algorithms.

We have also proposed and implemented an algorithm that runs off the chain server-side to successfully detect the attacker who introduced the trojan for the Federated Averaging algorithm. Our experiments show that we are able to effectively detect the participant who has introduced the model poisoning attack. Furthermore, the blockchain-based penalty system proves to be an efficient deterrent for an attacker to carry out any model poisoning attacks by injecting backdoors/trojan.

As a future work, we plan to develop a trojan detection algorithm for signSGD. This seems to be challenging due to the communication involving only bits instead of real numbers. Still, we believe this is feasible by using more trojan examples. In addition, we will look into integrating other federated learning aggregation algorithms in our system.

ACKNOWLEDGMENTS

The research reported herein was supported in part by NIH award 1R01HG006844, NSF awards, CNS-1837627, OAC-1828467, IIS-1939728, DMS-1925346, CNS-2029661 and ARO award W911NF-17-1-0356

REFERENCES

[1] [n.d.]. Common architectures in convolutional neural networks. https://www.jeremyjordan.me/convnet-architectures/

[2] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 30, 15 pages. https://doi.org/10.1145/3190508.3190538

[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*. 2938–2948.

[4] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. 2018. signSGD: Compressed Optimisation for Non-Convex Problems. arXiv:1802.04434 [cs.LG]

[5] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. 2019. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*. 634–643.

[6] Peva Blanchard, Rachid Guerraoui, Julien Stainer, et al. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*. 119–129.

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

[8] Miguel Castro and Barbara Liskov. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.* 20, 4 (Nov. 2002), 398–461. https://doi.org/10.1145/571637.571640

[9] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).

[10] H. Desai, K. Liu, M. Kantarcioglu, and L. Kagal. 2018. Adjudicating Violations in Data Sharing Agreements Using Smart Contracts. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 1553–1560.

[11] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. 2020. Mitigating Sybils in Federated Learning Poisoning. *arXiv preprint arXiv:1808.04866* (2020).

[12] Robin C Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* (2017).

[13] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. 2019. Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 455–463.

[14] Y. Hao, Y. Li, X. Dong, L. Fang, and P. Chen. 2018. Performance Analysis of Consensus Algorithm in Private Blockchain. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. 280–285.

[15] Markus Jakobsson and Ari Juels. 1999. *Proofs of Work and Bread Pudding Protocols(Extended Abstract)*. Springer US, Boston, MA, 258–272.

[16] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey. 2018. Measuring ethereum network peers. In *Proceedings of the Internet Measurement Conference 2018*. 91–104.

[17] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2017. Federated Learning: Strategies for Improving Communication Efficiency. arXiv:1610.05492 [cs.LG]

[18] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2009. CIFAR-10 (Canadian Institute for Advanced Research). (2009). http://www.cs.toronto.edu/~kriz/cifar.html

[19] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning Attack on Neural Networks. In *25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society.

[20] Chuan Ma, Jun Li, Ming Ding, Long Shi, Taotao Wang, Zhu Han, and H. Vincent Poor. 2020. When Federated Learning Meets Blockchain: A New Distributed Learning Paradigm. arXiv:2009.09338 [cs.NI]

[21] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).

[22] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2016. Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv:1602.05629 [cs.LG]

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

[24] Vaikkunth Mugunthan, Ravi Rahman, and Lalana Kagal. 2020. BlockFLow: An Accountable and Privacy-Preserving Solution for Federated Learning. arXiv:2007.03856 [cs.LG]

[25] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. *Cryptography Mailing list at https://metzdowd.com* (03 2009).

[26] Mustafa Safa Ozdayi, Murat Kantarcioglu, and Yulia R Gel. 2020. Defending Against Backdoors in Federated Learning with Robust Learning Rate. *arXiv preprint arXiv:2007.03767* (2020).

[27] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. 2019. Robust aggregation for federated learning. *arXiv preprint arXiv:1912.13445* (2019).

[28] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong. 2017. Performance Analysis of Private Blockchain Platforms in Varying Workloads. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. 1–6.

[29] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. 2019. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems* (2019).

[30] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*. 6103–6113.

[31] Neta Shoham, Tomer Avidor, Aviv Keren, Nadav Israel, Daniel Benditkis, Liron Mor-Yosef, and Itai Zeitak. 2019. Overcoming Forgetting in Federated Learning on Non-IID Data. *arXiv preprint arXiv:1910.07796* (2019).

[32] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. 2019. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963* (2019).

[33] Gavin Wood. [n.d.]. Ethereum: A secure decentralised generalised transaction ledger. ([n. d.]).

[34] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. 2018. Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates. In *International Conference on Machine Learning*. 5650–5659.

A HYPERPARAMETERS OF EXPERIMENTS

We remind the notation we used for our experiments, and report the hyperparameters accordingly.

- R: Number of rounds
- K: Total number of agents
- F: Fraction of corrupt agents
- C: Fraction of selected agents for training in a round
- E: Number of epochs in local training
- B: Batch size of local training
- η: Server's learning rate
- κ: Number of parameter to compute L_2 norm for (cf. Section 3.6.)

R	K	F	C	E	B	η	κ
100	10	0.1	1	2	256	1	1000

Table 2: Hyperparameters for small setting.

R	K	F	C	E	B	η	κ
200	100	0.1	0.1	5	32	1	1000

Table 3: Hyperparameters for large setting.