BEAS: Blockchain Enabled Asynchronous & Secure Federated Machine Learning

Arup Mondal*, Harpreet Virk*, Debayan Gupta

Ashoka University
arup.mondal_phd19@ashoka.edu.in harpreet.virk@alumni.ashoka.edu.in, debayan.gupta@ashoka.edu.in

Abstract

Federated Learning (FL) enables multiple parties to distributively train a ML model without revealing their private datasets. However, it assumes trust in the centralized *aggregator* which stores and aggregates model updates. This makes it prone to gradient tampering and privacy leakage by a malicious aggregator. Malicious parties can also introduce backdoors into the joint model by poisoning the training data or model gradients. To address these issues, we present BEAS, the first blockchain-based framework for N-party FL that provides strict privacy guarantees of training data using gradient pruning (showing improved differential privacy compared to existing noise and clipping based techniques). Anomaly detection protocols are used to minimize the risk of data-poisoning attacks, along with gradient pruning that is further used to limit the efficacy of *model-poisoning* attacks. We also define a novel protocol to prevent premature convergence in heterogeneous learning environments.We perform extensive experiments on multiple datasets with promising results: BEAS successfully prevents privacy leakage from dataset reconstruction attacks, and minimizes the efficacy of poisoning attacks. Moreover, it achieves an accuracy similar to centralized frameworks, and its communication and computation overheads scale linearly with the number of participants.

Introduction

Robust Machine Learning (ML) models require large amount of heterogeneous training data to obtain accurate results of any practical significance. In most scenarios, this data is often scattered across mutually distrusting entities that cannot directly share their secret private data with each other, or with a centralized aggregator, due to privacy regulations that restrict centralized collection (Sav et al. 2020; Ramachandran et al. 2021). Federated Learning (FL) (McMahan et al. 2017) enables multiple parties to distributively learn a shared model without revealing their private data; each party trains a *local* model using their own data, and exchanges only the model gradients with a centralized FL *server* or *aggregator*. The *aggregator* is responsible for storage and exchange of these gradients. It also periodically merges them, generally by taking their average, to generate a new *global* model that is then used in subsequent *local* rounds for pre-training. The aggregator is thus a central player that potentially represents a single point of failure (Sav et al. 2020).

(Bonawitz et al. 2019) proposed an improved FL framework by designing a hierarchical network of FL aggregators, each of which controls its own sub-population headed by a centralized coordinator. Although this reduces the risk of a malicious aggregator by distributing the task across multiple sub-networks, the architecture is not inherently free from centralized dependency (Sav et al. 2020).

(Song and Mittal 2020) showed that model gradients can sometimes unintentionally memorize features from training data, which can be exploited by a malicious aggregator to reconstruct sensitive information about it. This can be solved using various cryptographic techniques (Perry et al. 2014; Zhang et al. 2020; Di Crescenzo et al. 2014; Gupta et al. 2016; Mood et al. 2016), though these significantly impact the efficiency of the framework (Phong et al. 2018). Differential privacy (DP) (Dwork 2006) techniques like addition of noise or clipping of gradients can also be used to limit leakage of privacy. However, obfuscating model gradients using DP techniques often impacts accuracy (Lyu et al. 2020; Abadi et al. 2016). Moreover, the aggregator cannot examine the data used for training: malicious parties can train their *local* models using poisoned datasets, or tamper with their gradients, to introduce backdoors into the shared model (Jagielski et al. 2020; Bagdasaryan et al. 2020). A malicious aggregator can also skew the shared model by biasing contributions of preferred parties (Li et al. 2020). Data poisoning attacks have been addressed with anomaly detection algorithms such as *l-nearest aggregation* (Chen et al. 2018) and Multi-KRUM (Blanchard et al. 2017), where deviation or performance of updates is evaluated relative to the majority. However, (Bagdasaryan et al. 2020) demonstrates attacks where an adversary can use *constrain-and-scale* techniques to generate gradients that can evade anomaly detection.

Researchers have investigated eliminating the centralization in FL using decentralized frameworks. However, most of the current work primarily addresses only a subset from - secure design, byzantine tolerant gradient aggregation, protection from poisoning attacks, data privacy, client motivation, fairness guarantees, scalability, and network efficiency. Currently, only (Chen et al. 2018; Shayan et al. 2018) address a majority of these issues, but not all. Moreover, prior

*Corresponding authors, authors ordered alphabetically.

work does not consider the improved privacy protection offered by gradient pruning (Zhu, Liu, and Han 2019), or the critical issue of premature convergence: the *global* model can forget previously learned features if the *local* models prematurely converge while training on a large dataset.

We propose BEAS, the first blockchain enabled framework that allows collaborative training and evaluation of ML tasks in a distributed setting with strict privacy guarantees and security against adversarial clients. Our primary motivation to use blockchain are smart contracts, which can function as a trusted distributed application and be used for storage, exchange, and merging of gradients, eliminating centralized dependence entirely. Using a fully-decentralized blockchain, BEAS can enable secure FL executions with different types of layered architectures, such as feed-forward NN and CNN, on datasets that are heterogeneously distributed among N parties. Moreover, BEAS is designed to be scalable with parallel FL models running on the same platform using multi-channel blockchain architecture. This can be useful in various research and production applications.

Contributions. Our main contributions as follows:

- Blockchain-based framework for decentralized N-party (unbounded N) FL ensuring strict privacy guarantees of training data using gradient pruning based DP, and resistance to poisoning attacks.
- Minimize risk of data poisoning using a combination of protocols to identify adversaries: (i) Multi-KRUM (Blanchard et al. 2017) is used to guarantee resiliency from independent adversaries; and (ii) FoolsGold (Fung, Yoon, and Beschastnikh 2020) is used to identify Sybil groups.
- Implement and compare various DP techniques (Dwork 2006) to prevent direct leakage of training data from shared gradients. Our experiments show gradient pruning (GP) is more effective than existing DP techniques: it prevents reconstruction of training data from shared model gradients with minimal impact on performance, and defends against model poisoning. GP has not been used in prior work for privacy (GP's primary use: gradient compression). We also evaluate the efficiency of GP against model-poisoning attacks. To the best of our knowledge, BEAS is the first approach for decentralized privacy-preserving FL to use GP for improved differential privacy, and improved defense against model poisoning.
- BEAS is first to explore premature convergence in decentralized FL; we train using small clusters of data to prevent this. We're also first to support parallel FL collaboration on multiple tasks using the same platform with multi-channel blockchain architecture.
- We perform extensive experiments on multiple datasets and different NN architectures with promising results: BEAS achieves training accuracy on par with both – centralized and non-privacy preserving decentralized approaches. BEAS can train a 3-layer NN with 64 neurons per hidden-layer with the training dataset split amongst 20 participating clients on the MNIST (LeCun and Cortes 2010) dataset in 8.73 minutes, and on the Malaria Cell Image (Rajaraman et al. 2018) dataset in 16.11 minutes.

For ease of access, all of our code and experiments are available at: https://github.com/harpreetvirkk/BEAS.

Comparative Analysis

We compare BEAS against existing *state-of-the-art* frameworks for decentralised FL. BEAS uses multi-channel permissioned blockchain to store all model gradients, which enables rapid scalability, auditability, transparency, and trust amongst collaborating entities. Differential privacy (Dwork 2006) is used to obfuscate model gradients to prevent leakage of sensitive information of private data, and FoolsGold (Fung, Yoon, and Beschastnikh 2020) with Multi-KRUM (Blanchard et al. 2017) is used to provide resiliency from adversarial data poisoning attacks (Jagielski et al. 2020). Gradient pruning is also used to limit the efficacy of model poisoning attacks (Bagdasaryan et al. 2020), where malicious clients use *constrain-and-scale* techniques to evade gradient anomaly detection algorithms such as Multi-KRUM. Table 1 shows the comparative analysis of the proposed BEAS framework against other existing frameworks.

Related Work

Federated learning (FL) (McMahan et al. 2017) has emerged as a promising approach to collaboratively train a model by exchanging model parameters with a central aggregator, instead of the actual training data. However, parameter exchange may still leak a significant amount of private data. To overcome this leakage problem, several approaches have been proposed based on differential privacy (Lyu et al. 2020), multi-party computation (Mondal et al. 2022; Xu et al. 2019; Bonawitz et al. 2019; Ryffel et al. 2018), fully homomorphic encryption (Truex et al. 2019), Trusted Execution Environment (Mondal et al. 2021b,a), etc. However, due to the extensive use of cryptographic operations, these protocols remain too slow for practical use. Furthermore, in those settings, the aggregator is a central player, which also potentially represents a single point of failure (Sav et al. 2020).

To overcome this single point of failure, Shae et al. (Shae and Tsai 2018) considers a theoretical implementation of distributed learning and transfer learning with blockchain smart contracts to design a distributed parallel computing architecture, and (Lu, Tang, and Wang 2018) proposes crowdsourcing of ML tasks on public blockchains where nodes are incentivized for contributing their computational resources. Similarly, (Harris and Waggoner 2019) uses blockchain to build a public dataset, and smart contracts are used to host a continuously updated model. However, their focus is not on sensitive datasets: data needs to be published on the blockchain (Lu, Tang, and Wang 2018) or is shared freely amongst nodes for efficient computation (Harris and Waggoner 2019).

A number of approaches (Lu, Tang, and Wang 2018; Yong, Lee, and Wang 2017; Bhattacharya et al. 2019; Idé

Table 1: Comparison between various privacy-preserving federated learning framework.

Framework	Comms†	Threat Model		Privacy Guarantees		Security Guarantees		Techniques Used							Features and Code Availability									
		Aggregator	Participants	Inference	Training	Data Poisoning	Model Poisoning	Byzantine Attack	HE	TP	SS+AE	FE	DP	Blockchain	Identity Privacy	Statistical Heterogeneity	Scalability	Asynchronous Updates	Dynamic Participants	Decentralized	Premature Convergence	Reward Mechanism	Open-Source
BinDaaS (Bhattacharya et al. 2019)	3 rounds	□	–	○	○	○	○	○	○	○	○	○	○	●	●	○	●	○	○	○	○	○	×
PiRATE (Zhou et al. 2020)	3 rounds	☒	–	◐	◐	●	○	○	○	○	○	○	○	●	●	●	○	○	○	○	○	●	×
BAFFLE (Ramanan and Nakayama 2020)	3 rounds	■	–	●	●	●	○	○	○	○	○	○	○	●	●	●	○	○	○	○	○	●	×
Li et al. (Li et al. 2020)	3 rounds	☒	–	○	○	●	○	○	○	○	○	○	○	●	●	○	●	○	○	○	○	●	×
LearningChain (Chen et al. 2018)	3 rounds	■	–	●	●	◐	◐	◐	○	○	○	○	●	●	●	●	○	○	○	○	○	●	×
Biscotti (Shayan et al. 2018)	3 rounds	■	–	●	●	●	◐	◐	○	○	○	○	●	●	●	●	●	○	○	○	●	●	×
Truex et al. (Truex et al. 2019)	3 rounds	■	☒	○	○	○	○	○	○	●	○	○	●	○	○	◐	○	○	○	○	○	○	×
Bonawitz et al. (Bonawitz et al. 2019)	3 rounds	■	☒	○	○	○	○	○	○	○	●	○	○	○	●	○	○	○	◐	○	○	○	×
PySyft (Ryffel et al. 2018)	2 rounds	□	☒	○	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	×
FLTrust (Cao et al. 2020)	2 rounds	■	☒	◐	●	●	●	●	○	○	○	○	○	○	●	○	●	○	○	○	○	○	×
POSEIDON (Sav et al. 2020)	2 rounds	■	☒	●	●	○	○	○	●	○	○	○	○	○	●	●	○	○	○	●	○	○	×
Shokri et al. (Shokri and Shmatikov 2015)	1 round	□	□	○	○	○	○	○	○	○	○	○	○	○	◐	●	○	○	◐	○	○	○	×
PATE (Papernot et al. 2018)	1 round	□	□	○	○	○	○	○	○	○	○	○	○	○	○	◐	○	○	○	○	○	○	×
HybridAlpha (Xu et al. 2019)	1 round	■	☒	●	◐	◐	○	○	○	○	○	●	●	○	●	●	○	○	●	○	○	○	×
BEAS(This Work)	1 round	■	–	●	●	●	●	●	○	○	○	○	●	●	●	●	●	●	●	●	●	●	✓

BEAS proposes efficient privacy-preserving federated learning framework in decentralized settings with stronger security and privacy guarantees. BEAS also provides an extensive evaluation and comparison (1) over the large number of datasets, (2) provides an extensively compares with related work, (3) provides newer insights for future directions and a number of interesting application.
"HE" is homomorphic encryption'; "TP" is Threshold-Paillier system; "SS+AE" secret sharing with key agreement protocol and authenticated encryption scheme; "FE" is functional encryption; and "DP" is Differential Privacy.
† includes the number of communication rounds required in one epoch at the training phase between the aggregator and the participant.
□ denotes honest party; ☒ denotes semi-honest party; ■ denotes dishonest party; ○ denotes does not provides property; ◐ denotes partially provides property; ● denotes provides property.

2018; Majeed and Hong 2019; Wang et al.; Kuo and Ohno-Machado 2018) try to eliminate the centralized dependence in a learning framework using blockchain, these are still subject to poisoning attacks by malicious nodes that train their local models using poisoned datasets (Jagielski et al. 2020) or tamper with their model gradients (Bagdasaryan et al. 2020). (Li et al. 2020) presents a committee consensus mechanism to validate gradients where a leader is selected based upon performance in the previous round. However, this can result in poor performance of an honest node when the distribution of the leader's dataset is more similar to that of the malicious node than the honest node. (Song and Mittal 2020) further demonstrates membership inference attacks that can reveal whether a specific data point was used to train a given model, and how the models themselves can unintentionally memorize training inputs.

The current state-of-the-art for decentralized FL frameworks (Shayan et al. 2018; Chen et al. 2018) tries to eliminate the poisoning attacks, inference attacks, and membership attacks by using the several cryptographic techniques. However, due to the extensive use of cryptographic schemes, these frameworks remain too slow for practical use. The main advantage BEAS has over these approaches is the communication efficiency, and stronger data privacy and security guarantees (see Table 1). The existing approaches need more than one round of communication in the local model aggregating phase, BEAS only incurs a single round. Hence BEAS can be used to train machine learning models faster as demonstrated in the experimental section.

Background and Preliminaries

Federated Learning

Federated learning (FL) (McMahan et al. 2017) is a distributed machine learning approach that enables collabora-

tive model training on decentralized data. The training setup comprises multiple clients with private datasets that want to collaboratively train a shared global model, but do not wish to expose their secret private dataset to other participating clients (Yang et al. 2019). To do so, every client trains a model locally with their own data, and exchanges only the model parameters with an FL *aggregator* instead of directly sharing the private training data. The aggregator merges the received local gradients (generally by averaging) to generate the new global model, which is then sent back to all the participants to use as a pre-training model for the next iteration of local training. The process repeats until desired performance is achieved, or *ad infinitum*.

Blockchain

Blockchain is a decentralized immutable ledger of records, called blocks, that are sequentially linked using cryptography and maintained across a network of presumably distrustful peers (Narayanan et al. 2016; Nakamoto 2009). Every peer maintains a copy of the ledger, eliminating the dependency on a centralized authority. Blockchains typically allow execution of programmable scripts. Crypto currencies such as Bitcoin use these scripts to validate transactions (Nakamoto 2009). HyperLedger Fabric (Androulaki et al. 2018) extends this capability with the concept of smart contracts that can function as a trusted decentralized application (DApp) (Androulaki et al. 2018). In the context of decentralized FL, smart contracts can be used for storage, exchange, and merging of model gradients, replacing the dependence on a centralized aggregator. We implement BEAS using HyperLedger Fabric (Androulaki et al. 2018), which is an open source enterprise-grade permissioned blockchain framework, for the following reasons: (a) permissioned blockchain architecture, (b) easily scalable, (c) multi-channel blockchain design, (d) transaction level consensus, (e) no crypto-currency requirement, and (f) no proof-of-work/stake.

HyperLedger Fabric (Androulaki et al. 2018) is an open source enterprise-grade permissioned blockchain framework. At a high level, it is comprised of the following modular components (Maheshwari 2018; Androulaki et al. 2018):

1. Anything that can have value, state, and ownership are called *Assets*, and are represented as a set of key-value pairs.
2. *Ledger* records the state and ownership of an asset at a given point in time, as well as a log history that records every transaction.
3. *Smart contracts (or chaincode)* is software that comprises the business logic of the framework; that is, it defines the assets and the applicable transaction functions.
4. *Endorsing Peers* are a fundamental component of the entire framework. They host and access the ledgers, endorse transactions, interface with applications, and execute smart contracts within a secure container environment (e.g. Docker) for isolation.
5. *Endorsement policy* defines the necessary and sufficient conditions for a valid transaction endorsement.
6. *Channels* are logical structures formed by a collection of participating peers, allowing a group of peers to create a separate ledger of transactions.
7. *Membership service provider (MSP)* is responsible for associating entities in the network with cryptographically generated identities.
8. *Ordering service* packages transactions into blocks, and establishes consensus on the order of transactions. It guarantees the delivery of transactions in the network using a peer-to-peer gossip protocol.

We select Fabric over other blockchain platforms for BEAS due to the following reasons (Sharon and Gari 2018; Androulaki et al. 2018):

Permissioned blockchain: Anyone can join public permission-less blockchains networks, including anonymous and pseudonymous users. Fabric's permissioned blockchain design requires all participants to have cryptographically generated anonymous identities, which is useful in our case as malicious parties can be removed from the network when their adversarial intent is detected.

Scalability: Most public blockchains require all the nodes on the network to process transactions, which results in low transaction throughput and impacts scalability. In Fabric, network designers can plug in the requisite nodes as per their immediate requirements, making the framework highly scalable.

Multi-Channel: The need for data partitioning on the blockchain is essential for most business use-cases due to competitiveness, protection laws, and regulation on confidentiality of personal data. In Fabric, Channels allow for data to go to only the parties that need to know. In our case, we use multi-channel blockchain to facilitate parallel collaboration with multiple models on the same platform by setting up each channel with its own ML task, and a separate blockchain ledger to store the shared model gradient blocks.

Transaction Consensus: While most blockchain frameworks require consensus on a ledger state before transaction approval, Fabric relies on a transaction level consensus; in other words, the entire block doesn't require validation before being approved, but only the transaction.

No Crypo-Currency required: Public blockchain frameworks like Ethereum (Wood 2014) require crypto-currencies to execute chaincode functionality, which is redundant in our use case. Fabric can be used without a cryptocurrency.

No Proof of work/stake: Fabric does not depend upon miners working day and night to solve problems for block validation, which requires lots of computing power and is energy-intensive.

Differential Privacy

Differential privacy (Dwork 2006) can be used to ensure that the presence (or absence) of any given element in a dataset does not result in the generation of vastly different model gradients (which could potentially be used to reveal private information from shared model gradients in FL). Differential privacy in the context of privacy-preserving FL, is defined in Definition 0.1.

Definition 0.1 (Differential Privacy (Dwork 2006)) *A randomized function K gives $\epsilon-$differential privacy if for all model gradients G_1 and G_2, generated by training on datasets D_1 and D_2 differing on at most one element, the probability of function K resulting in an output S on G_1 is close to the probability of function K resulting in the same output S on G_2 as follows:*

$$Pr[K(G_1) \in S] \leq exp(\epsilon) \times Pr[K(G_2) \in S]$$

where all $S \subseteq Range(K)$.

Existing work majorly relies on output perturbation, by adding noise or clipping the norm of the gradients, for differentially private sharing of model gradients (Lyu et al. 2020; Shayan et al. 2018; Abadi et al. 2016; Balle and Wang 2018):

Addition of Noise: When the query is a function f, and the database is X, the true answer is the value $f(X)$. The mechanism K adds appropriately chosen random noise to the true answer to produce what we call the response (Dwork 2006). Generally, a Gaussian or Laplace distribution is used for generating the noise for gradient perturbation.

Gradient Clipping: Gradient value clipping involves clipping the derivatives of the loss function to have a given value if a gradient value is less than a negative threshold, or more than the positive threshold (Jason Brownlee 2019).

Gradient Pruning: In gradient pruning, gradients with small magnitudes are pruned to zero. The sparsity in the gradients increases as more and more gradient values are pruned, making attacks on data privacy hard (Tsuzuku, Imachi, and Akiba 2018; Lin et al. 2020).

BEAS Overview

BEAS aims to achieve secure and efficient N-party machine learning while ensuring strict privacy guarantees that prevent leakage of sensitive information from shared model gradients, as well as ensure resiliency from adversaries. We consider a setting where N mutually distrustful clients $\mathcal{C}_1, \mathcal{C}_2 ... \mathcal{C}_N$, that hold private datasets $\mathcal{D}_1, \mathcal{D}_2 ... \mathcal{D}_N$ respectively, want to collectively train a shared *global* model $\mathcal{M}_\mathcal{G}$ without exposing their secret private dataset $\mathcal{D}_i$ to other participants. This need for collaboration arises because a model trained only on an individual's data would exhibit poor performance, but a model trained by all participants will have near-optimal performance (Shayan et al. 2018; Yang et al. 2019). We also assume that the NN training parameters are known to all clients: the model architecture, hyper-parameters, optimization algorithm, and the learning task of the system (these can be requested from the endorsing peers). We target horizontal learning, where every user contributes data containing the same feature space, *i.e.* data with same columns, but different/overlapping rows.

Threat Model

We assume *curious and colluding* clients, who may collude to try to acquire sensitive information from other clients' private training datasets by inspecting the model gradients that are publicly published on the shared ledger. Moreover, clients may be adversarial, and contribute poisoned updates to introduce backdoors into the shared model. However, colluding clients cannot gain influence without acquiring sufficient stake. The adversary may control more than one client, as in Sybil attacks (Douceur 2002), with the intent to take control of the blockchain consensus by acquiring a majority stake (by creating false identities). However, we assume that they can not control more than f clients out of N total clients, when $2f + 2 < N$. Moreover, although adversaries may be able to increase the number of clients under their control, they cannot artificially increase their stake in the system except by either providing valid updates that pass Multi-KRUM (Blanchard et al. 2017), or by modifying their gradient updates to evade anomaly detection as in a model poisoning attack (Bagdasaryan et al. 2020). We further assume that the intent of the adversary is to harm the performance, or introduce backdoors, into the shared *global* model, or leak private information about the used training dataset. For the purpose of this work, we limit the adversaries to label flipping attacks (Taheri et al. 2020), pixel-pattern backdoor attacks (Bagdasaryan et al. 2020), and deep leakage from gradients using reconstruction attacks (Zhu, Liu, and Han 2019).

BEAS's operating steps are describes in Scheme 1. Algorithm 1 describes the high level required steps and operations of our proposed BEAS framework for the decentralized federated machine learning training in N-party settings.

Scheme 1: BEAS **Framework**

Input: Client $\mathcal{C}_i$ for $i \in \{1, \cdots, K\}$ holds its private dataset $\mathcal{D}_i$.
Output: Client $\mathcal{C}_i$ for $i \in \{1, \cdots, K\}$ obtain the collaboratively learned *global* model, $\mathbb{M}_G$.

1. Clients create cryptographically anonymous identities using the MSP.
2. To begin training process, any client $\mathcal{C}_i \in \mathcal{C}_K$ can set up a new channel c, define the training parameters and the model architecture, and generate a genesis block $\mathcal{M}_g$ by training on their own private data D_i locally. $\mathcal{M}_g$ is uploaded as the first *global* block on the channel ledger $\mathcal{L}_c$.
3. Other clients connect to the EP to request the latest *global* block from the channel ledger. They use the requested block to initialize a pre-trained model, and update it by training on their own private datasets D_i to generate new *local* gradients.
4. Client $\mathcal{C}_i$ sends their *local* gradients to the EP, which creates a new *local* block and shares it with the Ordering service.
5. The ordering service establishes consensus on the ordering of blocks, and consequently commits them onto each EP's ledger using a peer-to-peer gossip protocol.
6. The EP checks if the number of queued *local* blocks is $\geq$ than the merge threshold. If true, the Merge chaincode is triggered to evaluate the quality of

each *local* block by calculating the anomaly detection scores, and aggregating blocks using federated averaging to generate a new *global* block, which is then sent to the channel ledger.

7. Steps 3 to 6 get repeated until desired accuracy for the shared *global* is achieved or *ad-infinitum*.

Algorithm 1: BEAS Training Algorithm

Input: BEAS with K Local Blocks generated by K clients, for P_k the set of indexes of data points on client k, and $n_k = \mid p_k \mid$. $f_i(w) = l(x_i, y_i; w)$ is the loss of the prediction on example (x_i, y_i) made with model parameters w. When $K > t$, where t is the merging threshold, the merge chaincode is triggered to generate the Global Block. $\mathcal{M}_{FL}$:= Machine learning algorithms to be trained; ϵ := differential privacy guarantee; $\mathcal{C}_{1 \to N}$:= Set of N participants, where $\mathcal{C}_i$ holds its own dataset $\mathcal{D}_i$; c := local training cluster size.

Output: Trained Global Model $\mathcal{M}_\mathcal{G}$

`ChannelInitialize` (t)**:**
- **for** *each training round* **do**
  - $K = 0$
  - **for** *each client* $\mathcal{C}_k$ *in parallel* **do**
    - $\mathcal{M}_L^k$ = LocalTraining$(\mathcal{M}_G, k, w)$
    - $K = K + 1$
    - **if** $K > t$ **then**
      - $\mathcal{M}_G \to$ ModelAggregate$(\mathcal{M}_L$, w$)$

`LocalTraining` $(\mathcal{M}_G, k, w)$**:**
- pretrain $\mathcal{M}_L$ using $\mathcal{M}_G$
- $\mathcal{M}_L^k \to \frac{1}{n_k} \sum_{i \in P_k} f_i(\mathcal{M}_L^k) + \epsilon$ `// run on client k`
- **return** $\mathcal{M}_L^k$

`ModelAggregate` $(\mathcal{M}_L, w)$**:**
- MKScore$(\mathcal{M}_L^i) = \sum_{i \to j} \mid\mid \mathcal{M}_L^i - \mathcal{M}_L^j \mid\mid^2$
- **for** *top* m *blocks by score* **do**
  - $\mathcal{M}_G \to \sum_{k=1}^{K} \frac{n_k}{n} F_k(\mathcal{M}_L^i)$
- **return** $\mathcal{M}_G$

System Architecture

In traditional FL, each client $\mathcal{C}_i$ trains a *local* model ${\mathcal{M}_\mathcal{L}}^i$ with their own private data, and shares only the generated gradients with a centralized *aggregator* that is responsible for storage and exchange of model gradients, and periodically merges previous *local* model gradients to generate the new *global* model $\mathcal{M}_\mathcal{G}$. This is a vulnerable single point of failure as malicious aggregators can bias contributions of preferred clients over others (Li et al. 2020) to skew the shared *global* model. In a decentralized setting such as ours, we replace the role of the centralized aggregator with blockchain smart contracts (Androulaki et al. 2018) for storage, exchange, and merging of model gradients. The framework architecture is depicted in Figure 1 and is implemented using the Fabric Blockchain (Androulaki et al. 2018; fab 2020).

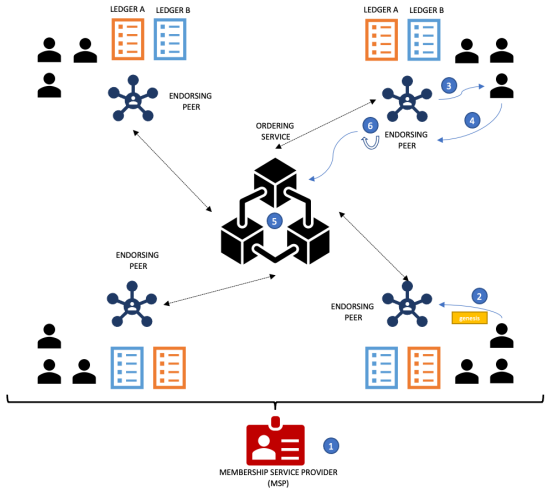

Figure 1: Platform Architecture for BEAS implemented using HyperLedger Fabric Blockchain (Androulaki et al. 2018; fab 2020). Steps annotated as detailed in Scheme 1.

The framework is designed to be scalable to incorporate large numbers of participating clients, as well as allow parallel FL models on the same platform with minimum overheads. In our experiments, we assume that the participating nodes are organized in a tree-network topology with every endorsing peer (EP) connected to multiple clients for communication efficiency. However, our framework is completely decentralized and network topology agnostic; it does not necessitate any topological node hierarchy, and each EP can be set up to work with a single node if required. The learning models are built using general-purpose ML APIs; hence BEAS can support any model architecture that can be optimized using SGD.

Clients generate cryptographically anonymous identities using the HyperLedger Fabric Membership Service provider (MSP) (fab 2020). Multi-channel blockchain design is used to facilitate collaboration on multiple tasks simultaneously. Every channel is set up with its own ML task, and a separate ledger. The ledger comprises a blockchain that stores model gradients as immutable and sequenced records in blocks, and a world-state database that stores the current state (Androulaki et al. 2018; fab 2020). We refer the reader to the Appendix for figures depicting the multi-channel blockchain design, as well as the block designs.

To begin the training process, any client can set up a new channel, define the training parameters and the model architecture, and generate a genesis block by training on their own private data locally. The generated genesis block is appended to the channel's ledger as the first global block. Now, other clients can connect to this channel to use the global model as per their own requirements, or collaborate to im-

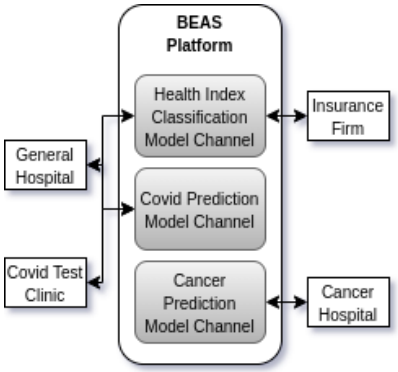

Figure 2: BEAS uses multi-channel blockchain to enable parallel FL task execution.

GLOBAL BLOCK
1. Time Stamp
2. Block Id
3. Previous Global Id
4. Gradient File Location
5. Training Data Size
6. BlockType = "G"
7. Gradient File Hash

LOCAL BLOCK
1. Time Stamp
2. Block Id
3. Previous Global Id
4. Gradient File Location
5. Training Data Size
6. BlockType = "L"
7. Gradient File Hash

Figure 3: BEAS blockchain block design.

prove the existing model with FL. In subsequent training rounds, clients request the Endorsing Peers (EP) that store a real-time copy of the ledger for the previous *global* block on the ledger, and use this as a pre-training model. They train using their own private data to generate updated gradients, and then send them back to the EP to create a new *local* block by executing smart contracts in an isolated environment. All the generated *local* blocks at EPs are then sent to an Ordering Service, which establishes consensus on the order of the created blocks before sending them back to EP for commitment to the ledger.

When a threshold t of *local* blocks is achieved on the ledger, a gradient merging smart contract is triggered to merge all the received *local* blocks using federated gradient averaging (McMahan et al. 2017) to generate a new *global* block. Since different clients have varying sizes of datasets that are not independent and identically distributed (Non-i.i.d), the merge contract weighs the contribution of each *local* block upon the size of its training dataset.

We also limit the amount of data that can be used by any client to train in a given *local* round to small chunks, and restrict *local* training to a low number of epochs. That is, clients divide their datasets into small subsets that are used for training in subsequent rounds, instead of training on the entire dataset in a single round of *local* training. This is done to ensure that the *local* models do not pre-maturely converge while training, as that can lead to forgetting of features learned in previous rounds.

Adversarial Poisoning

Malicious clients can tamper with their *local* training datasets to perform data poisoning attacks (Jagielski et al. 2020; Bagdasaryan et al. 2020) that can introduce backdoors into the shared *global* model. In fact, a single poisoned gradient can deviate the merged gradient significantly. Centralized FL frameworks majorly rely on server-owned validation datasets to evaluate *local* gradient performance before aggregating them into the *global* model, rejecting updates that perform poorly (Xie, Koyejo, and Gupta 2019). However, in a privacy-sensitive decentralized setting such as ours, detecting anomalous updates without using a publicly available testing dataset is required.

We use the *FoolsGold* (Fung, Yoon, and Beschastnikh 2020) (FG) defense protocol to address data poisoning attacks. In FL, training data is assumed to be non-IID. FG relies on similarity between client updates to distinguish honest participants from act-alike sybils by comparing the similarity of their gradient updates: sybils will contribute updates that are closer to each other than those among honest clients. It adapts the learning rate per client based on the similarity in indicative features amongst clients in any given round. As the framework faces larger groups of adversaries, it has more information to more reliably detect similarity between sybils. Unlike other defenses, FG does not require knowledge of the number of sybil attackers, and does not require modifications to the client-side protocol.

Unfortunately, as FG works by detecting similarity in gradients amongst Sybils, it fails to detect solitary attackers. For this case, BEAS complements FG with Multi-KRUM (MK) (Blanchard et al. 2017). Multi-KRUM (Blanchard et al. 2017) is a byzantine-resilient gradient aggregation algorithm that can address data poisoning attacks. In every federated round, it scores each *local* block based on its deviation from every other submitted blocks. It guarantees resiliency from f malicious updates out of n total updates, when $2f + 2 < n$. For update $V_i \; \forall \; i \in [1, n]$, $Score(V_i)$ is calculated as the sum of euclidean distances between V_i and V_j, where V_j denotes the $n - f - 2$ closest vectors to V_i as: $Score(V_i) = \sum_{i \to j} \mid\mid V_i - V_j \mid\mid^2$. Here, $i \to j$ denotes the fact that V_j belongs to $n - f - 2$ closest vectors to V_i. The $n - f$ updates with the lowest scores are selected for aggregation, and the rest are discarded. We refer the reader to (Blanchard et al. 2017) for security guarantees and convergence analysis of Multi-KRUM, and to (Fung, Yoon, and Beschastnikh 2020) for detailed description of the FoolsGold protocol.

To evade anomaly detection algorithms, malicious clients can also manipulate the training algorithm, their model parameters, or directly the model gradients, to generate poisoned model gradients. Bagdasaryan et al. (Bagdasaryan et al. 2020) demonstrates a backdoor model poisoning attack that uses *constrain-and-scale* techniques to generate gradients that do not look anomalous, but can introduce backdoors in the *global* model after merging. The generated model can achieve high accuracy on both – the main task, and an attacker-chosen backdoor subtask. Bagdasaryan et al. (Bagdasaryan et al. 2020) demonstrates that DP based

techniques with low clipping bounds and high noise variance can render the backdoor attack ineffective, but significantly decrease the accuracy of the *global* model on its main task. To minimize the efficacy of model poisoning attacks with DP, but without impacting model performance, we experiment with gradient pruning where gradients with small magnitudes are pruned to zero (Tsuzuku, Imachi, and Akiba 2018; Lin et al. 2020). Our experimental results show that as more and more gradient values are pruned, the sparsity in the model gradients increases: this has minimal impact on the main task, but minimizes the accuracy on the backdoor subtask under a model poisoning attack (Blanchard et al. 2017) due to gradient obfuscation, as it makes *constrain-and-scale* hard. We conclude that gradient pruning is an effective defense against model poisoning attacks on the backdoor subtask, and include pruning in the *local* training rounds for BEAS.

Privacy Leakage

In FL, client data is not directly shared with a centralized aggregator; to prevent leakage of data privacy, only the gradients are shared. However, (Song and Mittal 2020) demonstrates that a malicious aggregator can reveal sensitive information even from shared gradients that inadvertently retain features from the data used to train them. (Zhu, Liu, and Han 2019) further proposes a Deep Leakage from Gradient (DLG) attack where an adversary can reconstruct the training dataset and labels from gradients by first generating 'dummy' inputs and labels, and then performing the usual forward and backward pass using them to generate dummy gradients. Now, instead of optimizing model weights as in a typical learning setting, the adversary optimizes the dummy inputs and labels by minimizing the deviation between the dummy and real gradients, with the goal to make the dummy data close to the original ones, thereby revealing private information (Zhu, Liu, and Han 2019).

Prior work proposes the use of cryptographic techniques such as secure multi-party computation (MPC) (Perry et al. 2014), Homomorphic Encryption (Rivest, Adleman, and Dertouzos 1978), Functional Encryption (Xu and Huang 2020), and Secret Sharing (Shamir 1979) for the secure exchange of model gradients. However, cryptographic techniques involve significant computational overheads, and perform poorly in scale (Phong et al. 2018). Moreover, since cryptographic techniques encrypt shared gradients to prevent leakage of privacy, the gradients cannot be evaluated using anomaly detection algorithms. This renders the framework vulnerable to poisoning attacks. Similarly, DP techniques such as the addition of DP-noise and clipping of gradient values to limit privacy leakage by obfuscating model gradients have been explored in prior work, but impact accuracy (Lyu et al. 2020; Abadi et al. 2016; Dwork 2006).

Our experimental results demonstrate that gradient pruning is more effective than other DP-based techniques to preserve user data privacy under FL while ensuring minimal impact on model performance, but has not been used in recent work. To evaluate the extent of possible leakage of privacy, we implement the DLG Attack (Zhu, Liu, and Han 2019) with three different techniques – the addition of noise, gradient value clipping, and gradient pruning. We observe that gradient pruning helps minimize reconstruction of sensitive information of the training dataset from shared gradients. Moreover, pruning has minimal impact on model performance compared to existing noise and clipping based DP techniques.

Experimental Evaluation

In this section, we experimentally evaluate BEAS's performance and present our empirical results, followed by a security and privacy analysis of the framework.

Implementation Details

We implement the BEAS framework using the HyperLedger Fabric *v2.2* Blockchain (Androulaki et al. 2018; fab 2020), with the smart-contracts and client applications programmed using Node.js. All the experiments are performed using a Linux server with Intel Xeon E3-1200 v5 CPUs and GeForce RTX 2070 GPU, with 32 GB RAM. All client nodes are executed in isolated docker containers, and are connected using a 2Gbps bandwidth virtual network with an average network delay of 0.2ms.

Experimental Setup

Unless otherwise stated, we set up our test blockchain network with 5 endorsing peers (EP), each of which is connected to participating clients in two settings: (a) 4 clients per EP ($N = 4 \times 5 = 20$ Clients), and (b) 10 clients per EP ($N = 10 \times 5 = 50$ Clients), as our default network setting. In every training round, clients train a *local* model using a subset of their data (of cluster size c), and send the generated gradients to the EP as *local* blocks. When a threshold t of *local* blocks is achieved on the ledger, the merge chaincode is triggered to generate the new *global* block by aggregating the previous *local* blocks using federated averaging (McMahan et al. 2017). The merging threshold parameter t is kept low to ensure that the number of global training rounds increases, which has better impact on model performance than merging more gradient contributions per round as per our experiments. The value of cluster size c is set in a way that ensures at least 50 global training rounds can be achieved with the resultant dataset split. The cluster size c and the threshold t are defined in the following subsections based on the dataset used.

Datasets and Model Configurations

We report our evaluation of the BEAS framework using 3 popular and standard datasets with standard model configuration as follows:

MNIST (LeCun and Cortes 2010). Consists of 60,000 training images resized to (28, 28, 1), along with 10,000 testing images. Model used is a feed-forward CNN model. Following (McMahan et al. 2017; Bagdasaryan et al. 2020), we randomly split the dataset amongst the clients with an unbalanced sample from each class using a Dirichlet distribution (Minka 2003) with hyperparameter 0.9 to simulate *non-i.i.d.* distribution. Merging threshold t is set as 5, and *local* training is done for 5 epochs with the learning rate of

0.1 using a randomly selected cluster of size $c = 250$ from the clients data subset, with standard batch size 32.

Malaria Cell Image (Rajaraman et al. 2018). Consists of 27,558 images, resized to (50, 50, 3). We set aside the 20% of the dataset for testing. Model used is a feed-forward CNN model. We split the dataset as done for MNIST above. Merging threshold t is set as 5, and *local* training is done for 3 epochs with the learning rate of 0.1 using a randomly selected cluster of size $c = 100$ from the clients data subset, with standard batch size 32.

CIFAR-10 (Krizhevsky 2009). Consists of 50,000 images, resized to (32, 32, 3), along with 10,000 testing images. Model used is a feed-forward CNN model. To obtain the accuracy results, we execute a simulated BEAS (using Tensorflow) with approximated activation functions and a fixed-precision. We split the dataset as done for MNIST above. Merging threshold t is set as 5, and *local* training is done for 5 epochs with learning rate of 0.1 using a randomly selected cluster of size $c = 150$ from the clients data subset, with standard batch size 32.

Experimental Results

We evaluate BEAS in terms of (a) the accuracy of the resultant *global* model, and (b) the scalability of the framework. Tables 2 show the results obtained in our two settings with $N = 20$ and $N = 50$ respectively. The accuracy column compares the accuracy obtained using BEAS with a centralized ML approach, where the training dataset is available in a centralized setting. The execution time per local round, as well as the overall execution time are also mentioned.

Security and Privacy Analysis

We analyze the security and privacy of our proposed framework from (a) reconstruction attacks that are threat to client data privacy, and (b) two types of adversarial attacks – data poisoning attack (Jagielski et al. 2020) and model poisoning attack (Bagdasaryan et al. 2020).

Privacy Guarantees

We experiment with the Deep Leakage from Gradient Attack (DLG Attack) proposed by (Zhu, Liu, and Han 2019) to understand the extent of possible leakage from shared gradients. We also examine the impact of Gaussian noise addition, gradient value clipping, and gradient pruning to minimize this leakage. To simulate a DLG Attack (Zhu, Liu, and Han 2019), we begin by training an ML model to generate the gradients G_{real} using our dataset D_{real}. Then, we generate a pair of dummy inputs and labels D_{dummy} using random noise, and train the same ML model to get dummy gradients G_{dummy}. Now, instead of optimizing G_{dummy}, we optimize D_{dummy} using a loss function $|| \ D_{real} - D_{dummy} \ ||^2$ that minimizes the distance between the real gradients and the dummy gradients. Matching the gradients makes the dummy data close to the original ones. Our experiments show that as the optimization completes, the private training data can be completely revealed (We refer the reader to the Appendix for the reconstruction experiment figures).

We experiment with the following standard DP based schemes (Zhu, Liu, and Han 2019) to reduce adversarial reconstruction ability of sensitive information from shared model gradients on BEAS: (a) Gaussian Noise Addition: The mechanism adds appropriately chosen random noise to the generated gradients to produce obfuscated gradients. However, our experiments show that although it successfully prevents reconstruction at high standard deviation values, it negatively impacts the model performance. (b) Gradient Value Clipping: This forces element-wise gradient values to a specific minimum (or maximum) value if the gradient exceeded an expected range (Jason Brownlee 2019). However, our experiments show that it does not minimize leakage of private data from shared model gradients as the ground truth image can easily be reconstructed. (c) Gradient Pruning: Involves pruning of gradients with small magnitudes to zero to increase the sparsity between gradients values. In effect, gradient values with insignificant magnitudes are pruned, but gradient values with higher magnitude contributions are preserved. Our experiments show that as we increase gradient sparsity to more than 50%, the recovered images in a DLG attack (Zhu, Liu, and Han 2019) are no longer recognizable, successfully preventing leakage of sensitive information from shared gradients.

Table 3 shows the impact of different parameter settings for these DP techniques on the global model accuracy for BEAS. We refer the reader to the Appendix for the accuracy graphs. We observe that gradient pruning has minimal impact on model performance compared to other DP based techniques, but successfully minimizes reconstruction of training inputs from shared model gradients under the DLG Attack. This is consistent with the results obtained by (Zhu, Liu, and Han 2019), where the impact of DP techniques on privacy leakage is evaluated. We refer the reader to the Appendix for the reconstruction experiment figures.

Adversarial Security

Malicious users can train their local models using manipulated "poisoned" datasets to induce a specific outcome in the ML task at inference time (Jagielski et al. 2020). To simulate a data poisoning, we execute a label flipping attack (Nguyen et al. 2021; Tolpegin et al. 2020; Taheri et al. 2020) on the Malaria Cell Image dataset (Rajaraman et al. 2018). Label flipping attacks are a special type of data poisoning attacks, wherein the attacker can manipulate the class labels assigned to a fraction of training inputs. Label flipping attacks can significantly diminish the performance of the system, even if the attacker's capabilities are otherwise limited (Taheri et al. 2020). We experiment with 1, 5, and 10 adversarial participants in our attack setup, who modify their training datasets by flipping the class label from parasitized to uninfected, and vice versa. The aim is to make the *global* model more likely to classify the label in the learning task incorrectly (Nguyen et al. 2021; Tolpegin et al. 2020). To detect and reject malicious updates, we implement FoolsGold(Fung, Yoon, and Beschastnikh 2020) and Multi-KRUM (Blanchard et al. 2017). Table 4 shows the impact of label flipping attack with different number of adversarial participants. (Note that with Multi-KRUM, the accuracy de-

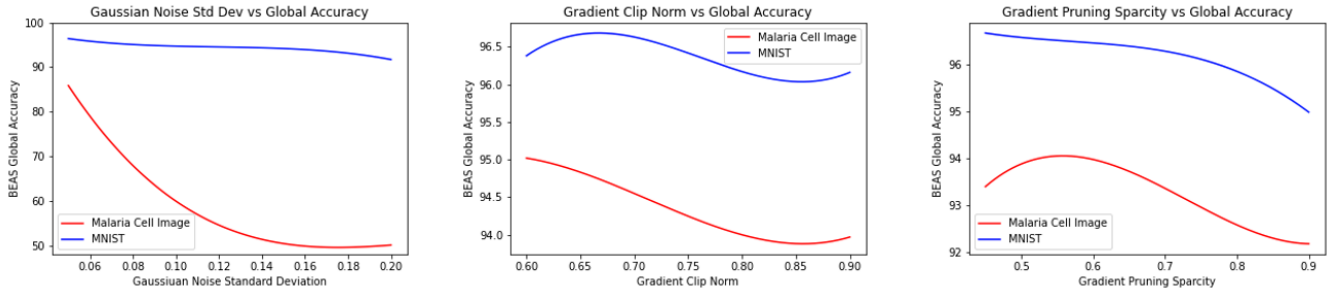

Figure 4: Impact of differential privacy parameter settings on BEAS global model accuracy; Dataset: Malaria Cell Image (Rajaraman et al. 2018), and MNIST (LeCun and Cortes 2010).

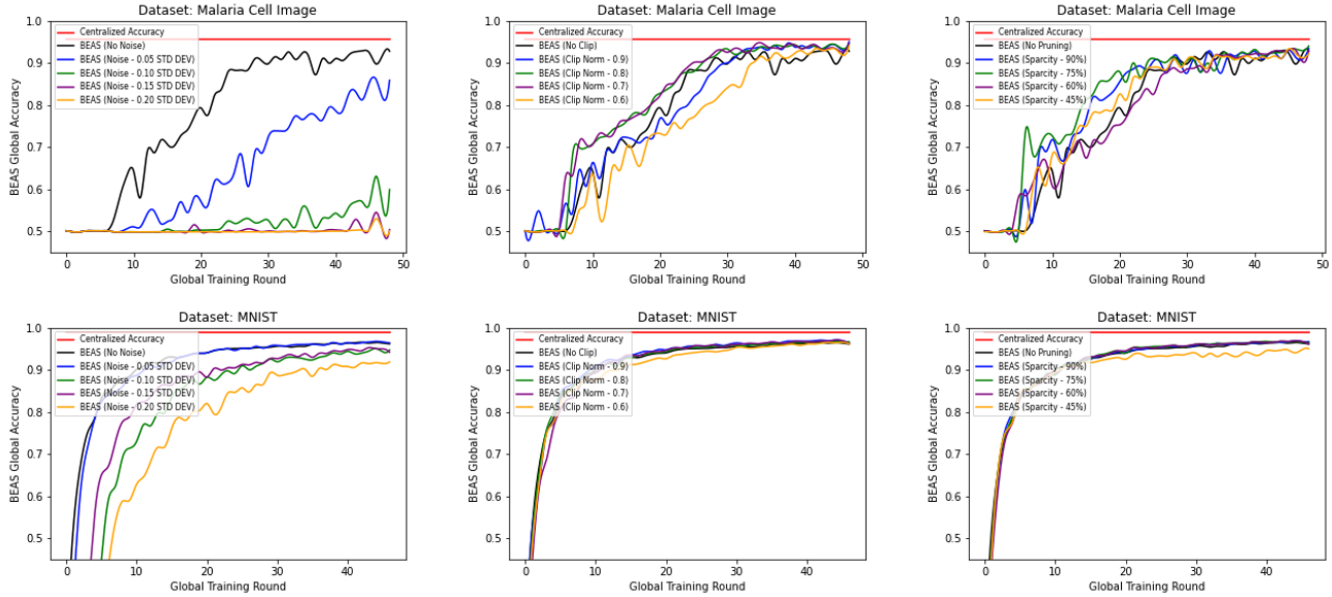

Figure 5: Impact of different differential privacy techniques on BEAS global model accuracy; Dataset: Malaria Cell Image (Rajaraman et al. 2018), and MNIST (LeCun and Cortes 2010).

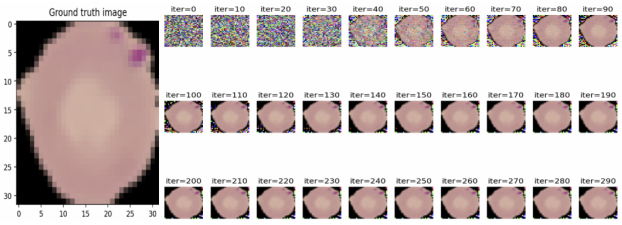

Figure 6: Private data reconstruction using Deep Leakage from Gradient Attack (Zhu, Liu, and Han 2019); as the optimization completes, D_{dummy} reveals the ground truth training image. Dataset: Malaria Cell Image (Rajaraman et al. 2018)

creases even without any adversarial clients. This is because the number of training samples also decreases with high rejection of blocks.)

To evade anomaly detection algorithms like Multi-KRUM (Blanchard et al. 2017), malicious clients can introduce backdoors into the joint model by manipulating the training algorithms, the model parameters, or the model gradients, to generate poisoned model gradients. (Bagdasaryan et al. 2020) proposes a backdoor model poisoning attack that uses *constrain-and-scale* techniques to generate poisoned model gradients that do not look anomalous, and replace the *global* model after aggregating with the other participants' models. The generated model can achieve high accuracy on both the main task, and an attacker-chosen backdoor subtask. However, the author also shows that DP based techniques with low clipping bounds and high noise variance can render the backdoor attack ineffective. However, our experimental results show that existing DP techniques such as addition of noise and clipping the gradient norms significantly impact the accuracy of the *global* model on its main task.

Since BEAS uses gradient pruning based DP, we compare

Table 2: BEAS's accuracy and execution times for $N = 20$ and $N = 50$ clients.

Dataset	Centralized Accuracy (%)	$N = 20$			$N = 50$		
		Accuracy (%)	Avg. Execution Time (s)		Accuracy (%)	Avg. Execution Time (s)	
			Local Training	Overall		Local Training	Overall
MNIST	95.53	92.74	1.89	524	90.11	1.89	726
Malaria	98.89	96.16	2.18	967	92.81	2.18	1276
CIFAR-10	72.81	61.03	38	21608	63.76	38	25966

Table 3: Impact of different differential privacy parameter settings on BEAS global model accuracy

Framework	Parameters	Accuracy (%)	
		MNIST	Malaria
Centralized	-	95.53	98.89
BEAS	-	92.74	96.16
BEAS + Gaussian Noise Addition (Std Dev)	Noise (0.10)	59.92	94.7
	Noise (0.15)	50.41	94.16
	Noise (0.20)	50.07	91.68
BEAS + Gradient Clipping (Clip Norm)	Clip (0.90)	95.02	96.38
	Clip (0.80)	94.55	96.63
	Clip (0.70)	94.7	96.17
BEAS + Gradient Pruning (Sparcity)	Pruning (0.90)	93.39	96.67
	Pruning (0.75)	93.97	96.46
	Pruning (0.60)	92.95	96.11

Table 4: BEAS accuracy with FoolsGold (FG) (Fung, Yoon, and Beschastnikh 2020) and Multi-KRUM (MK) (Blanchard et al. 2017) under Label Flipping attack (Taheri et al. 2020) for different number of adversaries and $(N = 20)$; Dataset: Malaria Cell Image (Rajaraman et al. 2018).

Defense	Number of Adversaries			
	0	1	5	10
NIL	96.16	96.02	82.88	57.20
MK	94.22	94.60	91.17	72.11
FG	95.63	82.11	87.50	85.72
MK + FG	94.16	90.26	87.24	83.66

Table 5: BEAS accuracy on main task and backdoor subtask with different differential privacy techniques under Pixel Pattern Backdoor Model Poisoning attack (Bagdasaryan et al. 2020) for different number of adversaries and $(N = 20)$; Dataset: Malaria Cell Image (Rajaraman et al. 2018).

Framework	Main Task Accuracy (%)			Backdoor Task Accuracy (%)		
Adversaries per Round	0	1	5	0	1	5
BEAS	96.16	95.81	96.08	11.06	28.20	61.85
BEAS + Noise (0.05)	85.84	84.66	82.10	09.76	19.44	49.16
BEAS + Clipping (0.80)	94.55	94.16	93.60	11.33	27.21	62.70
BEAS + Pruning (0.60)	92.95	92.67	92.88	10.20	22.46	43.88

its efficacy in minimizing the impact of a model poisoning attack w.r.t other differential privacy techniques. To simulate a model poisoning attack, we implement a pixel pattern model poisoning attack (Bagdasaryan et al. 2020) using the Malaria Cell Image Dataset (Rajaraman et al. 2018). Here, the attacker modifies a subset of the training image's pixels in order for the model to incorrectly classify the modified image on a backdoor task, and then uses *constrain-and-scale* techniques to evade gradient anomaly detection algorithms such as Multi-KRUM (Blanchard et al. 2017).

Table 5 shows the performance of the generated *global* model on the main task for 0, 1, and 5 adversaries per global iteration, along with the performance of the generated model on the backdoor subtask. We see that increasing the number of adversaries has minimal impact on the accuracy on the main task, but increases the accuracy of the backdoor subtask. We also note that Gradient Pruning minimizes the accuracy on the backdoor subtask, with minimal impact on the performance of the global model. We conclude that gradient pruning is the most effective defense against model poisoning attacks on the backdoor subtask.

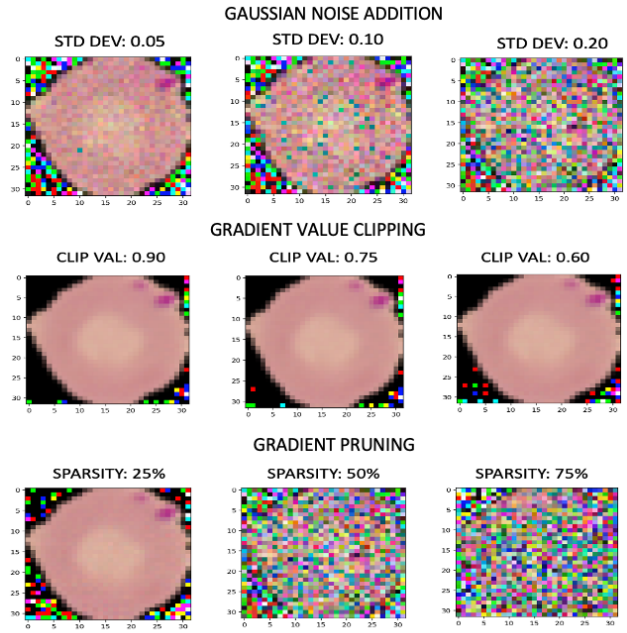

Figure 7: Differential Privacy Techniques against Deep Leakage from Gradient Attack (Zhu, Liu, and Han 2019) to reconstruct training data from shared model gradients; Reconstruction results after 300 iterations.

Conclusion

From adversarial poisoning attacks to client data privacy leakage, traditional FL setups face several challenges. We propose BEAS, a blockchain based framework which overcomes these challenges, and conduct extensive experiments with multiple datasets and observe promising results. BEAS achieves approximately 92.72% accuracy on the MNIST (LeCun and Cortes 2010) dataset, and 96.16% accuracy on the Malaria Cell Image (Rajaraman et al. 2018) dataset while training on DNNs. We also scale BEAS with a large number of participants (20 to 50), where it achieves a reasonable accuracy of 90.11% on MNIST. To ensure detection and rejection of malicious nodes that try to poison the global model with tampered datasets, we implement anomalous gradient detection (Blanchard et al. 2017; Fung, Yoon, and Beschastnikh 2020). We also implement gradient pruning to maximize privacy protection without sacrificing model utility loss, which is the case when using traditional DP techniques. Gradient pruning also helps us reduce the efficacy of model poisoning attacks (Bagdasaryan et al. 2020).

In the future, we wish to conduct tests using synthetic data for effective privacy preservation, and ablation studies where we observe how removal (or addition) of specific features in the model affect performance in a decentralized FL setting. We also plan to deploy our framework via open-source channels for different academic and industrial purposes to observe its working real-time.

References

2020. HyperLedger Fabric: A Blockchain Platform for the Enterprise. https://bit.ly/3tdkc4f. Accessed: 2021-01-21.

Abadi, M.; Chu, A.; Goodfellow, I.; McMahan, H. B.; Mironov, I.; Talwar, K.; and Zhang, L. 2016. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, 308–318. New York, NY, USA: Association for Computing Machinery. ISBN 9781450341394.

Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; and et al. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18. New York, NY, USA: Association for Computing Machinery. ISBN 9781450355841.

Bagdasaryan, E.; Veit, A.; Hua, Y.; Estrin, D.; and Shmatikov, V. 2020. How To Backdoor Federated Learning. In Chiappa, S.; and Calandra, R., eds., *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, 2938–2948. PMLR.

Balle, B.; and Wang, Y.-X. 2018. Improving the Gaussian Mechanism for Differential Privacy: Analytical Calibration and Optimal Denoising. In Dy, J.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 394–403. Stockholmsmässan, Stockholm Sweden: PMLR.

Bhattacharya, P.; Tanwar, S.; Bodke, U.; Tyagi, S.; and Kumar, N. 2019. BinDaaS: Blockchain-Based Deep-Learning as-a-Service in Healthcare 4.0 Applications. *IEEE Transactions on Network Science and Engineering*, 1–1.

Blanchard, P.; El Mhamdi, E. M.; Guerraoui, R.; and Stainer, J. 2017. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30, 119–129. Curran Associates, Inc.

Bonawitz, K.; Eichner, H.; Grieskamp, W.; Huba, D.; Ingerman, A.; Ivanov, V.; Kiddon, C.; Konečný, J.; Mazzocchi, S.; McMahan, H. B.; Overveldt, T. V.; Petrou, D.; Ramage, D.; and Roselander, J. 2019. Towards Federated Learning at Scale: System Design. arXiv:1902.01046.

Cao, X.; Fang, M.; Liu, J.; and Gong, N. Z. 2020. FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. *arXiv preprint arXiv:2012.13995*.

Chen, X.; Ji, J.; Luo, C.; Liao, W.; and Li, P. 2018. When Machine Learning Meets Blockchain: A Decentralized, Privacy-preserving and Secure Design. In *2018 IEEE International Conference on Big Data (Big Data)*, 1178–1187.

Di Crescenzo, G.; Feigenbaum, J.; Gupta, D.; Panagos, E.; Perry, J.; and Wright, R. N. 2014. Practical and privacy-preserving policy compliance for outsourced data. In *International Conference on Financial Cryptography and Data Security*, 181–194. Springer.

Douceur, J. R. 2002. The Sybil Attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, 251–260. Berlin, Heidelberg: Springer-Verlag. ISBN 3540441794.

Dwork, C. 2006. Differential Privacy. In Bugliesi, M.; Preneel, B.; Sassone, V.; and Wegener, I., eds., *Automata, Languages and Programming*, 1–12. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-35908-1.

Fung, C.; Yoon, C. J. M.; and Beschastnikh, I. 2020. Mitigating Sybils in Federated Learning Poisoning.

Gupta, D.; Mood, B.; Feigenbaum, J.; Butler, K.; and Traynor, P. 2016. Using intel software guard extensions for efficient two-party secure function evaluation. In *International Conference on Financial Cryptography and Data Security*, 302–318. Springer.

Harris, J. D.; and Waggoner, B. 2019. Decentralized and Collaborative AI on Blockchain. In *2019 IEEE International Conference on Blockchain (Blockchain)*, 368–375.

Idé, T. 2018. Collaborative Anomaly Detection on Blockchain from Noisy Sensor Data. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, 120–127.

Jagielski, M.; Severi, G.; Harger, N. P.; and Oprea, A. 2020. Subpopulation Data Poisoning Attacks. arXiv:2006.14026.

Jason Brownlee. 2019. How to Avoid Exploding Gradients With Gradient Clipping. https://bit.ly/3ovSLyZ. Accessed: 2021-01-20.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Technical report.

Kuo, T.-T.; and Ohno-Machado, L. 2018. ModelChain: Decentralized Privacy-Preserving Healthcare Predictive Modeling Framework on Private Blockchain Networks. *ArXiv*, abs/1802.01746.

LeCun, Y.; and Cortes, C. 2010. MNIST handwritten digit database.

Li, Y.; Chen, C.; Liu, N.; Huang, H.; Zheng, Z.; and Yan, Q. 2020. A Blockchain-Based Decentralized Federated Learning Framework with Committee Consensus. *IEEE Network*, 1–8.

Lin, Y.; Han, S.; Mao, H.; Wang, Y.; and Dally, W. J. 2020. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. arXiv:1712.01887.

Lu, Y.; Tang, Q.; and Wang, G. 2018. On Enabling Machine Learning Tasks atop Public Blockchains: A Crowdsourcing Approach. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, 81–88.

Lyu, L.; Li, Y.; Nandakumar, K.; Yu, J.; and Ma, X. 2020. How to Democratise and Protect AI: Fair and Differentially Private Decentralised Deep Learning. *IEEE Transactions on Dependable and Secure Computing*, 1–1.

Maheshwari, S. 2018. Blockchain basics: Hyperledger Fabric. https://developer.ibm.com/technologies/blockchain/articles/blockchain-basics-hyperledger-fabric/. Accessed: 2021-05-01.

Majeed, U.; and Hong, C. 2019. FLchain: Federated Learning via MEC-enabled Blockchain Network.

McMahan, H.; Moore, E.; Ramage, D.; Hampson, S.; and y Arcas, B. A. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*.

Minka, T. 2003. Estimating a Dirichlet distribution.

Mondal, A.; More, Y.; Ramachandran, P.; Panda, P.; Virk, H.; and Gupta, D. 2022. Scotch: An Efficient Secure Computation Framework for Secure Aggregation. *arXiv preprint arXiv:2201.07730*.

Mondal, A.; More, Y.; Rooparaghunath, R. H.; and Gupta, D. 2021a. Flatee: Federated Learning Across Trusted Execution Environments. *arXiv preprint arXiv:2111.06867*.

Mondal, A.; More, Y.; Rooparaghunath, R. H.; and Gupta, D. 2021b. Poster: FLATEE: Federated Learning Across Trusted Execution Environments. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, 707–709. IEEE.

Mood, B.; Gupta, D.; Carter, H.; Butler, K.; and Traynor, P. 2016. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, 112–127. IEEE.

Nakamoto, S. 2009. Bitcoin: A peer-to-peer electronic cash system.

Narayanan, A.; Bonneau, J.; Felten, E.; Miller, A.; and Goldfeder, S. 2016. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. USA: Princeton University Press. ISBN 0691171696.

Nguyen, T. D.; Rieger, P.; Yalame, H.; Möllering, H.; Fereidooni, H.; Marchal, S.; Miettinen, M.; Mirhoseini, A.; Sadeghi, A.-R.; Schneider, T.; and Zeitouni, S. 2021. FLGUARD: Secure and Private Federated Learning. arXiv:2101.02281.

Papernot, N.; Song, S.; Mironov, I.; Raghunathan, A.; Talwar, K.; and Erlingsson, Ú. 2018. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908*.

Perry, J.; Gupta, D.; Feigenbaum, J.; and Wright, R. N. 2014. Systematizing secure computation for research and decision support. In *International Conference on Security and Cryptography for Networks*, 380–397. Springer.

Phong, L. T.; Aono, Y.; Hayashi, T.; Wang, L.; and Moriai, S. 2018. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security*, 13(5): 1333–1345.

Rajaraman, S.; Antani, S. K.; Poostchi, M.; Silamut, K.; Hossain, M. A.; Maude, R. J.; Jaeger, S.; and Thoma, G. R. 2018. Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images. *PeerJ*, 6: e4568.

Ramachandran, P.; Agarwal, S.; Mondal, A.; Shah, A.; and Gupta, D. 2021. S++: A Fast and Deployable Secure-Computation Framework for Privacy-Preserving Neural Network Training. *arXiv preprint arXiv:2101.12078*.

Ramanan, P.; and Nakayama, K. 2020. BAFFLE : Blockchain Based Aggregator Free Federated Learning. In *2020 IEEE International Conference on Blockchain (Blockchain)*, 72–81.

Rivest, R. L.; Adleman, L.; and Dertouzos, M. L. 1978. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation, Academia Press*, 169–179.

Ryffel, T.; Trask, A.; Dahl, M.; Wagner, B.; Mancuso, J.; Rueckert, D.; and Passerat-Palmbach, J. 2018. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*.

Sav, S.; Pyrgelis, A.; Troncoso-Pastoriza, J. R.; Froelicher, D.; Bossuat, J.-P.; Sousa, J. S.; and Hubaux, J.-P. 2020. POSEIDON: Privacy-Preserving Federated Neural Network Learning. *arXiv preprint arXiv:2009.00349*.

Shae, Z.; and Tsai, J. 2018. Transform Blockchain into Distributed Parallel Computing Architecture for Precision Medicine. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 1290–1299.

Shamir, A. 1979. How to Share a Secret. *Commun. ACM*, 22(11): 612–613.

Sharon, C.; and Gari, S. 2018. Technical advantages of Hyperledger Fabric for blockchain networks. https://developer.ibm.com/technologies/blockchain/articles/top-technical-advantages-of-hyperledger-fabric-for-blockchain-networks/. Accessed: 2021-05-01.

Shayan, M.; Fung, C.; Yoon, C.; and Beschastnikh, I. 2018. Biscotti: A Ledger for Private and Secure Peer-to-Peer Machine Learning. *ArXiv*, abs/1811.09904.

Shokri, R.; and Shmatikov, V. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC*

conference on computer and communications security, 1310–1321.

Song, L.; and Mittal, P. 2020. Systematic Evaluation of Privacy Risks of Machine Learning Models. arXiv:2003.10595.

Taheri, R.; Javidan, R.; Shojafar, M.; Pooranian, Z.; Miri, A.; and Conti, M. 2020. On defending against label flipping attacks on malware detection systems. *Neural Computing and Applications*, 1–20.

Tolpegin, V.; Truex, S.; Gursoy, M. E.; and Liu, L. 2020. Data Poisoning Attacks Against Federated Learning Systems. arXiv:2007.08432.

Truex, S.; Baracaldo, N.; Anwar, A.; Steinke, T.; Ludwig, H.; Zhang, R.; and Zhou, Y. 2019. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 1–11.

Tsuzuku, Y.; Imachi, H.; and Akiba, T. 2018. Variance-based Gradient Compression for Efficient Distributed Deep Learning. arXiv:1802.06058.

Wang, W.; Chen, Z.; Yan, X.; and Tian, J. ???? Cortex - AI on Blockchain: The Decentralized AI Autonomous System. Technical report.

Wood, D. 2014. ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER.

Xie, C.; Koyejo, S.; and Gupta, I. 2019. Zeno: Distributed Stochastic Gradient Descent with Suspicion-based Fault-tolerance. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 6893–6901. PMLR.

Xu, R.; Baracaldo, N.; Zhou, Y.; Anwar, A.; and Ludwig, H. 2019. HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning. In Cavallaro, L.; Kinder, J.; Afroz, S.; Biggio, B.; Carlini, N.; Elovici, Y.; and Shabtai, A., eds., *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2019, London, UK, November 15, 2019*, 13–23. ACM.

Xu, Y.; and Huang, Y. 2020. Segment Blockchain: A Size Reduced Storage Mechanism for Blockchain. *IEEE Access*, 8: 17434–17441.

Yang, Q.; Liu, Y.; Chen, T.; and Tong, Y. 2019. Federated Machine Learning. *ACM Transactions on Intelligent Systems and Technology*, 10(2): 1–19.

Yong, H.; Lee, C.; and Wang, D. 2017. DeepBrain Chain: Artificial intelligence computing platform driven by blockchain. Technical report.

Zhang, C.; Li, S.; Xia, J.; Wang, W.; Yan, F.; and Liu, Y. 2020. BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 493–506. USENIX Association. ISBN 978-1-939133-14-4.

Zhou, S.; Huang, H.; Chen, W.; Zhou, P.; Zheng, Z.; and Guo, S. 2020. PiRATE: A Blockchain-Based Secure Framework of Distributed Machine Learning in 5G Networks. *IEEE Network*, 34(6): 84–91.

Zhu, L.; Liu, Z.; and Han, S. 2019. Deep Leakage from Gradients. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32, 14774–14784. Curran Associates, Inc.

Appendix

More Detail on Poisoning Attacks

In poisoning attacks, the adversary $\mathcal{A}^c$ manipulates the local models W_i of $k < \frac{K}{2}$ clients to obtain poisoned models W_i' which are then aggregated into the global model G_t and affect its behavior. Poisoning attacks can be divided into untargeted and targeted attacks. In untargeted attacks, the adversary's goal is merely to impact (deteriorate) the benign performance of the aggregated model, while in targeted attacks (also called backdoor attacks), the adversary wants the poisoned model G_t' to behave normally on all inputs except for specific attacker-chosen inputs $x \in \mathcal{I}_{\mathcal{A}^c}$ for which attacker-chosen (incorrect) predictions should be output (Nguyen et al. 2021).

Data Poisoning Attack In a data poisoning attack, the adversary $\mathcal{A}^c$ adds manipulated "poisoned" data to the training data used to train local model W_i (Nguyen et al. 2021). For this, the adversary can implement a label flipping attack (Tolpegin et al. 2020): given a source class c_{src} and a target class c_{target} from $\mathcal{C}$, each malicious participant $\mathcal{A}_i$ modifies their dataset D_i as follows: For all instances in D_i whose class is c_{src}, change their class to c_{target}. We denote this attack by $c_{src} \rightarrow c_{target}$. The goal of the attack is to make the final global model G_t more likely to misclassify the primary model task (Nguyen et al. 2021; Tolpegin et al. 2020).

Algorithm 2: Constrain-and-scale (Bagdasaryan et al. 2020) for model poisoning attack, where the objective function is modified by adding an anomaly detection term $\mathcal{L}_{ano}$; $\mathcal{L}_{model} = \alpha \times \mathcal{L}_{class} + (1-\alpha) \times \mathcal{L}_{ano}$. Hyperparameter α controls the importance of evading anomaly detection.

Constrain-and-scale($\mathcal{D}_{local}, D_{backdoor}$)
Initialize attacker's model X and loss function l
$X \leftarrow G^t$
$l \leftarrow \alpha \times \mathcal{L}_{class} + (1-\alpha) \times \mathcal{L}_{ano}$
for *epoch* $e \in E_{adv}$ **do**
  if $\mathcal{L}_{class}(X, D_{backdoor}) < \epsilon$ **then**
    break; `// Early stop, if model converges`
  end
  for *batch* $b \in \mathcal{D}_{local}$ **do**
    $b \leftarrow$ replace($c, b, D_{backdoor}$)
    $X \leftarrow X - lr_{adv} \times \nabla l(X, b)$
  end
  if *epoch* $e \in$ *step_sched* **then**
    $lr_{adv} \leftarrow lr_{adv}/step\_rate$
  end
  $\widetilde{L}^{t+1} \leftarrow \gamma(X - G^t) + G^t$; `// Scale up the model before submission`
end
return $\widetilde{L}^{t+1}$

Model Poisoning Attack In a model poisoning attack, the adversary $\mathcal{A}^c$ manipulates the training algorithms, their parameters, or directly manipulates (e.g., by scaling) the model W_i'. When performing the attack, the adversary seeks to maximize attack impact while ensuring the distance (e.g., Euclidean distance) between poisoned models W' and benign models W remains below the detection threshold ϵ such as Multi-KRUM (Blanchard et al. 2017) of the aggregator to evade possible anomaly detection performed by the aggregator on individual clients' model updates (Bagdasaryan et al. 2020; Tolpegin et al. 2020; Nguyen et al. 2021).

Bagdasaryan et al. (Bagdasaryan et al. 2020) propose a model poisoning attack that uses *constrain-and-scale* techniques to create a model that does not look anomalous and replaces the global model after averaging with the other participants' models (see Algorithm 2).