Training Encrypted Models with Privacy-preserved Data on Blockchain

Lifeng Liu
Zhejiang University, China
fallenkliu@gmail.com

Fengda Zhang
Zhejiang University,China
fdzhang@gmail.com

Yifan Hu
Zhejiang University, China
yifan_hu@zju.edu.cn

Gang Huang
Zhejiang Lab, China
huanggang@zju.edu.cn

Chao Wu
Zhejiang University, Data Science Institute, Imperial
chao.wu@zju.edu.cn

Jiawei Yu
Cybervein Foundation, China
arthuryu@cybervein.org

Jun Xiao
Zhejiang University, China
junx@cs.zju.edu.cn

ABSTRACT

Currently, training neural networks often requires a large corpus of data from multiple parties. However, data owners are reluctant to share their sensitive data to third parties for modelling in many cases. Therefore, Federated Learning (FL) has arisen as an alternative to enable collaborative training of models without sharing raw data, by distributing modelling tasks to multiple data owners. Based on FL, we premodel sent a novel and decentralized approach to training encrypted models with privacy-preserved data on Blockchain. In our approach, Blockchain is adopted as the machine learning environment where different actors (i.e., the model provider, the data provider) collaborate on the training task. During the training process, an encryption algorithm is used to protect the privacy of data and the trained model. Our experiments demonstrate that our approach is practical in real-world applications.

CCS CONCEPTS

Computing methodologies→Distributed artificial intelligence • Computing methodologies→Distributed computing methodologies • Security and privacy→Cryptography

Keywords

Blockchain; Machine; LearningDistributed; Learning Privacy

1. INTRODUCTION

Deep learning architectures [11] [4] are "thirsty" for data [14], and in traditional centralized machine learning (ML) paradigm, training data are centrally held by organizations executing the learning algorithm. However, many distributed learning systems have extended this approach by coordinating a set of training nodes, which are fully trusted, to access shared data, such approaches are limited in a real-world context, due to the following reasons:

- Data Cost

The collection and storage of huge training datasets require a high cost, which increases the difficulty of machine learning.

- Legal Restrictions

Data is valuable and sensitive. In many cases, for example hospital patients' data, organizations cannot share their data, due to legal restrictions.

- Model Protection

Models based on massive data training are of high value. Unfortunately, the current model training method lacks protection for the model.

Facing these challenges, in traditional FL [10] [13], the server initializes a global model, and then sends this model to each client to compute distributed updates. These updates are then aggregated by the server to gain a global update. Such an approach suffers two issues: 1) It's possible to restore the sensitive data of clients from their model update; 2) There is no protection of model, as clients can easily get a trained model or an almost trained model. Thus, we propose an approach based on FL for these issues. Main contributions of the paper can be summarized as follows:

- We propose and implement a decentralized training platform based on Blockchain with smart contract

By separating calculations from data owners, we redesign the FL algorithm with Blockchain [15] and store data in IPFS [3]. The nodes on Blockchain are divided into three actors [1]: the data provider, the provider, and the computation provider. All of them cooperate with each other to complete the training task, according to the smart contract.

- We train encrypted models on data for model protection

On the platform, we designed an encryption algorithm that combines the homomorphic encryption [9], asymmetric encryption [2], and symmetric encryption [1]. Such an encryption algorithm can protect user data and models during the training.

2. PLATFORM DESIGN

Based on the paradigm we proposed in [17], we have designed

ICVISP 2019, August 26–28, 2019, Vancouver, BC, Canada
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7625-9/19/08…$15.00
https://doi.org/10.1145/3387168.3387211

this platform architecture in Figure 1. We attempt to sketch the major components of the platform, describe the challenges, and identify the open issues, in the hope that this will be useful to implement the platform.

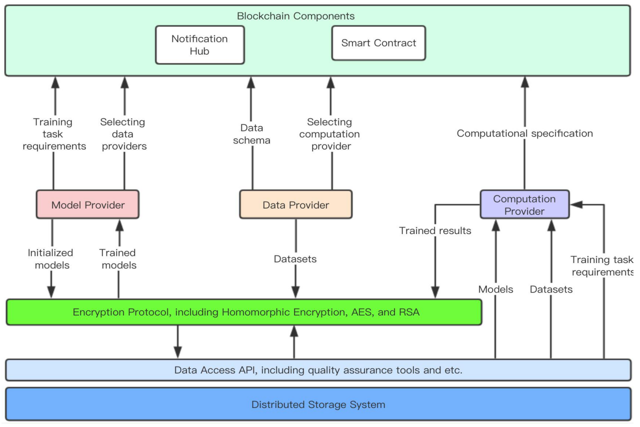

Blockchain Components
Notification Hub
Smart Contract
Training task requirements
Selecting data providers
Data schema
Selecting computation provider
Computational specification
Model Provider
Data Provider
Computation Provider
Trained results
Initialized models
Trained models
Datasets
Models
Datasets
Training task requirements
Encryption Protocol, including Homomorphic Encryption, AES, and RSA
Data Access API, including quality assurance tools and etc.
Distributed Storage System

Figure 1. Platform Design Overview.

There are three types of actors within this distributed modelling platform. Every node on Blockchain network belongs to one type of actors, including the model provider, the data provider, and the computation provider.

In brief, the model provider, who is the initiator of a training task, provides an initialized model. Training this model needs the training data from the data provider and the computing power of the computation provider. Finally, three actors cooperate to finish the training process securely and the model provider obtains a trained model to apply to the real scenarios.

We then discuss the detail features of the other components.

2.1 Blockchain Components

Blockchain has the advantage of decentralization and immutability for transactions. Therefore, we believe that Blockchain is naturally suitable for distributed learning algorithms. To facilitate the modelling tasks, the decentralized training platform has a set of components for supportive services, as shown in Figure 1:

- Notification hub

The notification hub is responsible for the transmission of messages among all nodes on Blockchain. It constructs an observer to monitor the verified transaction on the Ethereum [16] and decrypts the transaction information which is sent to the corresponding nodes according to the identity of the message.

- Smart contract

Smart contracts allow the performance of credible transactions without third parties. In our platform, the model provider will initialize the smart contract as the training task requirements. Model providers, data providers, and computation providers will perform the collaboration on the training task with each other.

2.2 Encryption Protocol

Encryption protocol takes the responsibility of guiding all nodes' encryption job. The key issue [17] is that we have found that using only homomorphic encryption cannot solve the problem of data and model protection. In this three-actor platform, the data provider obtains the HE public key distributed by the model provider for encrypting the data and distributes this data. Once the model provider gets this encrypted data, it will be able to decrypt the data with the HE private key. This breaks the rule that the model provider can't get the raw data from the provider.

Hence, we have designed a new encryption algorithm combining HE, ASE and RSA. Datasets and models will be encrypted by this algorithm before they are submitted to the storage system. The core idea of this algorithm is multi-layer encryption: data and models are encrypted by the HE public key, then these data and models are encrypted by the AES key, finally, the RSA protects these AES key. The detailed implementation of this algorithm is in Section 2.4.

This encryption algorithm is an option in the encryption protocol. We hope to contribute to other different encryption algorithms.

2.3 Platform Workflow

Now we will discuss the workflow of the platform.

- Preparation

Data providers submit the data schema to IPFS and get a hash address for informing others nodes. Then they send a hash address to Blockchain network. Meanwhile, computation providers will do a similar process to submit computational specification a hash address to the network, too.

- Model Provision

The model provider encrypts an initial (untrained or pre-trained) model and uploads the model to IPFS. Then, it initializes and publishes a smart contract (the training task) to the network.

- Model Training

After a learning task is distributed, a smart contract will be executed to match suitable data providers. With N matched data providers, each data provider will encrypt a dataset and search for M computation providers for training. When these computation providers accept the request, the data provider will upload the dataset to IPFS. Then, M computation providers will download the model and the dataset from IPFS according to hash address. Finally, M computation providers start to train the model with datasets.

- Model Aggregation

Computation providers will send the updated model to IPFS after the current training round is done. Then the corresponding model provider will download updated models and aggregate M models' parameters to the aggregated model.

- Iterative Training

According to the above steps, the task will repeat multiple rounds until the model provider get the trained model. The difference is that the model provider set the previous aggregative model to the current initial model. The implement is shown in Section 3.

2.4 Redesigned FL with Encryption

Redesigned FL allows us to make better use of idle computing resources belonging to others. Our algorithm is to implement training the encrypted model on privacy-preserved data. The goal is to ensure that only the model provider can obtain trained and unencrypted models, only the data provider holds the unencrypted and original training data, and the computing provider is only responsible for the calculation. As shown in Algorithm 1.

In a word, to get the trained model M, the model provider sets the aggregative model M_i into the initial model M_0 for iterative training. The training task is done when the global model M_i is converged. The model provider can apply the trained model M in reality.

Algorithm 1. An illustration of Redesigned FL with Encryption.

Algorithm 1 An illustration of Redesigned FL with Encryption. K_H^P/K_H^S represents the pair of homomorphic encryption keys, $K_{A_i}^P/K_{A_i}^S$ represents the pair of asymmetric encryption keys, and K_{B_i} represents the symmetric encryption key.

```
1:  **Model Provider executes:**
2:  initialize un-trained model $M_0$; broadcast smart contracts; select Data Providers
3:  generate HE $K_H^P/K_H^S$,RSA $K_{A_3}^P/k_{A_3}^S$, AES $K_{B_2}$
4:  $M_0 \xrightarrow{K_H^P} f_H(M_0)$
5:  broadcast $f_H(M_0)$ and $K_H^P$
6:  **while** $M_i$ *is not converged* **do**
7:      **Data Provider executes:**
8:      upload data schema, generate AES $K_{B_1}$
9:      **if** Data Provider accepts request from Model Provider **then**
10:         select $N$ Computation Providers, download their $K_{A_2}^P$, and $K_H^P$,
11:         $D \xrightarrow{K_H^P} f_H(D) \xrightarrow{K_{B_1}} f_{B_1}(f_H(D))$
12:         $K_{B_1} \xrightarrow{K_{A_2}^P} f_{A_2}(K_{B_1})$
13:         upload $f_{A_2}(K_{B_1})$ and $f_{B_1}(f_H(D))$
14:     **end if**
15:     **Computation Provider executes:**
16:     upload computational specification
17:     generate RSA $K_{A_2}^P/K_{A_2}^S$ and AES $K_{B_2}$
18:     **if** Computation Provider accepts request from Data Provider **then**
19:         download $f_H(M_i)$, $f_{B_1}(f_H(D))$, $f_{A_2}(K_{B_1})$
20:         $f_{A_2}(K_{B_1}) \xrightarrow{K_{A_2}^S} K_{B_1}$
21:         $f_{B_1}(f_H(D)) \xrightarrow{K_{B_1}} f_H(D)$
22:         (training $f_H(M_i)$ on $f_H(D)) \rightarrow f_H(M_i)$
23:         $f_H(M_i) \xrightarrow{K_{B_2}} f_{B_2}(f_H(M_i))$
24:         $K_{B_2} \xrightarrow{K_{A_3}^P} f_{A_3}(K_{B_2})$
25:         upload $f_{A_3}(K_{B_2})$, $f_{B_2}(f_H(M_i))$
26:     **end if**
27:     $f_{A_3}(K_{B_2}) \xrightarrow{K_{A_3}^S} K_{B_2}$
28:     $f_{B_2}(f_H(M_i)) \xrightarrow{K_{B_2}} f_H(M_i)$
29:     aggregate $\sum_{i=0}^N(f_H(M_i)) \rightarrow \overline{(f_H(M_i))}$
30: **end while**
31: Model Provider obtains the $M$
```

3. PLATFORM IMPLEMENTATION

In this section, the details of the platform implementation will be discussed. The architecture of the platform is shown in Figure 2.

As the implementation framework of Figure 1, this platform provides the front-end pages for user interaction. Different actors log in different front-end pages and have different functions. The backend is responsible for managing actors, distributing requests, encryption protocol, and so on. Log alarm module plays a role for notification hub. The platform uses IPFS [3] to implement DFS functions. Additionally, Ethereum will be used as a Blockchain Server. To get unified management, we adopt docker as the training environment for the computation provider.

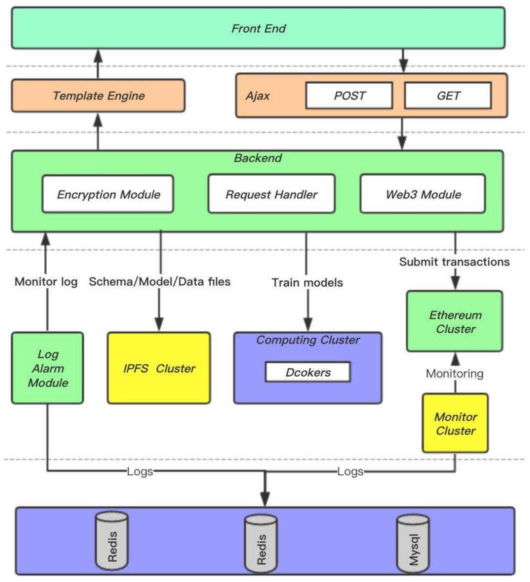


Figure 2. Distributed Training Platform.

3.1 Web Interface

The web interface is responsible for user interaction and initiates requests for the web backend. In this module, we have used the Vue.js framework to quickly develop front-end pages. In these well-interactive web pages, users can log in, submit transactions to Ethereum and manage Ethereum wallets which are implemented by web3.js:

1. On the model provider's web page, there are a number of functions: applying the HE provided in the encryption protocol to encrypt the model locally, submitting the smart contract to Ethereum, uploading and downloading files in IPFS, choosing model aggregation mode and so on.

2. On the data provider's web page, the platform provides several functions: broadcasting data schema, downloading the HE public key from the model provider to encrypt data, applying the AES/RSA algorithms, uploading/downloading file and so on and son on.

3. We can broadcast computational specification, apply the encryption algorithm, print and record training logs, download/upload file and execute train tasks on the computation provider's web page and so on.

3.2 Local Web Backend

We build a simple, reliable and efficient backend in Golang, with the following modules:

- Request handler module

With net/http package, we create HandleFunc registers the handler function for the given pattern in the DefaultServeMux. Every unique pattern will map to different handler functions, such as uploading files to IPFS cluster.

- Encryption module

Encryption module will manage the encryption algorithm and pass this encryption algorithm to the corresponding actor to generate the encryption key locally.

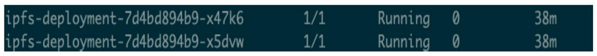
- Web3 module

We use Web3.js to interact with a remote Ethereum cluster, using a HTTP connection.

3.3 Docker & K8S Implementation

Docker is a platform for developers to develop, deploy, and run applications with containers. The container is responsible for creating and managing containers to train the distributed model by implementing docker and k8s. One container corresponds to one computation provider's training environment. Based on the above modules we will disclose the details of the platform implementation in this section.

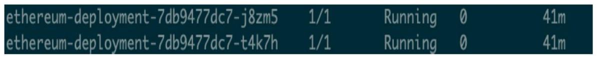
1. IPFS Cluster: we package the IPFS service into a single Docker image and used the Docker image to publish two Pods on the k8s cluster. The two Pods constitute an IPFS private cluster with two nodes. The k8s create Service object to exposes the IP and port of the IPFS cluster to facilitate the platform to use. The following figure shows the list of Pods where the first column is the names of two pods, the second column is the number of dockers in the pod, the third column is the running status, and the fourth column is the last column is the time of running.



```
ipfs-deployment-7d4bd894b9-x47k6        1/1       Running   0          38m
ipfs-deployment-7d4bd894b9-x5dvw        1/1       Running   0          38m
```

Figure 3. The List of Pod Columns of IPFS Nodes.

2. Ethereum Cluster: Similarly, we build a private chain of Ethereum with two Pods on the k8s cluster, and deployed our smart contract on it. PoW is used as its consensus mechanism. We adopt Solidity as the language for smart contracts. Gas cost for the smart contract is shown in Table I. The smart contract interface code is detailed in the appendix A. At the same time, the Service exposes the interface of Ethereum to facilitate the platform to use. The data provider and the model provider can use the web3 service locally. They packaged the IPFS hash of the data file and the IPFS hash of the model file into a transaction and submitted it to Ethereum. The Ethereum cluster's figure is shown the same layout as the figure 3.



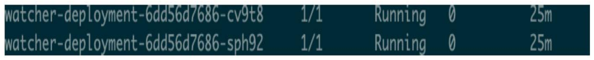
```
ethereum-deployment-7db9477dc7-j8zm5    1/1       Running   0          41m
ethereum-deployment-7db9477dc7-t4k7h    1/1       Running   0          41m
```

Figure 4. The List of Pod Columns of Ethereum Cluster.

Table 1. Gas Cost for Smart Contract Functions.

Function	Gas
authority()	728
cancel()	877
dataConsumer()	816
dataProvider()	706
executeForcefulRefund()	commonly 21000
executeForcefulSettle()	commonly 21000
expirationTime()	724
initTransaction(address,bytes32,bytes32)	81112
ipfsConsumerPubKeyAddress()	500
ipfsDataAddress()	544
ipfsEncryptedAesKeyAddress()	632
ipfsSchemaAddress()	654
provideData(bytes32,bytes32)	61061
requestForcefulRefund()	20835
requestForcefulSettle()	21100
requestedForcefulRefund()	618
requestedForcefulSettle()	690
verifiedData()	commonly 21000

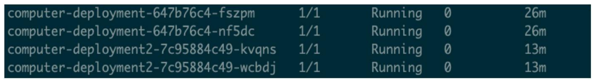
3. Monitoring Cluster: we also package the code of the listening blockchain transaction written in Go language into an image, and use the image to build a monitoring cluster with two Pods in the k8s cluster. The monitoring cluster will constantly monitor the transaction information in the Ethereum private chain. When the transaction containing the operation information is monitored, it parses the operation information and calls the interface exposed by the corresponding operation cluster according to the operator address included in the training information to transmit the operation information to the operation cluster. The monitor cluster's figure is shown the same layout as the figure 3.



```
watcher-deployment-6dd56d7686-cv9t8     1/1       Running   0          25m
watcher-deployment-6dd56d7686-sph92     1/1       Running   0          25m
```

Figure 5. The List of Pod Columns of the Monitor Cluster.

4. Computing Cluster: we package the training program based on the go language into an image, and use the image to build two computing clusters with two Pods on the k8s cluster. After the Pod runs well, the Service exposes the interfaces of the two computing clusters to serve the training of the model. The two computing clusters' figure is shown the same layout as the figure 3.



```
computer-deployment-647b76c4-fszpm      1/1       Running   0          26m
computer-deployment-647b76c4-nf5dc      1/1       Running   0          26m
computer-deployment2-7c95884c49-kvqns   1/1       Running   0          13m
computer-deployment2-7c95884c49-wcbdj   1/1       Running   0          13m
```

Figure 6. The Pod List of the Two Computing Clusters.

3.4 Experiments and Results

All the experiments are based on the server with 128G memory, 128-core Intel Xeon CPUs and two v100 NVIDIA GPUs. There are two ways that we train the distributed model. We take the MNIST dataset as an example and use a simple fully connected network. The training step is shown in Figure 7.

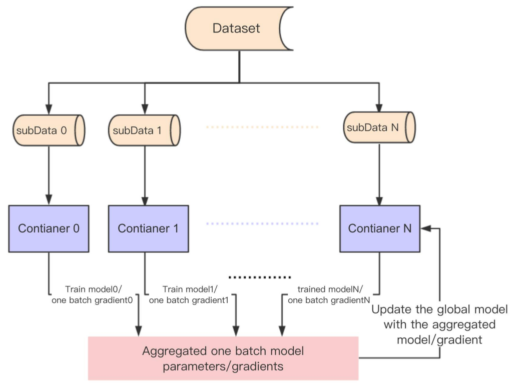

Dataset
subData 0
subData 1
subData N
Contianer 0
Contianer 1
Contianer N
Train model0/ one batch gradient0
Train model1/ one batch gradient1
trained modelN/ one batch gradientN
Update the global model with the aggregated model/gradient
Aggregated one batch model parameters/gradients

Figure 7. Model Aggregation Process.

We build up 1 model provider node, 5 data provider nodes and 5 computation provider nodes on the platform.

1. Model parameters aggregation.
On the data provider's web page.
The MNIST dataset is divided into 5 parts randomly to simulate the multiple decentralized datasets [5]. Every data provider can obtain one of these datasets as their own raw data.

On the model provider's web page.
We broadcast a smart contract and initialize a model M_0. Additionally, the model provider aggregates all trained parameters by using FederatedAveraging algorith [10]. And we choose to aggregate parameters on the model providmer's web page.

On the computation provider's web page.
The platform establishes a unified training environment and one computation provider can build up a docker which downloads the dataset and the model M_0.

Then all the actors will cooperate with each other to complete the training task according to the workflow in Section 2.3.
Finally, when the global model is converged, we will get the trained model M on the model provider's page and validate this

model's accuracy on the test dataset.

In the following Figure 8, the green line is the accuracy of main model which is aggregated by other containers' model parameters. The red line represents the accuracy of the traditional_model. The red line converges one step ahead of the green line, but their final correct rate is very close. We compare the above method with the traditional training method. In the traditional method, the same model was implemented only with Numpy. The distributed training reaches the accuracy of 93.1% while the traditional training method reaches the accuracy of 95.5%.

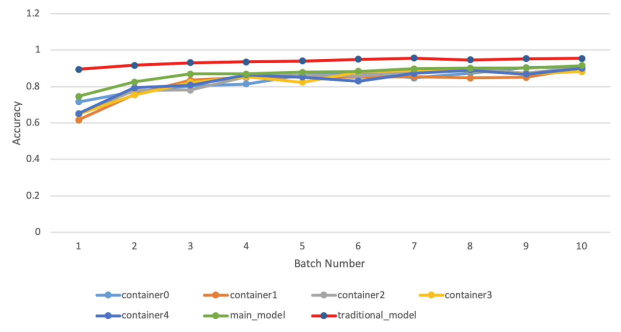


Figure 8. Accuracy Statistics of Parameters Aggregation Training Method.

2. Model gradients aggregation
We choose to aggregate gradients on the model provider's web page. The only difference is that when every docker trains the model and computes the gradients with one batch, they will send the gradients to the model provider for aggregation.

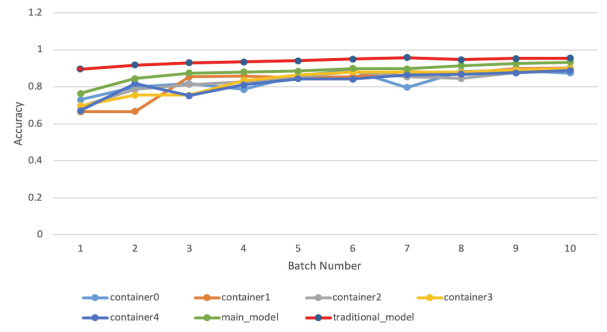


Figure 9. Accuracy Statistics of Gradients Aggregation Training Method.

In Figure 9, the red line converges one step ahead of the green line, but their final correct rate is very close. The main_model is aggregated by 5 containers' model gradients. As we see, they converged after 10 batches. What's more, the distributed training reaches the accuracy of 92.2% while the traditional training method reaches the accuracy of 95.5%.

4. DISCUSSIONS AND FUTURE WORK

In this paper, we have presented and implemented a novel distributed algorithm to train protected models with privacy-preserved data on Blockchain. Furthermore, we have proposed the redesigned FL with encryption algorithm by using HE, RSA and AES. To more successfully apply such a new system, our future work has the following challenges:

- Aggregation Algorithms

In this platform, we just use the simple FederatedAveraging algorithm for aggregating models. In the future, we hope to contribute other practical federated algorithms to our platform.

- Encryption Alternatives

In the current work presented, homomorphic encryption is used for training tasks. However, we found that the implementation of fully homomorphic encryption is a few orders of magnitude slower than operations on unencrypted data. Actually, MPC [7] and DP[8] can also work as alternatives to solve this issue.

- Blockchain Limitations

When the Blockchain is used in a real-world application, the scalability of the network (transactions per second) will become a key issue [18] [6]. "Sharding technology [12]" is one of the "chain expansion" solutions, which is considered as a solution to effectively solve the Blockchain throughput problem.

ACKNOWLEDGMENTS

This work is supported by Fundamental Research Funds for the Central Universities, Artificial Intelligence Research Foundation of Baidu Inc., Zhejiang University and Cybervein Joint Research Lab, Zhejiang Natural Science Foundation (LY19F020051, R19F020009, LZ17F020001), National Natural Science Foundation of China (61572431), Key R&D Program of Zhejiang Province (2018C01006), Program of China Knowledge Center for Engineering Sciences and Technology, Program of ZJU and Tongdun Joint Research Lab, Program of ZJU and Horizon Robotics Joint Research Lab, Joint Research Program of ZJU and Hikvision Research Institute, and Major Scientific Research Project of Zhejiang Lab (No. 2018EC0ZX01-1).

REFERENCES

[1] Bellare, M., Desai, A., Jokipii, E., & Rogaway, P. (1997, October). *A concrete security treatment of symmetric encryption*. In Proceedings 38th Annual Symposium on Foundations of Computer Science (pp. 394-403). IEEE.

[2] Bellare, M., & Rogaway, P. (1994, May). Optimal asymmetric encryption. *In Workshop on the Theory and Application of of Cryptographic Techniques* (pp. 92-111). Springer, Berlin, Heidelberg.

[3] Benet, J. (2014). Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561.

[4] Bengio, Y. (2009). *Learning deep architectures for AI. Foundations and trends® in Machine Learning*, 2(1), 1-127.

[5] Caldas, S., Wu, P., Li, T, Konečný, J., McMahan, H. B, Smith, V., & Talwalkar, A. (2018). Leaf: A benchmark for federated settings. arXiv preprint arXiv:1812.01097.

[6] Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., & Song, D. (2016, February). On scaling decentralized blockchains. In International Conference on Financial Cryptography and Data Security (pp. 106-125). Springer, Berlin, Heidelberg.

[7] Du, W., & Atallah, M. J. (2001, September). *Secure multi-party computation problems and their applications*: a review and open problems. In Proceedings of the 2001 workshop on New security paradigms (pp. 13-22). ACM.

[8] Dwork, C. (2011). Differential privacy. Encyclopedia of Cryptography and Security, 338-340.

[9] Gentry, C., & Boneh, D. (2009). *A fully homomorphic encryption scheme* (Vol. 20, No. 09). Stanford: Stanford University.

[10] Konečný, J., McMahan, H. B., Yu, F. X, Richtárik, P, Suresh, A. T, & Bacon, D. (2016). *Federated learning: Strategies for improving communication efficiency*. arXiv preprint arXiv:1610.05492. .

[11] LeCun, Y., Bengio, Y., & Hinton, G. (2015). *Deep learning. nature*, 521(7553), 436.

[12] Luu, L., Narayanan, V., Zheng, C., Baweja, K, Gilbert, S, & Saxena, P. (2016, October). *A secure sharding protocol for open blockchains*. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 17-30). ACM.

[13] McMahan, Brendan, and Daniel Ramage. (2017). "Federated learning: Collaborative machine learning without centralized training data." Google Research Blog.

[14] Shen, Y., Guo, B., Shen, Y., Duan, X., Dong, X., & Zhang, H. (2016). A pricing model for big personal data. *Tsinghua Science and Technology*, 21(5), 482-490.

[15] Swan, M. (2015). Blockchain: Blueprint for a new economy. " O'Reilly Media, Inc.".

[16] Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, 151(2014), 1-32.

[17] Wu, C., Xiao, J., Huang, G., & Wu, F. (2019). Galaxy Learning-A Position Paper. arXiv preprint arXiv:1905.00753. Date:June

[18] Zheng, Z., Xie, S., Dai, H. N., Chen, X., & Wang, H. (2018). Blockchain challenges and opportunities: *A survey. International Journal of Web and Grid Services*, 14(4), 352-375.

APPENDIX A SMART CONTRACT

In this section we provide the function signatures and main variables of our smart contract.

```
pragma solidity ^0.4.25;
contract DataTransfer {
  address public authority = < Address 0x... > ;
  uint duration = < Duration here > ;
  address public dataConsumer = msg.sender;
  address public dataProvider;
  bytes32 public ipfsSchemaAddress;
  bytes32 public ipfsConsumerPubKeyAddress;
  bytes32 public ipfsEncryptedAesKeyAddress;
  bytes32 public ipfsDataAddress;
  uint public expirationTime;
  bool public requestedForcefulRefund = false;
  bool public requestedForcefulSettle = false;

  modifier onlyBy(address _account) {}
  modifier onlyAfter(uint _time) {}
  modifier onlyBefore(uint _time) {}
  modifier etherProvided() {}
  modifier dataProvidedByProvider(bool
    ↪ _isProvided) {}

  function initTransaction(address
    ↪ _dataProvider, bytes32
    ↪ _ipfsSchemaAddress, bytes32
    ↪ _ipfsConsumerPubKeyAddress) public
    ↪ payable onlyBy(dataConsumer)
    ↪ etherProvided {}

  function provideData(bytes32
    ↪ _ipfsEncryptedAesKeyAddress, bytes32
    ↪ _ipfsDataAddress) public
    ↪ onlyBy(dataProvider)
    ↪ onlyBefore(expirationTime) {}

  function cancel() public onlyBy(dataConsumer)
    ↪ dataProvidedByProvider(false)
    ↪ onlyAfter(expirationTime) {}

  function verifiedData() public
    ↪ onlyBy(dataConsumer)
    ↪ dataProvidedByProvider(true) {}

  function requestForcefulRefund() public
    ↪ onlyBy(dataConsumer) {}

  function requestForcefulSettle() public
    ↪ onlyBy(dataProvider)
    ↪ dataProvidedByProvider(true) {}

  function executeForcefulRefund() public
    ↪ onlyBy(authority) {}

  function executeForcefulSettle() public
    ↪ onlyBy(authority) {}
}
```

Listing 1. Smart contract defined in transfer.sol. Variable ‘authority’ is the Ethereum address of the Authority. Variable ‘duration’ is the time duration permitted to perform the trade and call the corresponding functions. Both ‘authority’ and ‘duration’ need to be pre-defined before computation Conference Location:El Paso, Texas USA