OmniLytics: A Blockchain-based Secure Data Market for Decentralized Machine Learning

Jiacheng Liang [1 2] **Wensi Jiang** [1 3] **Songze Li** [1]

Abstract

We propose OmniLytics, a blockchain-based secure data trading marketplace for machine learning applications. Utilizing OmniLytics, many distributed data owners can contribute their private data to collectively train a ML model requested by some model owners, and get compensated for data contribution. OmniLytics enables such model training while simultaneously providing 1) model security against curious data owners; 2) data security against curious model and data owners; 3) resilience to malicious data owners who provide faulty results to poison model training; and 4) resilience to malicious model owner who intents to evade the payment. OmniLytics is implemented as a smart contract on the Ethereum blockchain to guarantee the atomicity of payment. In OmniLytics, a model owner publishes encrypted initial model on the contract, over which the participating data owners compute gradients using their private data, and securely aggregate the gradients through the contract. Finally, the contract reimburses the data owners, and the model owner decrypts the aggregated model update. We implement a working prototype of OmniLytics on Ethereum, and perform extensive experiments to measure its gas cost and execution time under various parameter combinations, demonstrating its high computation and cost efficiency and strong practicality.

[*]Equal contribution [1]Hong Kong University of Science and Technology, Hong Kong, China [2]University of Electronic Science and Technology of China, Chengdu, China [3]Central South University, Changsha, China. Correspondence to: Jiacheng Liang <jliangbb@connect.ust.hk>, Wensi Jiang <jiangwensi@csu.edu.cn>, Songze Li <songzeli8824@gmail.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

1. Introduction

With the rapid development of sensing, processing, and storage capabilities of computing devices (e.g., smartphones and IoT devices), the collection and storage of data has been increasingly convenient and cost-effective (Cornet & Holden, 2018; Sheng et al., 2013). On the other hand, crowdsourcing big data has been shown to be extremely effective in improving the performance of various tasks in, e.g., healthcare, smart city, and recommender systems (Al Nuaimi et al., 2015; Hashem et al., 2016; Raghupathi & Raghupathi, 2014; Wang et al., 2018; Yin et al., 2013). The abundant supply of data stored locally at individual nodes and the huge demands from data-intensive applications incentivise the development of a data market, on which data owners can easily trade the rights of using their data with interested consumers for monetary returns.

Conventionally, a data market is often implemented as a centralized service platform that collects data from data owners and sells raw or processed data to the consumers (Krishnamachari et al., 2018; Mišura & Žagar, 2016; Niu et al., 2018a). This approach leaves the platform as a single point of security vulnerability for the data market, and corruption on the platform servers may lead to severe security issues including leakage of private data, faulty computation results, and manipulation of data price. A number of recent works have proposed to leverage technologies of decentralized systems like blockchains and smart contracts to tackle the weakness of the centralized implementation (see, e.g., (Duan et al., 2019; Koutsos et al., 2020; Özyilmaz et al., 2018; Ramachandran et al., 2018)).

To further improve data security, more advanced cryptographic techniques like homomorphic and functional encryption have been utilized to generate analytics over the raw data for consumers to purchase without revealing the data themselves (Duan et al., 2019; Koutsos et al., 2020; Niu et al., 2018a;b). However, we note that these approaches are limited in the following three aspects: 1) data owners upload the encrypted raw data on the blockchain, which leads to permanent loss of the data ownership to any adversarial party with the decryption key; 2) the available analytics are limited to simple linear combinations; 3) they still require a (trusted or untrusted) third party other than the data owner

and consumer acting like a broker or service provider to maintain the utility and security of trading session.

In this paper, our goal is to build a secure, robust, and broker-free data market for general machine learning applications. Particularly, a model owner would like to crowdsource training data from interested data owners through the data market for improving the quality of its ML models (e.g., a deep neural network for image classification). Moreover, we enforce the following privacy and security requirements:

- *Model privacy*: the ML model is only known at the model owner and not revealed to other parties;
- *Data privacy*: model owner learns nothing about data owners' private other than the learnt model;
- *Byzantine resistance*: 1) robust to malicious data owners who intentionally provide incorrect data, and 2) robust to malicious model owner who tries to evade paying for the model training.

We propose a novel blockchain-based data market named OmniLytics, which is the first implementation of an Ethereum smart contract (ETH) that simultaneously satisfies all the above security requirements. During a trading session, the model owner deploys a smart contract with the encrypted initial ML model, then interested data owners retrieve the published model, compute locally the gradients using their private data, and upload the gradients back to the contract. During this process, the data owners hide their local gradients with pair-wise masks generated according to the secure aggregation protocol (Bonawitz et al., 2017) to further protect data leakage. Moreover, OmniLytics implements the multi-Krum algorithm (Blanchard et al., 2017a;b) to remove faulty computation results from malicious data owners. Finally, the model owner decrypts the aggregated gradients from many data owners with its private key, and uses the results to update its model. The training reward is provided by the model owner when deploying the contract, and is automatically distributed to honest data owners by the contract, preventing malicious model owners from evading payments after obtaining the trained model.

We implement an Ethereum smart contract SecGraCollect and the off-chain application of the proposed OmniLytics data market. We conduct extensive experiments to measure the gas cost and the execution time of SecGraCollect under various combinations of system parameters. Within the considered parameter range, all interaction processes between the data owner and the contract are completed within 0.8s, and all interaction processes between the model owner and the contract are completed within 18s. This demonstrates the high efficiency and feasibility of the proposed OmniLytics data market.

Related works

Secure data markets. While traditional data markets are implemented as an intermediate service platform that helps to enforce desirable rules about data privacy, data ownership, and data usage, this requires full trust on the platform and leaves the platform the single point of failure in maintaining these properties. To resolve this issue, implementing the data market service over decentralized systems like blockchains has been recently proposed (Banerjee & Ruj, 2018; Duan et al., 2019; Koutsos et al., 2020; Özyilmaz et al., 2018; Ramachandran et al., 2018). In these implementations, encrypted data are uploaded to the blockchain, on which a smart contract with funding deposit from the consumer is executed automatically, guaranteeing the atomicity of the payment. To further enhance the data privacy and robustness to malicious behaviors, more advanced techniques like homomorphic encryption, functional encryption, and differential privacy have been utilized to generate simple analytics over the raw data for sale without revealing the individual data points (Duan et al., 2019; Koutsos et al., 2020; Niu et al., 2018b), and zero-knowledge proofs and trusted hardware like Intel SGX have been used to guarantee the correctness of the computations (Duan et al., 2019; Koutsos et al., 2020; Niu et al., 2018a).

Federated learning on blockchains. Federated learning (FL) (McMahan et al., 2017) has recently been proposed as a privacy-preserving framework for distributed ML where a set of clients, instead directly uploading their private data to the cloud, upload the gradients computed from the data, which are aggregated at a cloud server to update a global model. In addition, techniques of masking local gradients like secure aggregation (Bell et al., 2020; Bonawitz et al., 2017; So et al., 2021) and differential privacy (Geyer et al., 2017; Wei et al., 2020) have been developed for FL to further protect potential data leakage.

Recently, it has been proposed to execute FL tasks on blockchains to combat server corruption and facilitate a more fair and transparent reward distribution (see, e.g., (Kim et al., 2019; Liu et al., 2020; Ma et al., 2020; Shayan et al., 2020; Zhao et al., 2020)). In (Zhao et al., 2020), a FL on blockchain system is designed for the smart appliance manufactures to learn a ML model from customers' data. Differential privacy techniques are applied on the clients' extracted features to protect data privacy from the computed gradients that are uploaded on the blockchain. A reputation-based incentive mechanism is designed to reward contributing customers and punish malicious ones. However, one weakness of the design in (Zhao et al., 2020) is that the initial model of the manufacturer is revealed to public, which may not be desirable for the model owner, especially for an iterative training session where the initial model for an iteration is learnt using customers' data from previous iterations.

Also, (Zhao et al., 2020) does not provide a blockchain implementation of their design. In (Liu et al., 2020), a P2P payment blockchain protocol is designed to compute and distribute the profit in a FL session, based on evaluations of Shapley values. The protocol in (Liu et al., 2020) is fundamentally different from the proposed OmniLytics data market as FL and reward distribution are performed on two separate systems in (Liu et al., 2020).

2. System description

We consider a network of many compute nodes (e.g., mobile devices like smartphones, or institutional entities like hospitals, banks, and companies), each of which has some local data storage and processing power. Some node would like to learn a machine learning model (e.g., to predict the preference of a customer on a particular product). We call such node the model owner, denoted by MO. However, as the MO may not have sufficient amount of data locally to train a good model, it intends to crowdsource data from other nodes to improve the model quality. In return, the MO compensates the nodes who contribute to the model training with their local private data. We name these nodes as data owners, denoted by DOs.

To facilitate a secure and faithful data trading process as described above, we aim to build a secure data market that meets the following service and security requirements:

- **Symmetric sessions.** Any node in the network can freely choose to be an MO and initialize a data trading session for local model training. Any node is also free to join any on-going data trading session to contribute its data to MO's model training.

- **Model and data privacy.** The MO would not like to directly reveal its model (before and after training) to the DOs, as the model itself may leak confidential information about the MO (e.g., the model might be pre-trained using MO's private data). On the other hand, a DO would not like to reveal its private data directly to the MO and other DOs, as that would lead to permanent relinquishing control over their data.

- **Byzantine resistance.** The designed data market should provide robustness to malicious behaviors of Byzantine MO and DOs. Specifically, it should protect the quality of the trained model from being undermined by malicious DOs who might arbitrarily deviate from the training protocol. Also, it should enforce that honest DOs who faithfully follow the protocol get properly compensated for their contributions to the model training.

3. Secure federated machine learning

To protect the data privacy of the DOs, we adopt the privacy-preserving distributed machine learning framework named federated learning (McMahan et al., 2017), where gradients computed from private data instead of the data itself are exchanged between participating nodes to collaboratively train a global model. Additionally, we employ secure multi-party computing techniques to further protect the privacy of the MO's model and the DOs' data, and the security of the training process.

3.1. Federated learning framework

A federated learning (FL) instance consists of a central server and a group of N clients. Each client k has locally a dataset $\mathcal{S}_k = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_{M_k}, \boldsymbol{y}_{M_k})\}$ of M_k data samples. Each data sample $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ consists of an input vector $x_i \in \mathbb{R}^d$ and its label $y_i \in \mathbb{R}^p$ for some input dimension d and output dimension p. The goal of FL is to train a global model $\boldsymbol{W}$ (e.g., a neural network) to minimize the objective function $\mathcal{L}(\boldsymbol{W}) = \sum_{k=1}^{N} p_k \mathcal{L}_k(\boldsymbol{W})$. Here $\mathcal{L}_k(\boldsymbol{W}) = \frac{1}{M_k} \sum_{i=1}^{M_k} \frac{1}{2} ||\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i||_2^2$ is the empirical mean-square loss at client k, and $\hat{\boldsymbol{y}}_i$ is the predicted label of $\boldsymbol{x}_i$ using $\boldsymbol{W}$. The weight $p_k \triangleq \frac{M_k}{\sum_{k=1}^{N} M_k}$.

The server collaborates with the clients to train the global model using gradient descent over multiple iterations. To start with, the server broadcasts an initial model $\boldsymbol{W}^{(0)}$ to all clients. In each iteration t of the training process, each client k uses the global model received from the last iteration $\boldsymbol{W}^{(t-1)}$ and its local private data $\mathcal{S}_k$ to compute the gradient $\boldsymbol{G}_k^{(t-1)} = \frac{\partial \mathcal{L}_k(\boldsymbol{W}^{(t-1)})}{\partial \boldsymbol{W}^{(t-1)}}$ and sends it to the server. The server aggregates the received gradients from the N clients and updates the model for some learning rate η as follows

$$\boldsymbol{W}^{(t)} = \boldsymbol{W}^{(t-1)} - \eta \sum_{k=1}^{N} p_k \boldsymbol{G}_k^{(t-1)}. \tag{1}$$

We consider training a deep neural network $\boldsymbol{W}$. In what follows, we describe the security mechanisms we adopt to protect the privacy of the model parameters and the clients' private data, for a single iteration of federated learning (hence we will omit all iteration indices).

3.2. Privacy-preserving techniques for FL

3.2.1. Global model encryption

For a feed-forward neural network $\boldsymbol{W}$ of L layers, as depicted in Figure 1, we represent the model parameters in layer $l \in \{1, \ldots, L\}$ as a matrix $\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$, where n_l denotes the number of neurons in layer l. To protect the privacy of model parameters, as done in (Yang et al., 2020), the server encrypts the model parameters $\boldsymbol{W}$ into $\widetilde{\boldsymbol{W}}$ before

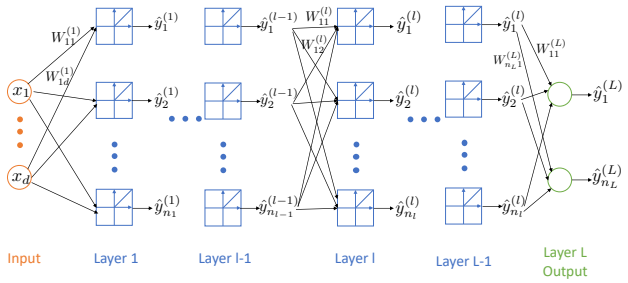

Figure 1. Illustration of a feed-forward neural network with L layers and ReLU activation function.

distributing them to the clients such that

$$\widetilde{\boldsymbol{W}}^{(l)} = \begin{cases} \boldsymbol{R}^{(l)} \circ \boldsymbol{W}^{(l)}, & 1 \le l \le L-1, \\ \boldsymbol{R}^{(l)} \circ \boldsymbol{W}^{(l)} + \boldsymbol{R}^{(a)}, & l = L, \end{cases} \tag{2}$$

where $\circ$ denotes the Hadamard product. Here for each layer $1 \le l \le L-1$, the multiplicative masks are generated as

$$R_{ij}^{(l)} = \begin{cases} r_i^{(1)}, & l = 1, \\ r_i^{(l)} / r_j^{(l-1)}, & 2 \le l \le L-1, \\ 1/r_j^{(L-1)}, & l = L, \end{cases} \tag{3}$$

for some positive noise vector $\boldsymbol{r}^{(l)} = [r_1^{(l)}, \ldots, r_{n_l}^{(l)}]^\top \in \mathbb{R}_{>0}^{n_l}$ randomly sampled at the server. For the last layer L of $\boldsymbol{W}$, for some additive noise vectors $\boldsymbol{r}^{(a)} = [r_1^{(a)}, \ldots, r_{n_L}^{(a)}]^\top \in \mathbb{R}^{n_L}$ and $\boldsymbol{\gamma} = [\gamma_1, \ldots, \gamma_{n_L}]^\top \in \mathbb{R}^{n_L}$, both generated randomly at the server, we have $R_{ij}^{(a)} = \gamma_i \cdot r_i^{(a)}$.

The server publishes the encrypted global model $\widetilde{\boldsymbol{W}} = \{\widetilde{\boldsymbol{W}}^{(l)}\}_{l=1}^{L}$ and the noise vector $\boldsymbol{r}^{(a)}$ for the clients to perform local model training.

3.2.2. Local Gradient Computation

After receiving the encrypted global model $\widetilde{\boldsymbol{W}} = \{\widetilde{\boldsymbol{W}}^{(l)}\}_{l=1}^{L}$ and the noise vector $\boldsymbol{r}^{(a)}$, each client k computes the gradient over the encrypted model using its local data in $\mathcal{S}_k$, following the forward and backward approach.

During the forward propagation, for each data sample $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}_k$, the output vector of layer l, $\widehat{\boldsymbol{y}}^{(l)} = [\widehat{y}_1^{(l)}, \ldots, \widehat{y}_{n_l}^{(l)}]^\top \in \mathbb{R}^{n_l}$ is computed as

$$\widehat{\boldsymbol{y}}^{(l)} = \begin{cases} \boldsymbol{x}, & l = 0, \\ \mathrm{ReLU}(\widetilde{\boldsymbol{W}}^{(l)} \widehat{\boldsymbol{y}}^{(l-1)}), & 1 \le l \le L-1, \\ \widetilde{\boldsymbol{W}}^{(l)} \widehat{\boldsymbol{y}}^{(l-1)}, & l = L. \end{cases} \tag{4}$$

After obtaining $\{\widehat{\boldsymbol{y}}\}_{l=1}^{L}$, the client performs backward propagation to compute the gradient of $\widetilde{\boldsymbol{W}}$ subject to the MSE loss, i.e., $\mathcal{L}(\widetilde{\boldsymbol{W}}; (\boldsymbol{x}, \boldsymbol{y})) = \frac{1}{2} ||\widehat{\boldsymbol{y}}^{(L)} - \boldsymbol{y}||_2^2$. We know from Theorem 2 of (Yang et al., 2020) that for each layer $l = 1, \ldots, L$, the partial gradient with respect to $\widetilde{\boldsymbol{W}}^{(l)}$ satisfies

$$\begin{aligned} \frac{\partial \mathcal{L}(\widetilde{\boldsymbol{W}}; (\boldsymbol{x}, \boldsymbol{y}))}{\partial \widetilde{\boldsymbol{W}}^{(l)}} = & \frac{1}{\boldsymbol{R}^{(l)}} \circ \frac{\partial \mathcal{L}(\boldsymbol{W}; (\boldsymbol{x}, \boldsymbol{y}))}{\partial \boldsymbol{W}^{(l)}} \\ & + \boldsymbol{r}^\top \boldsymbol{\sigma}_{(\boldsymbol{x}, \boldsymbol{y})}^{(l)} - v \boldsymbol{\beta}_{(\boldsymbol{x}, \boldsymbol{y})}^{(l)}, \end{aligned} \tag{5}$$

where $\frac{1}{\boldsymbol{R}^{(l)}}$ is the $n_l \times n_{l-1}$ matrix whose (i,j)-th entry is $\frac{1}{R_{ij}^{(l)}}$, and $\boldsymbol{r} = \boldsymbol{\gamma} \circ \boldsymbol{r}^{(a)}$. Here, for $\alpha = \widehat{y}_1^{(L-1)} + \cdots + \widehat{y}_{n_{L-1}}^{(L-1)}$, we have

$$\begin{cases} \boldsymbol{\sigma}_{(\boldsymbol{x}, \boldsymbol{y})}^{(l)} = \alpha \frac{\partial \widehat{\boldsymbol{y}}^{(L)}}{\partial \widetilde{\boldsymbol{W}}^{(l)}} + \left(\frac{\partial \mathcal{L}(\widetilde{\boldsymbol{W}}; (\boldsymbol{x}, \boldsymbol{y}))}{\partial \widehat{\boldsymbol{y}}^{(L)}} \right)^\top \frac{\partial \alpha}{\partial \widetilde{\boldsymbol{W}}^{(l)}}, \\ \boldsymbol{\beta}_{(\boldsymbol{x}, \boldsymbol{y})}^{(l)} = \alpha \frac{\partial \alpha}{\partial \widetilde{\boldsymbol{W}}^{(l)}}, \\ v = \boldsymbol{r}^\top \boldsymbol{r}. \end{cases} \tag{6}$$

Client k repeats this computation for all data points in $\mathcal{S}_k$, and computes the average gradients and the corresponding noise terms, for each layer $l = 1, \ldots, L$:

$$\begin{cases} \nabla \mathcal{L}_k(\widetilde{\boldsymbol{W}}^{(l)}) = \frac{1}{M_k} \sum\limits_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}_k} \frac{\partial \mathcal{L}(\widetilde{\boldsymbol{W}}; (\boldsymbol{x}, \boldsymbol{y}))}{\partial \widetilde{\boldsymbol{W}}^{(l)}}, \\ \widetilde{\boldsymbol{\sigma}}_k^{(l)} = \frac{1}{M_k} \sum\limits_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}_k} \boldsymbol{\sigma}_{(\boldsymbol{x}, \boldsymbol{y})}^{(l)}, \\ \boldsymbol{\beta}_k^{(l)} = \frac{1}{M_k} \sum\limits_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}_k} \boldsymbol{\beta}_{(\boldsymbol{x}, \boldsymbol{y})}^{(l)}. \end{cases} \tag{7}$$

Finally, each client k computes $\boldsymbol{\sigma}_k^{(l)} = \mathrm{diag}(\boldsymbol{r}^{(a)}) \widetilde{\boldsymbol{\sigma}}_k^{(l)}$, where $\mathrm{diag}(\boldsymbol{r}^{(a)})$ is a diagonal matrix with diagonal entries $r_1^{(a)} \ldots, r_{n_L}^{(a)}$.

3.2.3. Secure Aggregation

While the local computations in (7) are performed on encrypted model parameters, directly communicating them to the server may still reveal information about the private data in $\mathcal{S}_k$. To prevent data leakage, we adopt the security aggregation protocol in (Bonawitz et al., 2017) to mask local computation results. Specifically, for each layer l, each client k concatenates its local computation results $\nabla \mathcal{L}_k(\widetilde{\boldsymbol{W}}^{(l)})$, $(\boldsymbol{\sigma}_{k,1}^{(l)}, \ldots, \boldsymbol{\sigma}_{k,n_L}^{(l)})$, and $\boldsymbol{\beta}_k^{(l)}$ into a vector $\boldsymbol{q}_k^{(l)}$ of length $(n_L + 2) n_l n_{l-1}$, and sends a masked vector $\widetilde{\boldsymbol{q}}_k^{(l)} = \boldsymbol{q}_k^{(l)} + \boldsymbol{z}_k^{(l)}$ with some random mask $\boldsymbol{z}_k^{(l)}$ to the server. The secure aggregation protocol guarantees that, let $\boldsymbol{Q}_k = [\boldsymbol{q}_k^{(1)}, \ldots, \boldsymbol{q}_k^{(L)}]$, 1) server does not know anything about each individual $\boldsymbol{Q}_k$; and 2) server can compute the aggregation of the local computations of the N clients $\sum_{k=1}^{N} \boldsymbol{Q}_k$.

3.2.4. Gradient Decryption

Given the aggregated computation result $\sum_{k=1}^{N} \frac{M_k}{M} \boldsymbol{Q}_k$ with $M = \sum_{k=1}^{N} M_k$, the server would like to recover the gra-

dient of model parameter $\boldsymbol{W}$ over the data batch $\bigcup_{1\le k\le N}\mathcal{S}_k$. To do that, for each layer l, with the aggregation results

$$\nabla\mathcal{L}(\widetilde{\boldsymbol{W}}^{(l)}) = \sum_{k=1}^{N}\frac{M_k}{M}\nabla\mathcal{L}_k(\widetilde{\boldsymbol{W}}^{(l)}),$$
$$\boldsymbol{\sigma}_i^{(l)} = \sum_{k=1}^{N}\frac{M_k}{M}\boldsymbol{\sigma}_{k,i}^{(l)}, \quad i=1,\dots,n_L,$$
$$\boldsymbol{\beta}^{(l)} = \sum_{k=1}^{N}\frac{M_k}{M}\boldsymbol{\beta}_k^{(l)},$$

the server uses its private variables $\boldsymbol{R}^{(l)}$, $\boldsymbol{\gamma}$, and v to recover the gradient as

$$\widehat{\boldsymbol{G}}^{(l)} = \boldsymbol{R}^{(l)} \circ \left(\nabla\mathcal{L}(\widetilde{\boldsymbol{W}}^{(l)}) - \sum_{i=1}^{n_L}\gamma_i\boldsymbol{\sigma}_i^{(l)} + v\boldsymbol{\beta}^{(l)}\right) \quad (8)$$

Theorem 1. *Given the aggregation result $\sum_{k=1}^{N}\frac{M_k}{M}\boldsymbol{Q}_k$, for each layer $l=1,\dots,L$, the server can correctly reconstruct the gradient over the data batch $\bigcup_{1\le k\le N}\mathcal{S}_k$ by performing the computation in (8), i.e., $\widehat{\boldsymbol{G}}^{(l)} = \nabla\mathcal{L}(\boldsymbol{W}^{(l)})$.*

Proof: See Appendix A. □

After the successful reconstruction of the gradient $\nabla\mathcal{L}(\boldsymbol{W}^{(l)})$ for each layer l, the server updates the model parameters following (1).

4. The proposed secure data market

While the above described techniques can help to protect the privacy of model parameters and DOs' local data in the federated learning framework, they are still insufficient to tackle the security challenge for which a malicious MO may falsely claim that no training results are received from the DOs and refuse to reimburse the DOs, or simply leave the system without paying for the model training. We propose to leverage blockchain technologies to resolve this problem. Moreover, we design OmniLytics, which is an end-to-end solution to provide a transparent, fair, yet private and secure data market.

Specifically, OmniLytics implements the gradient collection and reward distribution process through a smart contract named SecGraCollect on an underlying blockchain. SecGraCollect executes a single iteration of the model update as in (1), with the initial model from the MO, and the gradients collected over up to R rounds and from up to N DOs in each round. The secure federated learning techniques described in Section 3 are employed to protect the model privacy of the MO and the data privacy of the DOs. The use of smart contract enforces automatic payment towards participating DOs whose computation results are considered valid, via some verification mechanism implemented on the contract.

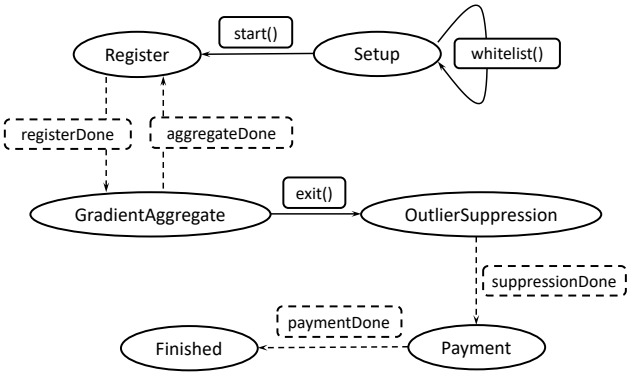

Figure 2. State transition of the smart contract SecGraCollect. The six states of SecGraCollect are represented by ovals. State transitions are triggered by either applying a method (in a solid box), or occurrence of an event (in a dashed box).

Gradient collection and aggregation. The MO initializes the data trading session by deploying a smart contract SecGraCollect with training reward deposit on the blockchain. As shown in Figure 2, SecGraCollect transitions between six states, i.e., Setup, Register, GradientAggregate, OutlierSuppression, Payment, and Finished. Upon deployment, SecGraCollect is in the Setup state with a set of DOs the MO would like to purchase data from specified by a whitelist() method. The MO encrypts its initial model $\boldsymbol{W}$ as described in Section 3.2.1, and issues a transaction with the start() method specifying the following public parameters:

- The encrypted model parameters $\widetilde{\boldsymbol{W}}$;
- Minimum number of data points required for each participating DO to compute its gradient, denoted by M_0;
- Number of rounds to collect gradients, denoted by R;
- Maximum number of distinct DOs to collect gradients from within each round, denoted by N.

Executing start() moves SecGraCollect into the Register state, and the contract starts to register for the DOs who are willing to participate in the gradient aggregation for the first round. In each round r, $r=1,\dots,R$, after N DOs have registered for this round, SecGraCollect moves to the GradientAggregate state and starts to collect local computation results from the DOs. Each of the N DOs registered for round r performs local computations on the encrypted model $\widetilde{\boldsymbol{W}}$ using M_0 private data points as specified in Section 3.2.2, and then sends the masked computation results to the contract following the secure aggregation protocol

in Section 3.2.3. After receiving the masked computation results from all registered DOs in that round, the contract aggregates them to obtain

$$\boldsymbol{A}_r = \frac{1}{N}\sum_{k=1}^{N} \boldsymbol{Q}_k = \frac{M_k}{M}\sum_{k=1}^{N} \boldsymbol{Q}_k, \tag{9}$$

where $\boldsymbol{Q}_k = [\boldsymbol{g}_k^{(1)}, \ldots, \boldsymbol{g}_k^{(L)}]$ are the local computation results of kth DO, and $M_k = M_0$ for all $k = 1, \ldots, N$. If otherwise the results of some DOs are still missing after certain amount of time, the aggregation in round r fails and we have $\boldsymbol{A}_r = \emptyset$.

After the aggregation process of round r, SecGraCollect moves back to the Register state for the next round $r+1$. Only DOs whose local computation results have not been incorporated in the aggregation results of any previous rounds are eligible to register. By the end of the aggregation process the contract SecGraCollect obtains a set of results $\{\boldsymbol{A}_r\}_{r\in\mathcal{P}}$, where $\mathcal{P} \subseteq \{1, \ldots, R\}$ denotes the indices of the rounds in which the secure aggregation had been successfully executed. The contract transits to the OutlierSuppression state if R rounds of gradient aggregation have been executed or is manually triggered by the exit() method.

Outlier suppression. During the gradient collection and aggregation process, Byzantine DOs may upload maliciously generated gradients which corrupt the aggregation results in $\{\boldsymbol{A}_r\}_{r\in\mathcal{P}}$. SecGraCollect adopts Byzantine resistance techniques to remove outliers in the OutlierSuppression state. Specifically, we assume that the DOs' data are i.i.d., and at most $\mu < \frac{1}{2}$ fraction of the aggregation results $\{\boldsymbol{A}_r\}_{r\in\mathcal{P}}$ may be corrupted by malicious DOs. The contract runs a distance-based outlier suppression algorithm m-Krum (Blanchard et al., 2017a;b) (Algorithm 1) on $\{\boldsymbol{A}_r\}_{r\in\mathcal{P}}$ to select a subset $\mathcal{P}' \subset \mathcal{P}$ of $|\mathcal{P}'| = m$ aggregation results that are considered computed correctly, for some $m < (1-2\mu)|\mathcal{P}| - 2$.

Reward distribution. The contract enters the Payment state after the outlier suppression and the set $\mathcal{P}'$ is obtained. The training reward deposited by the MO on the contract are evenly distributed into accounts of the DOs whose computation results from rounds in $\mathcal{P}'$ have been accepted.

Gradient decryption at the model owner. After the execution of the SecGraCollect contract, the MO obtains from the contract the selected aggregation results $\{\boldsymbol{A}_r\}_{r\in\mathcal{P}'}$. For each selected round $r \in \mathcal{P}'$, the MO decrypts the plain gradient $\nabla\mathcal{L}_r(\boldsymbol{W}^{(l)})$ according to (8), for each layer $l = 1, \ldots, L$, using its private variables $\boldsymbol{R}^{(l)}$, $\boldsymbol{\gamma}$, and v.

Finally, the MO obtains the aggregated gradients for each layer l:

$$\nabla\mathcal{L}(\boldsymbol{W}^{(l)}) = \sum_{r=1}^{|\mathcal{P}'|} \frac{1}{|\mathcal{P}'|}\nabla\mathcal{L}_r(\boldsymbol{W}^{(l)}), \tag{10}$$

and uses them to update its ML model.

5. Security analysis

In this section, we analyze the security properties of OmniLytics. Particularly, our analysis includes the following three aspects: 1) the confidentiality of the model parameters; 2) the privacy of each DO's data; and 3) the security of the model update.

5.1. Confidentiality of model parameters

Given the encrypted initial model $\widetilde{\boldsymbol{W}}$ and the additive noise vector $\boldsymbol{r}^{(a)}$, it is clear that the DOs could not recover the multiplicative masks $(\boldsymbol{R}^{(l)})_{l=1}^{L}$ and the additive masks $\boldsymbol{R}^{(a)}$ in (2). Hence, they would not be able to recover the model parameters $\boldsymbol{W}$.

5.2. Confidentiality of local data

While each data owner can participate in computation aggregation in at most one round in the contract SecGraCollect, its private data is only related to the aggregation result $\boldsymbol{A}_r$ for a single round r. We consider an honest-but-curious threat model where a subset $\mathcal{C} \subset \{1, \ldots, N\}$ of DO in round r may collude to infer the private data of some DO in the same round. Based on the privacy guarantee of the secure aggregation protocol (Bonawitz et al., 2017) employed by the contract, we argue that as long as the number of colluding DOs $|\mathcal{C}|$ is less than some secure parameter T[1], no information about other DOs' private data other than the summation of their local computation results can be inferred.

5.3. Security of model update

To combat malicious data owners uploading faulty computation results to the contract, we employ the m-Krum algorithm from (Blanchard et al., 2017b) to select the aggregation results from a subset of $\mathcal{P}' \subset \mathcal{P}$, which are considered to be close to the expected value with respect to the underlying data distribution.

We note that with our construction, all the aggregation results $\{\boldsymbol{A}_r\}_{r\in\mathcal{P}}$ from the successful rounds are independently and identically distributed (since each data owner performs local computations with M_0 data points, and each round aggregates results from N DOs). Therefore, according to Proposition 2 and Proposition 3 in (Blanchard et al., 2017b), as long as $|\mathcal{P}'| < (1-2\mu)|\mathcal{P}| - 2$, where μ is the maximum fraction of the aggregation results that may be corrupted, the estimated overall gradient in (10) provides a

[1]Each DO's secret keys to generate random masks are secret shared with other DOs such that any T colluding DOs can reveal the secret keys of any data owner.

close approximation of the true gradient, which leads to the convergence of the model training.

6. Experiments

We implement a working prototype of OmniLytics over Ethereum, using Solidity (Sol) to develop the contracts, Python for the off-chain applications and Pytorch (pyt) for the neural network training.

For all experiments we consider training a shallow two-layer neural network for the Boston House Price Dataset (Bos). The network contains an input layer of dimension 13, a hidden layer with ReLu activation of dimension 16, and an output layer of dimension 1.

6.1. Setup and system-level optimization

We deploy the smart contract SecGraCollect via the Remix IDE (Rem) on the local Geth Ethereum Testnet (get). We connect the off-chain applications to the smart contracts using the Web3py library (Web) and monitor the created transactions using Etherscan. Each data owner is connected to the Geth network with a unique Ethereum address. We conduct experiments on a machine with AMD R5-5600X CPU @3.70 GHz, Nvidia RTX3070 GPU, 32 GB of Memory and 1 TB SSD.

SecGraCollect consists of four major operations: Register, PubKeyInteract, GradientAggregate, and OutlierSuppression. A model owner deploys SecGraCollect on Ethereum, and performs OutlierSuppression with the contract after GradientAggregate. After performing their local computations, data owner instances greedily register with active contract to upload their results until the results are incorporated in the final aggregation. Data owners within the same aggregation group runs PubKeyInteract to exchange public keys for secure aggregation as done in (Bonawitz et al., 2017).

Data owners pay for the transaction fee to upload computation results and secure aggregation, which will be reimbursed by the model owner in the Payment phase.

During the secure aggregation process in each round, while the default Pytorch data type is $float32$, we scale each value by 10^8 and aggregate the integer part to improve the precision of the aggregation result. We turn on automatic mining mode and set the mining time to generate a new block to 1 second.

Parallel group aggregation. We perform a system-level optimization such that each round of secure aggregation is carried out in parallel to speed up contract processing. This means that if all the data owners in one round have submitted their local results, the process of secure aggregation would be performed. There is no need to wait until previous rounds are completed. When the last round is completed, model owner can initiate the multi-Krum process to obtain the final result.

6.2. Gas consumption measurement

We measure the deployment cost of the SecGraCollect contract, which depends on the size of its bytecode. We also measure the gas cost of executing each part of SecGraCollect, under various settings for number of rounds (R) and the number of data owners in each round (N).

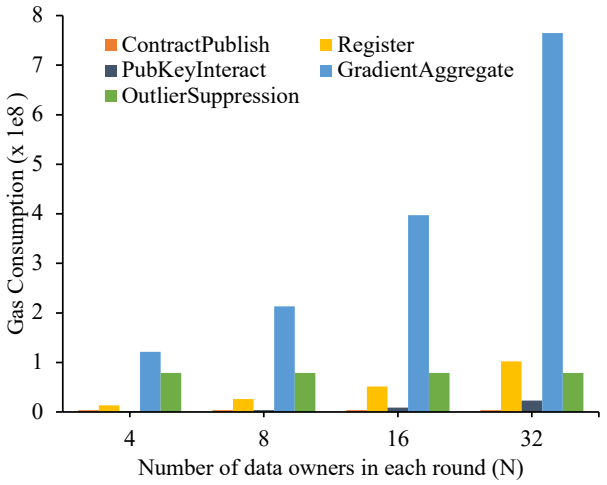

Figure 3. Gas consumption of the SecGraCollect contract for $R = 12$ aggregation rounds and $\mu = 20\%$ adversarial data owners, for different number of data owners in each round.

First, we fix the number of GradientAggregate rounds $R = 12$ and an estimated fraction of adversarial data owners $\mu = 20\%$, and measure the gas cost of the SecGraCollect contract for different number of DOs in each round (N). We observe in Figure 3 that as N varies from 4 to 32, the total gas consumption of Register, PubKeyInteract, and GradientAggregate increases linearly N. In contrast, since the number of rounds R has not changed, the computation complexity of the multi-Krum algorithm does not change, and the gas cost of OutlierSuppression stays almost constant.

Next, we fix the number of data owners in each round $N = 16$ and $\mu = 20\%$, and evaluate the impact of number of groups R on the gas consumption. As shown in Figure 4, for fixed N, the gas costs of Register, PubKeyInteract, and GradientAggregate increase linearly with R. The computational complexity of multi-Krum scales quadratically with R, which leads to a faster increase in the gas cost of OutlierSuppression.

6.3. Running time measurement

We measure the running time of the contract, which consists of executing the Register, PubKeyInteract,

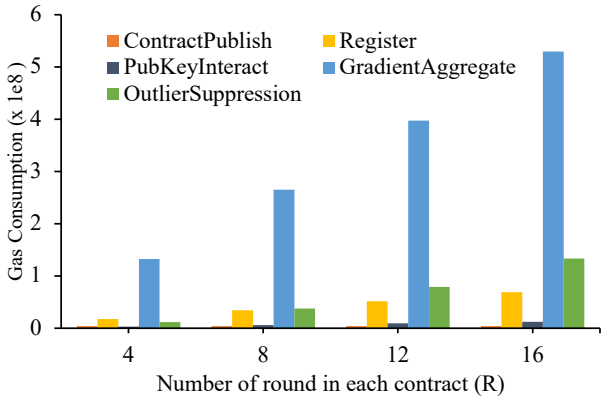

Figure 4. Gas consumption of the SecGraCollect contract for $N = 16$ data owners in each aggregation round and $\mu = 20\%$ adversarial data owners, with different number of rounds in the contract.

GradientAggregate, and OutlierSuppression steps. The measured execution time includes the time spent by data owners to retrieve function values from the contract, and to send transactions to the blockchain miners. We note that the time measured here does not include the block mining time and the time for model encryption and gradient decryption performed at the model owner.

In general, according to the run-time breakdowns in Tables 1 and 2, the PubKeyInteract step takes the least amount of time among the four as it simply returns a PubKey array to each DO in a group. The Register step lasts longer as its methods need to record the public keys of participating DOs on the contract and return the corresponding information to them. The GradientAggregate step costs more time as it includes the time for the DOs to send long computation results to the contract. Finally, the execution time of the multi-Krum algorithm in the OutlierSuppression step dominates the entire run-time of the contract.

Impact of N. We observe in Table 1 that the running times of the Register and PubKeyInteract steps are very small, and vary slowly as the number of data owners in each group increases. The running time of GradientAggregate increases with N, as more computation results need to be aggregated in each group. The execution time of the OutlierSuppression step dominates the entire contract execution. However, as we perform multi-Krum across groups after secure aggregation, increasing number of DOs within a group does not significantly increase the execution time of OutlierSuppression. We plot the total running time of SecGraCollect in Figure 5. We observe that the running time increases mildly with N as the execution time of the bottleneck operation OutlierSuppression is not significantly affected by N.

Table 1. Breakdown of the SecGraCollect running time (seconds) for different number of data owners in each round.

N	4	8	16	32
Register	0.0367	0.0382	0.0405	0.0392
PubKeyInteract	0.0218	0.0231	0.0250	0.0287
GradientAggregate	0.2400	0.3295	0.4712	0.7704
OutlierSuppression	6.1561	6.9003	7.9378	9.9919
Total	6.4546	7.2911	8.4745	10.8302

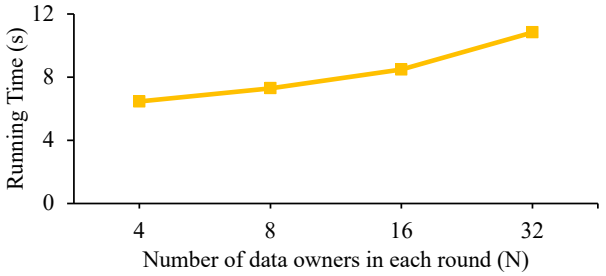

Figure 5. Total running time of the SecGraCollect contract for different number of data owners in each round.

Impact of R. We observe from Table 2 that the GradientAggregate time remains almost constant as the number of groups R increases. This is due to our system-level optimization to parallelize the aggregation operations of all groups. As expected, the run-time of the OutlierSuppression step increases significantly with R. We observe from figure 6 that, as R increases, the execution time of OutlierSuppression increases quadratically, which dominates the overall execution time.

Table 2. Breakdown of the SecGraCollect running time (seconds) for different number of aggregation rounds.

R	4	8	12	16
Register	0.0375	0.0392	0.0405	0.0383
PubKeyInteract	0.0234	0.0242	0.0250	0.0251
GradientAggregate	0.4553	0.4407	0.4712	0.4614
OutlierSuppression	0.8625	3.2678	7.9378	18.5000
Total	1.3786	3.7720	8.4745	19.0249

7. Conclusion

In this paper, we develop OmniLytics, the first Ethereum smart contract implementation of a secure data market for decentralized machine learning that simultaneously achieves 1) model privacy against curious data owners; 2) data privacy against curious model and data owners; 3) resilience against Byzantine data owners who intentionally provide

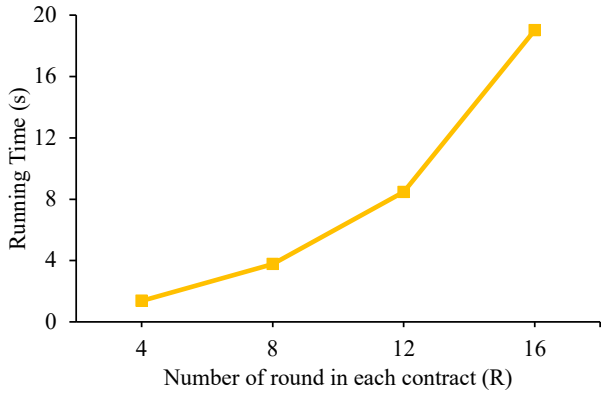

Figure 6. Total running time of the SecGraCollect contract for different number of aggregation rounds.

faulty results; and 4) resilience to Byzantine model owner who tries to evade payment. We develop and deploy an Ethereum smart contract SecGraCollect, and measure its gas consumption and run-time performance over various system parameters. Through extensive experiments we observe high computation and cost efficiency of SecGraCollect, which demonstrate the practicality of the proposed OmniLytics protocol as a secure data market.

References

The boston housing dataset. https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html. Accessed: 2021-06-05.

Ethereum smart contracts. https://ethereum.org/en/developers/docs/smart-contracts/. Accessed: 2021-06-05.

Remix - ethereum ide. https://remix.ethereum.org/. Accessed: 2021-06-05.

Solidity. https://docs.soliditylang.org/. Accessed: 2021-06-05.

Web3py. https://web3py.readthedocs.io/en/stable/. Accessed: 2021-06-05.

Official go implementation of the ethereum protocol. https://geth.ethereum.org/. Accessed: 2021-06-05.

Pytorch. https://pytorch.org/. Accessed: 2021-06-05.

Al Nuaimi, E., Al Neyadi, H., Mohamed, N., and Al-Jaroodi, J. Applications of big data to smart cities. *Journal of Internet Services and Applications*, 6(1):1–15, 2015.

Banerjee, P. and Ruj, S. Blockchain enabled data marketplace–design and challenges. *arXiv preprint arXiv:1811.11462*, 2018.

Bell, J. H., Bonawitz, K. A., Gascón, A., Lepoint, T., and Raykova, M. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1253–1269, 2020.

Blanchard, P., El Mhamdi, E. M., Guerraoui, R., and Stainer, J. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 118–128, 2017a.

Blanchard, P., Mhamdi, E. M. E., Guerraoui, R., and Stainer, J. Byzantine-tolerant machine learning. *arXiv preprint arXiv:1703.02757*, 2017b.

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

Cornet, V. P. and Holden, R. J. Systematic review of smartphone-based passive sensing for health and well-being. *Journal of biomedical informatics*, 77:120–132, 2018.

Duan, H., Zheng, Y., Du, Y., Zhou, A., Wang, C., and Au, M. H. Aggregating crowd wisdom via blockchain: A private, correct, and robust realization. In *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 1–10. IEEE, 2019.

Geyer, R. C., Klein, T., and Nabi, M. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

Hashem, I. A. T., Chang, V., Anuar, N. B., Adewole, K., Yaqoob, I., Gani, A., Ahmed, E., and Chiroma, H. The role of big data in smart city. *International Journal of Information Management*, 36(5):748–758, 2016.

Kim, H., Park, J., Bennis, M., and Kim, S.-L. Blockchained on-device federated learning. *IEEE Communications Letters*, 24(6):1279–1283, 2019.

Koutsos, V., Papadopoulos, D., Chatzopoulos, D., Tarkoma, S., and Hui, P. Agora: A privacy-aware data marketplace. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1211–1212. IEEE, 2020.

Krishnamachari, B., Power, J., Kim, S. H., and Shahabi, C. I3: An iot marketplace for smart communities. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 498–499, 2018.

Liu, Y., Ai, Z., Sun, S., Zhang, S., Liu, Z., and Yu, H. Fedcoin: A peer-to-peer payment system for federated learning. In *Federated Learning*, pp. 125–138. Springer, 2020.

Ma, C., Li, J., Ding, M., Shi, L., Wang, T., Han, Z., and Poor, H. V. When federated learning meets blockchain: A new distributed learning paradigm. *arXiv preprint arXiv:2009.09338*, 2020.

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

Mišura, K. and Žagar, M. Data marketplace for internet of things. In *2016 International Conference on Smart Systems and Technologies (SST)*, pp. 255–260. IEEE, 2016.

Niu, C., Zheng, Z., Wu, F., Gao, X., and Chen, G. Achieving data truthfulness and privacy preservation in data markets. *IEEE Transactions on Knowledge and Data Engineering*, 31(1):105–119, 2018a.

Niu, C., Zheng, Z., Wu, F., Tang, S., Gao, X., and Chen, G. Unlocking the value of privacy: Trading aggregate statistics over private correlated data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2031–2040, 2018b.

Özyilmaz, K. R., Doğan, M., and Yurdakul, A. Idmob: Iot data marketplace on blockchain. In *2018 crypto valley conference on blockchain technology (CVCBT)*, pp. 11–19. IEEE, 2018.

Raghupathi, W. and Raghupathi, V. Big data analytics in healthcare: promise and potential. *Health information science and systems*, 2(1):1–10, 2014.

Ramachandran, G. S., Radhakrishnan, R., and Krishnamachari, B. Towards a decentralized data marketplace for smart cities. In *2018 IEEE International Smart Cities Conference (ISC2)*, pp. 1–8. IEEE, 2018.

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

Sheng, X., Tang, J., Xiao, X., and Xue, G. Sensing as a service: Challenges, solutions and future directions. *IEEE Sensors journal*, 13(10):3733–3741, 2013.

So, J., Güler, B., and Avestimehr, A. S. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1):479–489, 2021.

Wang, Y., Kung, L., and Byrd, T. A. Big data analytics: Understanding its capabilities and potential benefits for healthcare organizations. *Technological Forecasting and Social Change*, 126:3–13, 2018.

Wei, K., Li, J., Ding, M., Ma, C., Yang, H. H., Farokhi, F., Jin, S., Quek, T. Q., and Poor, H. V. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security*, 15:3454–3469, 2020.

Yang, X., Feng, Y., Fang, W., Shao, J., Tang, X., Xia, S.-T., and Lu, R. Computation-efficient deep model training for ciphertext-based cross-silo federated learning. *arXiv e-prints*, pp. arXiv–2002, 2020.

Yin, H., Sun, Y., Cui, B., Hu, Z., and Chen, L. Lcars: a location-content-aware recommender system. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 221–229, 2013.

Zhao, Y., Zhao, J., Jiang, L., Tan, R., Niyato, D., Li, Z., Lyu, L., and Liu, Y. Privacy-preserving blockchain-based federated learning for iot devices. *IEEE Internet of Things Journal*, 2020.

Appendix

A. Proof of Theorem 1

First we have for each $l = 1, \ldots, L$,

$$\sum_{i=1}^{n_L} \gamma_i \boldsymbol{\sigma}_i^{(l)} = \sum_{i=1}^{n_L} \gamma_i \sum_{k=1}^{N} \frac{M_k}{M} \boldsymbol{\sigma}_{k,i}^{(l)} \tag{11}$$

$$= \sum_{k=1}^{N} \frac{M_k}{M} \sum_{i=1}^{n_L} \gamma_i r_i^{(a)} \widetilde{\boldsymbol{\sigma}}_{k,i}^{(l)} \tag{12}$$

$$= \sum_{k=1}^{N} \frac{M_k}{M} \boldsymbol{r}^\top \widetilde{\boldsymbol{\sigma}}_k^{(l)}. \tag{13}$$

In addition, we have by (5) that

$$\nabla \mathcal{L}_k(\widetilde{\boldsymbol{W}}^{(l)}) = \frac{1}{\boldsymbol{R}^{(l)}} \circ \nabla \mathcal{L}_k(\boldsymbol{W}^{(l)}) + \boldsymbol{r}^\top \widetilde{\boldsymbol{\sigma}}_k^{(l)} - v\boldsymbol{\beta}_k^{(l)}. \tag{14}$$

Thus, we can obtain that for each l,

$$\widehat{\boldsymbol{G}}^{(l)} = \boldsymbol{R}^{(l)} \circ \left(\nabla \mathcal{L}(\widetilde{\boldsymbol{W}}^{(l)}) - \sum_{i=1}^{n_L} \gamma_i \boldsymbol{\sigma}_i^{(l)} + v\boldsymbol{\beta}^{(l)} \right) \tag{15}$$

$$= \boldsymbol{R}^{(l)} \circ \sum_{k=1}^{N} \frac{M_k}{M} \left(\nabla \mathcal{L}_k(\widetilde{\boldsymbol{W}}^{(l)}) - \boldsymbol{r}^\top \widetilde{\boldsymbol{\sigma}}_k^{(l)} + v\boldsymbol{\beta}_k^{(l)} \right) \tag{16}$$

$$\overset{(14)}{=} \boldsymbol{R}^{(l)} \circ \sum_{k=1}^{N} \frac{M_k}{M} \frac{1}{\boldsymbol{R}^{(l)}} \circ \nabla \mathcal{L}_k(\boldsymbol{W}^{(l)}) \tag{17}$$

$$= \sum_{k=1}^{N} \frac{M_k}{M} \nabla \mathcal{L}_k(\boldsymbol{W}^{(l)}) = \nabla \mathcal{L}(\boldsymbol{W}^{(l)}) \tag{18}$$

B. Pseudo code of m-Krum

Algorithm 1 m-Krum

Input: $\{\boldsymbol{A}_r\}_{r \in \mathcal{P}}$: aggregation results of successfully executed rounds, μ: fraction of rounds whose result are corrupted

Output: $\{\boldsymbol{A}_r\}_{r \in \mathcal{P}'}$ with $|\mathcal{P}'| = m$

$\mathcal{T} = \mathcal{P}$, $\mathcal{P}' = \emptyset$
for $i = 1, \ldots, m$ **do**
  for $r \in \mathcal{T}$ **do**
    neighbors $= |\mathcal{T}| - \mu|\mathcal{P}| - 2$ closest (ℓ_2 distance) vectors to $\boldsymbol{A}_r$
    $S(r) = \sum_{\boldsymbol{A} \in \text{neighbors}} ||\boldsymbol{A}_r - \boldsymbol{A}||^2$
  end for
  $r^* = \arg\min_r S(r)$
  $\mathcal{T}$.remove(r^*)
  $\mathcal{P}'$.add(r^*)
end for
return $\{\boldsymbol{A}_r\}_{r \in \mathcal{P}'}$