

# DemoFL: Democratic Federated Learning Based on Multi-party Computation and Consensus Algorithm

Zhaoyang Han, Chunpeng Ge, Zhe Liu\*, and Chuan Ma

**Abstract**—Federated Learning (FL) is a framework based on which numerous parties can train a machine learning model cooperatively without leaking information about their data. However, attackers can reconstruct FL parties' data through the leaked parameters. Therefore, secure aggregation is critical to ensure privacy-preserving in FL frameworks. The current secure aggregation algorithms in FL rely on the parties to establish a secure channel between each other, which leads to a heavy communication overhead. Moreover, establishing secure channels between parties in FL may be unrealistic in some scenarios.

In this paper, we propose Democratic Federated Learning (DemoFL), which is a privacy-preserving FL framework based on secure multi-party computation (MPC) and simplified consensus algorithms. Our method adopts secret sharing to hide local model parameters, and runs a consensus algorithm to ensure that the communication is reliable and robust. DemoFL utilizes a hierarchical structure to reduce the complexity from  $O(n^2)$  to approximately  $O(n)$ , and it realizes high security and robustness without reducing accuracy. We also conducted a performance evaluation to demonstrate the practicality and efficiency of our proposed framework.

**Index Terms**—Federated Learning, Secure Aggregation, Machine Learning, Multi-party Computation, Consensus algorithm

## I. INTRODUCTION

Mobile devices in daily life are equipped with increasing powerful computing and storage abilities, which enables individual devices to accumulate enormous valuable information. It facilitates a huge number of rising technologies such as edge computing and smart home. Meanwhile, the accumulated data in users' devices can be used to train models for various practical purposes due to the flourishing of machine learning. In traditional machine learning frameworks, data needs to be gathered in a central server in order to execute the learning process. However, most data collected by mobile devices is sensitive. Users usually refuse to send their private data to others, such as a learning center, which impedes the development of distributed learning among common users.

Since the computing ability of mobile devices is powerful enough to run small-scale machine learning tasks, FL [1], which is a distributed machine learning framework, was proposed to address the privacy problem in learning tasks. FL

is developing rapidly and has increasing number of applications. For example, Google's keyboard query suggestions [2] project was an effective application of FL. Figure 1 illustrates the structure of FL. In each round of FL, parties receive a global model from the server and train their models based on their own data respectively. Afterwards, each party sends the parameters of their models to the server while the server runs a particular aggregation algorithm to compute the global model based on these parameters. In such frameworks, users do not need to send their data to the learning server, and thus privacy of participants are protected.

Though FL can protect users' privacy, attackers can eavesdrop a party's parameter and then establish the inference attack [3], [4], [5]. Additionally, as the server is provided by an untrusted third party [5], it can easily obtain all participants' parameters in the aggregation process and conduct the inference attack for the reason that in the conventional FL framework [1], parameters are directly sent to the server, which makes the server gathered all information easily. Therefore, sending parameters to the server directly faces the threat of inference attacks. This kind of attack is quite severe as it can be established by any malicious or semi-honest participant in the FL process. Therefore, how to conduct joint learning without leaking parameters to others comes into focus.

Secure aggregation protocols [6] enable a group of parties who have private information to compute a function of these private without revealing them. There are many works paid attention to secure aggregation for better solutions [6], [7], [8], [9], [10]. For example, Shi *et al.* [6] utilized homomorphic encryption (HE) methods to achieve secure addition. With the data encrypted, attackers cannot obtain any useful information from leaked messages. Therefore, secure aggregation is suitable for FL, which helps to protect intermediate models' parameters. A trivial solution is to employ HE to implement secure aggregation, however, HE algorithms suffer from low efficiency which is hardly acceptable in FL [11]. Differential Privacy (DP) is another feasible method to implement secure aggregation whereas DP based frameworks add noises to the parameters to deceive the attackers and these noises also have an influence on the learning result and reduce accuracy. Moreover, Blockchain-based methods [12], [13], [14] are also promising, and the generally used consensus algorithms in them can be inspiring.

One may think that the multi-party computation (MPC) [15] technique can be leveraged to implement secure aggregation directly. However, most MPC protocols rely on the prerequisite that all participants are able to communicate with each other. In practice, common users of a FL task are unknown to each

Zhaoyang Han, Chunpeng Ge, and Zhe Liu are with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, 210016, China (e-mail: nuaahanzy@gmail.com; gecp@nuaa.edu.cn; sduliuzhe@gmail.com).

Chuan Ma is with the School of Electrical and Optical Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: chuan.ma@njust.edu.cn).

\*Corresponding author: Zhe Liu.

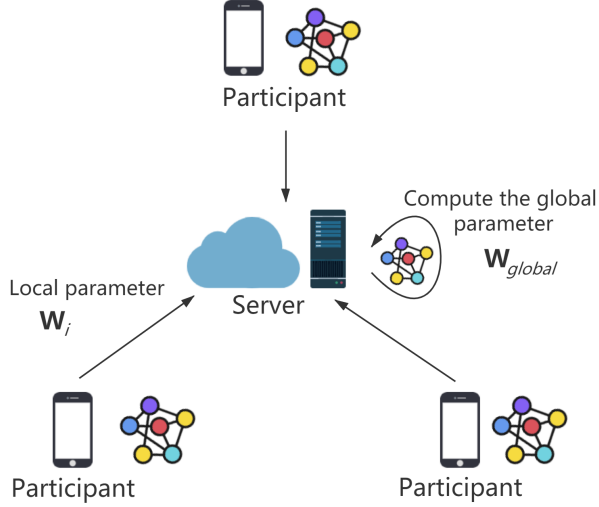


Fig. 1. The structure of FL. Participants train local models respectively and send their parameters to the server for aggregation.

other, which means one party cannot communicate with another directly. This is a severe obstacle between MPC and FL, therefore, there are some works about realizing P2P communication in FL. For example, Bonawitz *et al.* [16] utilized Diffie-Hellman key exchange to generate pairwise secret masks, however, this method has a dissatisfactory efficiency. The high communication overhead during MPC demands prompt solution. [17] also analyzed the MPC process in FL and tried to reduce the overheads by introducing a method based on aggregation committee, which still has a high price on set-up. In addition, Bonawitz's work also pointed out that the robustness of FL frameworks is also significant because most MPC protocols will fail if there are some participants dropping out, and it usually costs a lot to recover from situations where several nodes are dropping out or crashed. In practice, mobile devices' loss of communication happens frequently, which may cause breakdowns and delays. In summary, employing traditional MPC methods in a system with a large number of users is faced with a problem with efficiency and instability.

#### A. Motivation

We aim to design a federated learning framework which has these properties:

- 1) It protects users' privacy by employing MPC, while solving the problem that participants are unknown to each other.
- 2) It has a low communication overhead, avoiding constructing overmuch pairwise connections.
- 3) It is secure in semi-honest environments, where all users and the server may eavesdrop information.
- 4) It is highly robust, which has the capability to handle emergencies where several participants may lose connection.
- 5) It has no negative influence on the quality of the trained model.

#### B. Our contribution

- 1) We propose Democratic Federated Learning (DemoFL), a novel FL framework that utilizes MPC to protect users' privacy. DemoFL employs a smart additive secret sharing protocol to realize MPC, which helps to hide intermediate parameters.
- 2) DemoFL utilizes hierarchical structure to improve efficiency: our model first selects some leaders, who will construct secure communication with other parties. Afterwards, a party only needs to exchange information with the leaders. The leaders will send the received information to the server, who helps to forward the information to the corresponding destinations. Appointing leaders can reduce the need of communications compared to selecting leaders based on consensus algorithms.
- 3) We take advantage of consensus algorithm to achieve high robustness. Consensus algorithm helps to handle unexpected situations where a leader node or a common user is crashed. It makes all parties come to an agreement democratically and rapidly when such situations happen.
- 4) We conducted a series of experiments and verified that DemoFL has satisfactory efficiency and robustness without reducing accuracy.

#### C. Roadmap

In Section II we introduce the background of knowledge and some definitions. Next, we introduce related work and some platforms of federated learning in Section III. Section IV detailedly illustrates our proposed framework including the attack model. Evaluations for efficiency and security are stated in Section V, followed with experimental results in Section VI. Finally, we give the conclusion and future expectations in Section VII.

## II. BACKGROUND

In this section, we briefly introduce the concepts of key exchange, Federated Learning, multi-party computation and Consensus Algorithms, which are used in our proposed framework.

#### A. Key Exchange

Key exchange protocols allow several parties to share secret keys under unsafe conditions. Diffie-Hellman [18] (DH) is a prestigious key exchange protocol. We introduce how DH protocol helps two parties to achieve agreement: suppose  $a$  and  $b$  want to obtain a secret key by means of DH. First they select a group  $G$  of order  $q$ , and randomly choose a number  $x_a$  and  $x_b$  respectively. Then  $a$  calculates  $g_a = g^{x_a}$ , where  $g$  is a generator of  $G$ , and  $b$  calculates  $g_b = g^{x_b}$ .  $a$  sends  $g_a$  to  $b$ , and  $b$  sends  $g_b$  to  $a$ . Finally they can calculate the shared secret  $s_{ab}$  respectively:

$$s_{ab} = g_b^{x_a} = g_a^{x_b} = g^{x_a x_b}$$

In this scheme, only  $g_a$  and  $g_b$  are sent under an unsafe condition. Attackers cannot infer  $s_{ab}$  from  $g_a$  and  $g_b$ , therefore

$a$  and  $b$  can communicate privately by means of  $s_{ab}$ . DH is lightweight and efficient, and it can be expanded to multi-party versions easily in order to enable more parties to share pairwise keys.

### B. Federated Learning

FL requires a number of users to jointly train a model. It usually runs for rounds. In each round, each user will train the model based on its data and get the model's parameter. Denote the  $i$ -th user's local parameter in the  $t$ -th round by  $W_i^t$ . When all users have trained their model in the  $t$ -th round, an aggregation algorithm  $Agg$  will be called to compute the global model's parameter  $Agg(W_1^t, W_2^t, \dots, W_n^t)$ , where  $n$  is the total number of users. For example, weighted average was adopted as  $Agg$  in FedAvg [1]. FL not only helps to protect users' privacy but also deals with the "data in form of isolated islands" problem for companies or institutions. Yang *et al.* also categorized FL into horizontal, vertical and hybrid styles based on the fact that whether the data shares the same feature space or entities [19].

Federated Learning can be generalized to two work environments:

- 1) **Among-institutions:** In this situation, FL is usually used to help companies or other institutions solving the "data in form of isolated islands" problem. Generally, companies and institutions are in equal status in an FL framework. Therefore, we can suppose one party can communicate with any other one privately, i.e., P2P is already enabled in this environment.
- 2) **Server based:** In this case a company or institution adopts FL to train a model based on their users' data while protecting their privacy. An FL party is a common user, which means users' communication usually needs to go through the server. With an honest-but-curious server, all information may be eavesdropped by it. Therefore, the problem with confidence is severe in such environments.

Figure 2 illustrates the structures of two environments. Executing MPC protocols and consensus algorithms are at a low price in the first environment. In the second environment, if local parameters need to be protected, we need to run some key-exchange protocols first to construct secure channels among users. One of our contribution is that we solved the problem in the second environment by building connections selectively and efficiently.

### C. Multi-party Computation

Secure MPC is a branch of cryptography which enables several parties to compute a particular function without leaking their own data (inputs). Suppose there are  $n$  parties employing MPC to compute a function  $F$ , and the  $i^{th}$  party has its parameter  $A_i$ . Their goal is to compute  $R = F(A_1, A_2, \dots, A_n)$ . MPC has the feature that participants can only obtain  $R$  from the process and the party  $P_i$  has no idea about parameter  $A_j (j \neq i)$ . This feature fits the aggregation algorithm in FL greatly.

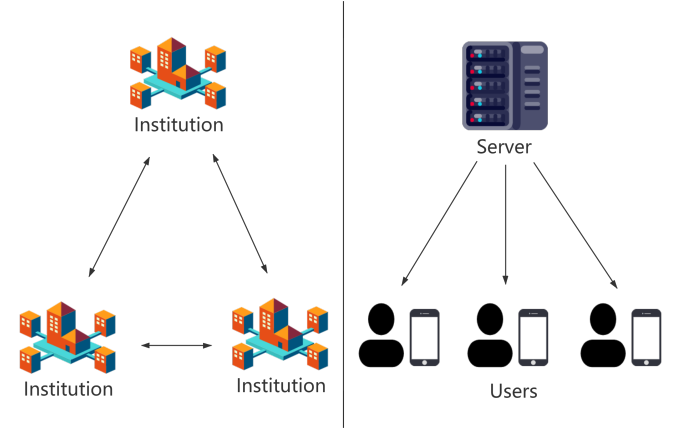


Fig. 2. The structures of two work environments in FL. The left is the "Among-institutions" model where institutions can communicate with others directly, and the right is "Server based" model where parties exchange information in virtue of the server.

It was first proposed by Yao to solve the millionaire problem [15]. Most MPC protocols depend on two cryptography technologies: secret sharing [20] and oblivious transfer [21]. MPC can be implemented with garbled circuits, multi-party circuit-based protocols or hybrid methods [22]. It also benefits from fully homomorphic encryption (FHE) algorithms. Garbled circuits and FHE suffer from complicated design and poor efficiency. Therefore, secret sharing methods are more favored to solve the privacy-preserving problem in FL. SPDZ [23] (speedz) is a practical and secure MPC protocol introduced by Damgard *et al.* It supports addition and multiplication by means of the triples [24], which are generated by somewhat homomorphic encryption (SHE). Our method does not require multiplication and hence we do not need to generate the triples choose to use simple additive MPC.

A secret sharing scheme involves a secret  $s$ , a set of  $n$  parties, and a collection  $A$  of subsets of parties. Each party has its share of  $s$ . The secret sharing scheme ensures any subset in  $A$  can reconstruct  $s$  [25]. By means of secret sharing, we can implement secure addition among parties, which is essential in our framework.

### D. Consensus Algorithms

In a distributed or multi-party system, there is always a problem with consensus, i.e., in such systems parties always need to achieve agreement on a certain value. This could be difficult without any strategy because different parties may be in different statuses and have multifarious matters. Consensus algorithms are adopted to address such problems. It is widely used in blockchain and various famous areas.

Paxos was the first consensus algorithm introduced by Lamport [26]. It is used in a lot of famous projects such as Ceph [27]. Raft is a modification of Paxos which is more industrial-friendly [28]. It contains two phases: leader election and log replication. Parties can achieve agreements based on leaders. Considering that FL model is a semi-decentralized system, our framework can utilize Raft algorithm

to select several leaders, based on which MPC protocols can be executed efficiently and robustly.

### III. RELATED WORK

HE based solutions are intuitively effective to solve aggregation problems, and there are many researchers tried to reduce the overhead caused by HE. Stephen *et al.* [29] designed an additively homomorphic encryption method for federated learning in 2017. However, the encryption method is complicated that it does not reach high efficiency. The state-of-the-art HE method was proposed by Zhang and Li [30]. It reduced the encryption overhead greatly at the cost of trivial loss of accuracy. However, it still costs much time on computation compared to MPC based and DP based methods.

DP based solutions have better performance than HE based solutions, while Wei *et al.* [31] has indicated that there is a tradeoff between the performance and security levels, which means it needs numerous adjustments to adopt DP methods efficiently. Robin *et al.* [32] tested DP in FL in 2017 and the result showed that DP's influence on accuracy is untrivial. Bayesian differential privacy [33] was proposed in 2019. It considered the probability and distribution of data and is effective for machine learning models whose data are often restricted to a particular type. While it may be inefficient in vertical federated learning situations. Whereas all these methods did not prove DP does not impact the accuracy in other more complicated and large-scale models, which means it is different to find a universal DP method for all machine learning models.

MPC based methods have the least computation cost but require more communications. A typical MPC federated learning model is implemented by Google [16]. It requires pairwise key exchange among all clients, which results in enormous overhead on communication. It also provides robustness by means of mask mechanism with some random numbers. However, these mechanisms bring about more overhead. Other researches present hybrid methods [34], [35]. These hybrid methods combine MPC with either HE or DP and adjust the tradeoff between computation and communication.

Federated learning is more and more practical nowadays and there are already many FL platforms: FATE [36] is an open-source federated learning project proposed by Webank's AI Department. It adopts both MPC and HE to implement secure aggregation, while it is still absorbing state-of-the-art methods for privacy-preserving. Pysyft [37] is another open-source federated learning framework presented by OpenMined. It is based on Pytorch and offers HE, DP, and MPC as alternative methods to realize privacy-preserving. However, the cost of time of Pysyft is dozens of times than pure Pytorch, which indicates that privacy-preserving methods of current federated learning platforms need to be improved.

### IV. DEMOCRATIC FEDERATED LEARNING

In this section, we introduce the attack model and the detailed design of the proposed framework Democratic Federated Learning framework. At first definitions of some symbols are listed below:

- **N**: The total number of parties in FL system, and the  $i^{th}$  party is marked as  $P_i$ .
- **S**: The server, which is the host of federated learning and helps other parties to forward messages.
- **frac**: The fraction of users that will participate in the learning process each round.
- **n**: In each epoch,  $n$  parties will be chosen to train the model.  $n = N * frac$ .
- **N<sub>l</sub>**: Number of leaders selected from  $N$  parties.
- **L**: The set of leaders.
- **W**: The parameters of a machine learning model.  $W_i$  means the  $i^{th}$  party's local parameter, and  $W_{global}$  is the global model's parameter.
- **C**: The total number of training data.  $C = \sum_{i=1}^n C_i$ , where  $C_i$  is the number of training data of the  $i^{th}$  party  $P_i$ .

Our framework is based on FedAvg, which computes the weighted average of local parameters as the global parameter. In epoch  $t + 1$ , the server's target is to compute the following formula:

$$W_{global}^{t+1} = \sum_{i=1}^n \frac{C_i}{C} W_i^t \quad (1)$$

Although some researches protect  $C_i$  from semi-honest attackers by means of multi-party multiplication, leakage of  $C_i$  does not help attackers to reconstruct any data. In addition,  $C$  can also be aggregated based on multi-party addition since it only needs addition to compute  $C$  from all the components  $C_i, i \in \{1, 2, 3\}$ . Thus, we can set  $G_i = C_i * W_i$  to simplify the expression and computation, i.e., multi-party multiplication is unnecessary in this framework. Then the goal of each epoch is adjusted to:

$$W_{global}^{t+1} = \frac{\sum_{i=1}^n G_i^t}{C} \quad (2)$$

So far, we modeled the FedAvg algorithm to a form where only secure multi-party addition is needed to enable privacy-preserving. When the server obtains  $C$  and the sum of  $G_i$  by means of multi-party addition, it can compute  $W_{global}$  locally.

#### A. Attack Model

An attacker can be either a party or the server, and we assume the attackers is **honest-but-curious** (or semi-honest). An honest-but-curious attacker gathers the information it receives instead of deviating from protocols. The attacker's target is to reconstruct a user's data based on its parameter, which has a high probability of being eavesdropped by the attacker. If the attacker is the server, we can suppose all the communications are monitored by it. We also assume that there can be collusions among attackers, however, they do not have the authority and ability to cheat on some processes such as the election.

#### B. Framework Design

We take the server-based situation for example. Our framework consists of four processes: **leader election**, **key exchange**, **secure learning** and **reorganization**. In the leader

election phase, the parties elect several leaders who are responsible for the MPC protocols. In the key exchange phase, all parties build secure channels with the elected leaders, based on which a party can privately communicate with leaders without leaking any information to the server. In the secure learning process, the whole system executes MPC enabled federated learning. The reorganization process is activated when a node is crashed. E.g., if a leader is crashed, the reorganization process will generate a new leader as the replacement. The four processes are elaborated below:

1) **Leader election:** A server-based federated learning system is different from other joint-systems such as blockchains. Therefore, consensus algorithms can be adjusted more compatible with federated learning models. Our framework adopted a simplified version of Raft: since we have a server in the system, the leaders can be appointed by the “center” randomly instead of being elected by all participants, which can improve the performance greatly because it saves the time for users to achieve a consensus. This appointment does not destroy “democracy” because of the “leader-tenure” and “self-recommendation” regime, which will be introduced later.

At the very first, the server randomly selects several parties as leaders. The server generates a set  $L$  consisting of the identifiers of leaders. Then it sends  $L$  to all parties who will participate in the federated learning process. Those participants who are not leaders are called common users or common clients. Afterward, Key exchange protocols will be executed to construct secure channels between common parties and leaders. Note that the secure channels will be constructed between leaders with all  $N$  parties instead of  $n$  participants.

The technique of “Heartbeat” is used permanently to detect whether a leader is active: we set an interval  $t_h$ . The server will send a “heartbeat” to all leaders every  $t_h$  passed. If the server does not receive the response in time, it can determine that the leader is crashed and start the reorganization process.

To provide higher security, “leader-tenure” is adopted. I.e., the system will revoke one’s leadership regularly, and randomly select another common party as the new leader. This method can prevent collusion attacks where the server selects leaders dishonestly. Details will be discussed later in Section IV.

In the among-institutions environment, the election process is different because there is not a server helping to appoint leaders. Therefore, the leader election process is more Raft-like: first, the parties act as candidates and send “self-recommendation” to the others with different delays. Then, a party will vote for the first  $N_l$  parties that it has received “self-recommendation” from. The  $N_l$  parties with the most votes become leaders. The detailed algorithm can be referred to as Raft algorithm [28].

2) **Key exchange:** When the parties receive  $L$ , they will know who the leaders are. In order to communicate with the leaders privately, the parties need to run key-exchange protocols to share secret keys. We employ Diffie-Hellman protocol [18] to realize it.

In a server-based environment, a server can communicate with any clients while the parties are unfamiliar with each other. Therefore, the key exchange process should be con-

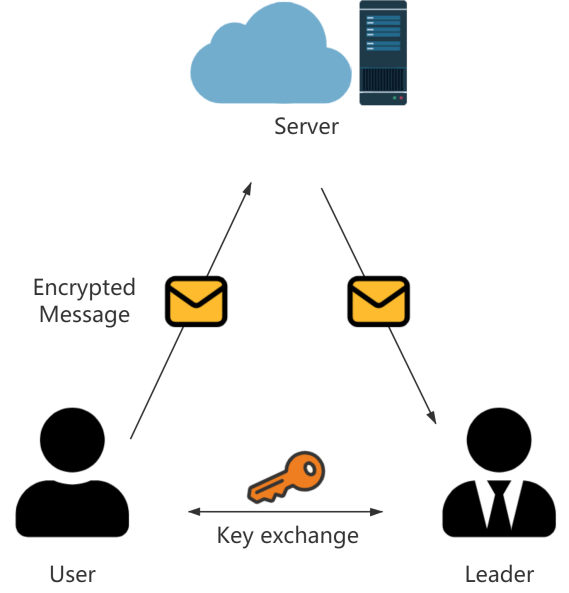


Fig. 3. The institution can communicate with all users directly while all the leader-user pair share a secret key used for private communication. The server helps to forward encrypted messages from which it can know nothing.

ducted in virtue of the server. In the premise that the server is honest-but-curious, parties can exchange keys in a “man-in-the-middle” scheme, where the “man” is the honest server. The server delivers information for users to help them complete DH protocol. This process is secure because information leaked to the server in key-exchange protocols is useless.

After the key exchange process, each party-leader pair, a party  $P_i$  with a leader  $L_j$  will share a secret key  $K_{ij}$ . By means of  $K_{ij}$  the party  $P_i$  and leader  $L_j$  can encrypt their data against the honest-but-curious attacker. The message  $m$  can be encrypted as  $Enc_{K_{ij}}(m)$ . Figure 3 illustrates the institution-leader-user structure. Normally, the institution and the common users only need to communicate with the leaders.

3) **Secure learning:** The aggregation algorithm is the same as FedAvg [1]. A secure multi-party addition protocol is used to realize privacy-preserving. There are  $N_l$  leaders in the system. Without loss of generality, we suppose  $N_l = 3$ . The flow chart of secure learning process is shown in Figure 4 and the process is illustrated as below:

- 1) All the selected parties train the model locally. And a party  $P_i$  will get a parameter  $G_i$ .
- 2) Each  $G_i$  is divided into 3 pieces  $G_i^1$ ,  $G_i^2$  and  $G_i^3$ , where  $G_i = G_i^1 + G_i^2 + G_i^3$ . Then  $G_i^j$  is sent to the  $j^{th}$  leader  $L_j$  respectively.
- 3) A leader  $L_j$  will record a set  $B_j$ , which consists of the parties that have sent messages to  $L_j$ . When all the parties’ parameters are sent or time is up, the leaders will send all  $B_j$  to the server and the server will compute the intersection as  $B$ . Then  $B$  is sent back to all leaders. Each leader reserve parameter shares according to  $B$  in case that some parties did not send parameter pieces to all leaders successfully. Parties not included in the

intersection are then removed.

- 4) Leader  $L_j$  then computes the sum of the parameters:

$$A_j = \sum_{i=1}^n G_i^j$$

Afterwards,  $A_j$  will be sent to the server.

- 5) The server computes:

$$W_{global} = \frac{A_1 + A_2 + A_3}{C} = \frac{\sum_{i=1}^n G_i}{C}$$

and it is the target of FedAvg algorithm.  $W_{global}$  is used to update the global model and the server will prepare for the next learning round.

$C$  can be obtained in the same manner: replace  $G_i$  with  $C_i$  and the server can get  $C$  privately. In fact,  $C_i$  and  $G_i$  are sent to leaders at the same time. Alternatively, each  $C_i$  can also be directly sent to the server without encrypted since it does not help attackers understanding the parameters. When the system decides to discard a party  $P_i$  due to  $B$ , it also needs to discard  $C_i$ , which will change the amount of training data  $C$ . The pseudocode of secure learning is illustrated in Algorithm IV-B3. Neither the server nor any leader can figure out what exactly a certain party's parameter is based on the information it can receive.

4) **Reorganization:** The reorganization process provides the robustness for the framework. Normally, the server will notice a leader's crash within a heartbeat cycle. Meanwhile, a party will report to the server if it finds a leader is crashed. After noticing a leader is crashed, the server sends "stop" signals to all parties, which will stop the learning processes. Then all parties will perform an election campaign according to the simplified consensus algorithm: they send "self-recommendation" signals to the server. The server will receive numerous signals and the first one to arrive will be chosen as the new leader.  $L$  will be updated and sent to all common parties. Finally, the learning process will be restarted.

### C. Among-institutions Model

Since secure P2P communication is already constructed, the key exchange process is removed in this situation. In addition, consensus algorithms can be executed naturally in such an environment as it is in a decentralized system where parties are equal. Therefore, leaders are elected by voting as the original Raft does. The leader-tenure and reorganization processes also vote for new leaders instead of getting leaders from a server.

## V. EFFICIENCY AND SECURITY EVALUATION

### A. Communications

As illustrated in Section III, our framework only adopts lightweight algorithms to enhance security. Therefore, the execution overhead can be ignored compared to the communication overhead. Notice that any communication in our framework goes through the server. E.g., if  $P_i$  wants to send encrypted message  $c$  to  $L_j$ ,  $c$  will be sent to server  $S$  first. Afterwards  $S$  will forward  $c$  to  $L_j$ . In general, our P2P communications are emulated with client/server communications.

### Algorithm 1 Secure Learning Algorithm

---

**Require:**  $G_i, L, K_i$  (the set of secret keys for party  $P_i$ ),  $E_{max}$  (the maximum epoch for federated learning)

```

1: function SPLIT_PARAMETER( $G_i$ )
2:    $r \leftarrow \text{RANDOM}(0, 0.5)$ 
3:    $G_i^1 \leftarrow G_i * r$ 
4:    $r \leftarrow \text{RANDOM}(0, 1 - r)$ 
5:    $G_i^2 \leftarrow G_i * r$ 
6:    $G_i^3 \leftarrow G_i - G_i^1 - G_i^2$ 
7:   return  $G_i^1, G_i^2, G_i^3$ 
8: end function
9:
10: function PARTY_SEND( $G_i, L, K_i$ )
11:    $G_i^1, G_i^2, G_i^3 \leftarrow \text{SPLIT\_PARAMETER}(G_i)$ 
12:   for  $j \in \text{indexes of } L$  do
13:      $E_{ij} \leftarrow \text{ENC}(K_{ij}, G_i^j)$ 
14:     if Sends  $E_{ij}$  to  $L_j$  not successfully then
15:       Report "Lj is crashed" to  $S$ 
16:     end if
17:   end for
18: end function
19:
20: function LEADER_SEND( $K_j$ )
21:    $B_j \leftarrow \emptyset$ 
22:   Receive  $E_{ij}$ s from all possible  $P_i$  and add  $i$  to  $B_j$ 
23:   Send  $B_j$  to  $S$ 
24:   Receive  $B$  from  $S$ 
25:   Remove  $E_{ij}$ s whose  $i$  is not included in  $B$ 
26:    $G_i^j \leftarrow \text{DEC}(K_{ij}, E_{ij})$  for each  $E_{ij}$ 
27:    $A_j \leftarrow \sum_{i=1}^n G_i^j$ 
28:   Sends  $A_j$  to  $S$ 
29: end function
30:
31: function SERVER_AGGREGATION( $e$ )
32:   if  $e \geq E_{max}$  then
33:     return True
34:   end if
35:   Receive  $A_j$ s from leaders
36:    $W_{global} \leftarrow \frac{A_1 + A_2 + A_3}{C}$ 
37:   UPDATE( $Model_{global}, W_{global}$ )
38:   // Next epoch
39:   Select  $n$  parties to participate in the next epoch
40:   Send  $W_{global}$  to these  $n$  parties
41:   return SERVER_AGGREGATION( $e+1$ )
42: end function

```

---

This method accelerates communication greatly because it costs much for two strangers to exchange messages directly. E.g., if two parties want to communicate directly, both of them need to store the addresses, confirm the "accept" signal after each message-exchange, *et al.* However, with a powerful server helping to forward, these things are no longer concerns for parties.

In each round, there are  $n$  common parties and  $N_l$  leaders. To analyze the communication overhead clearly and without loss of generality, we can suppose there is no common party that is also a leader at the same time. Common parties,



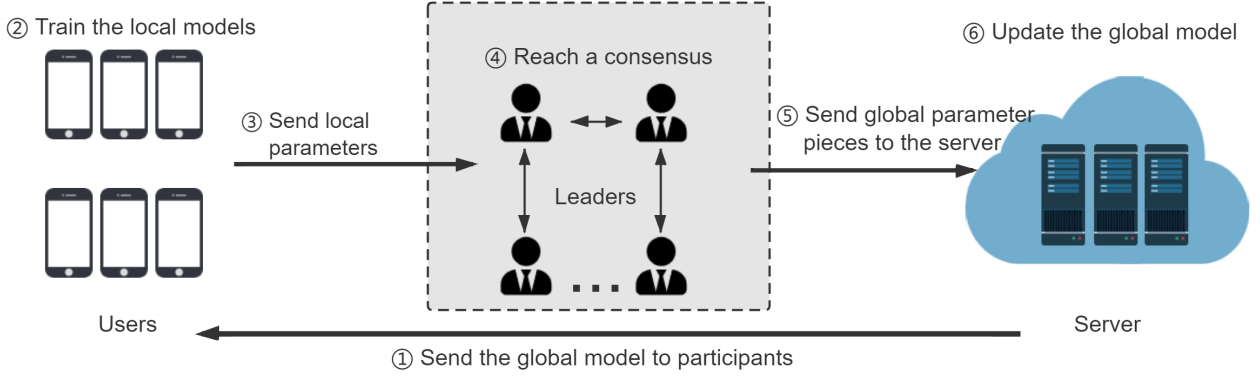


Fig. 4. The flow chart of the secure learning process. It is based on the fact that common users have already constructed secure communication channels with leaders. Note that a participant can be a leader and a common user at the same time, and the averaging was already done by leaders, therefore, the server only needs to summarize the global parameter pieces. After step ⑥, the process will enter the next epoch and restart from step ① if the current epoch has not reached the  $epoch_{max}$ .

leaders together with the server  $S$  are all parties requiring communication. Generally,  $N_l$  is a very small number such as 3, therefore we can treat it as a constant number. Our framework can be categorized into **set-up** and **epoch-learning** two phases. We analyze these two phases respectively:

- **Set-up:** All  $N - N_l$  parties need to construct secure channels with all  $N_l$  leaders. Suppose it needs  $D$  communications in each DH protocol, then the amount of communications of the set-up phase is  $D * (N - N_l) * N_l$ . The time complexity is  $O(N)$  based on the fact that  $D$  and  $N_l$  are small constant numbers. In the re-organization process,  $S$  chooses a new leader that needs to conduct DH protocols with all  $N - N_l$  parties, which is  $D * (N - N_l)$  communications. Therefore, a re-organization process also costs  $O(N)$  communications.
- **Epoch-learning:** In each epoch,  $S$  sends the current  $W_{global}$  to  $n$  parties, which cost  $n$  communications. Then every common party sends  $W_{ij}$  to  $N_l$  leaders respectively, which cost  $n * N_l$  communications. Finally leaders send  $A_j$ s to  $S$ , which cost  $N_l$  communications. Thus the total cost of one epoch is  $n + n * N_l + N_l$ , which is  $O(n)$ .

In the time complexity aspect, our framework does not result in higher overhead expect for an  $O(N)$  preprocessing compared to the original FedAvg algorithm. In the vertical aspect, our framework does not require more message-exchanges for any party-leader pair.

### B. Security Evaluation

The evaluation is based on the fact that our adversaries are all honest-but-curious. Since our framework is based on MPC researches [20], [38], [39], the security against message-leakage can be guaranteed. And the security of using MPC in FL has been proved by Zhu *et al.* [40] The riskiest threat is the collusion attack. Colluding with a common party has no contribution to an attack because it lets out nothing but the information about this common party, which belongs to the attacker side. Therefore, we only discuss situations that collusion among leaders.

Apparently, the attacker must collude with all leaders in order to reconstruct one party's parameter. Since the leaders are assigned by the server randomly, it is hardly possible for all attackers to be elected as leaders with the server being honest. However, it is necessary to discuss the situation where the server cheats to select leaders as its wish, though it does not deviate from the learning protocols. In such situations, the leaders are always selected by the unreliable server. To address this problem, we introduced leader-tenure in Section III, which forces the system to change leaders regularly. Since it needs all leaders' betrayal to attack successfully, the system only needs to change one leader regularly. Changing one leader is equivalent to a re-organization process, whose time cost is  $O(N)$ . Therefore, our framework has high security against collusive honest-but-curious attackers.

## VI. EXPERIMENTAL RESULTS

### A. Implementation

Our framework is implemented with Pytorch. We used AES-GCM-128 as the authenticated encryption algorithm as Bonawitz *et al.* did [16]. We adopted MNIST and CIFAR-10 as datasets, which are also used in the original FedAvg [1]. We trained a simple convolutional neural network for the classification task. Our experiments are carried on a PC with an Intel i7-8700 CPU (3.2GHz), 16 GB of RAM, and a GTX 1080 GPU. The model was executed in a single thread to facilitate comparing and analyzing. The optimizer was stochastic gradient descent (SGD) and the learning rate is 0.01. The *fraction* is set as 0.1 which means that 10% of clients will be chosen to carry on the training process in each epoch.

### B. Accuracy

Although our work does not modify the learning module compared to other federated learning frameworks, we still conducted a series of experiments to observe the accuracy. We compared FedAvg with our framework on both independently identically distribution (IID) data and non-IID data to verify

the effectivity. Since this experiment aims to prove the validity instead of high accuracy, the models were not trained to obtain high accuracy. We conducted experiments with MNIST in non-IID and CIFAR-10 in IID, and Figure 5 shows the result. With IID data, our framework obtains a similar performance compared with FedAvg. With non-IID data, the two systems did not match as well as they were with IID data when the *epoch* was small. However, they finally converged to the same stable scope with the same speed, which confirmed the differences were caused by biases. Therefore, our framework can achieve a same learning goal as FedAvg while providing privacy guarantees and system robustness.

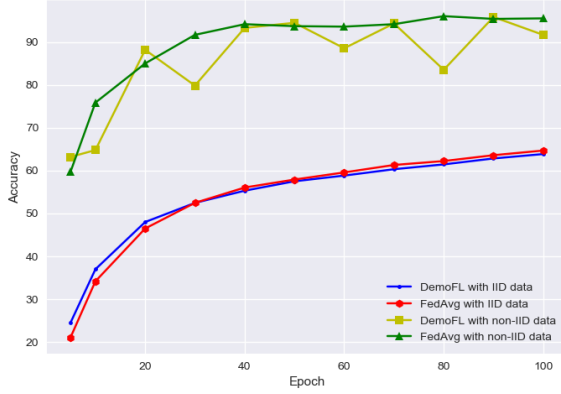


Fig. 5. The accuracy of FedAvg [1] and DemoFL with IID data from CIFAR-10, and non-IID data from MNIST. The horizontal axis *epoch* means the total rounds that the federated learning system has been trained for.

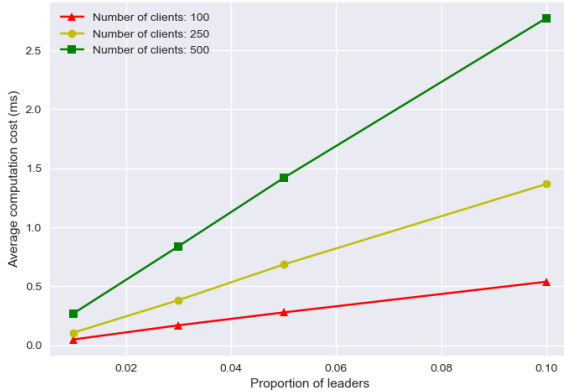


Fig. 6. The average time spent on computation in set-up process DemoFL VS different proportions of leaders.

### C. Set-up Overhead

In the set-up phase, our system selects several clients as leaders, and the number of leaders impacts the efficiency and a particular leader's load because a common user needs to build secure communication channels with all leaders. To discover the relation between number of leaders and the computaion

overhead, we set the number of clients in different levels, and conducted experiments with different number of leaders. Figure 6 shows the linear relation between each user's computation time spent on Diffie Hellman key-exchange protocol and the number of leaders. The overhead of time in set-up process is also very low because DemoFL has no complex calculations as Bonawitz *et al.* [16]'s secure aggregation scheme does. In Bonawitz's scheme, participants need to calculate secret keys for all other users, and generate t-out-of-n secret shares for the pseudorandom generator (PRG) seed and their secret keys. The comparison is illustrated in Figure 7. Since Bonawitz's scheme has a time complexity of  $O(n^2)$ , the computation overhead of it is much higher than DemoFL. Method of Kanagavelu *et al.* [17] performs the same as Bonawitz's scheme because they also have an  $O(n^2)$  set-up. Therefore, employing fewer leaders can help to improve efficiency, and in contrast, employing more leaders can be beneficial to security because a successful attack requires all leaders compromised.

Meanwhile, with more leaders, one common user needs to store more keys for leaders. Apparently, the relationship between the number of leaders and one user's storage overhead is also linear. However, in Bonawitz's scheme, a client needs to store two t-out-of-n secret shares for all others. The storage complexity of DemoFL and Bonawitz's scheme is  $O(N_l)$  and  $O(n)$  respectively, where  $N_l \ll n$ . Therefore, DemoFL has a very trivial set-up overhead on both time and storage, which is friendly to individual participants such as smart phones.

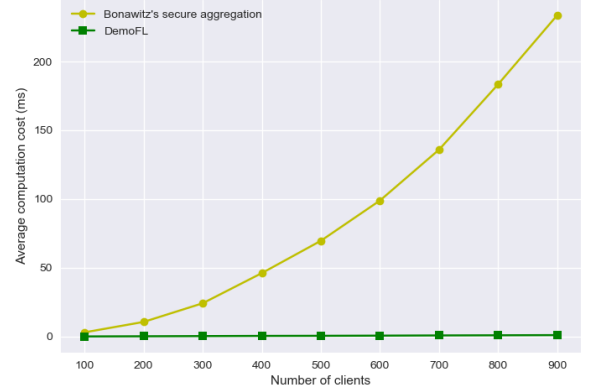


Fig. 7. The average time spent on computation in set-up process of Bonawitz's secure aggregation [16] and DemoFL VS the number of clients. The proportion of leaders is set to 3%.

### D. Efficiency

Our framework primarily addresses the server-based situation, where a participant's computing capability is far weaker than the server. And for the server, the overhead of computations such as MPC is insignificant compared to the communication overhead because of its powerful computing capability. Therefore, we carried out experiments to evaluate the computation overheads for participants only, and communication overheads for both participants and servers in DemoFL and Bonawitz's secure aggregation scheme. Experiments in this



subsection are with the assumption that there is no dropouts or crashes.

To evaluate the computation overheads, we fixed the number of clients to 100 and the proportion of leaders to 3, and tested the average running time of a single participant. In DemoFL, participants are divided into two categories: common users and leaders, and a participant can hold both identities at the same time. While in Bonawitz's secure aggregation scheme, all users are treated equally. The result is displayed in Figure 8. It suggests that participants of DemoFL has obvious advantage on computation overhead. The reason is that participants of Bonawitz's secure aggregation scheme need to carry out more encryptions/decryptions for unmasking.

Measuring the communication overheads with wall clock time is difficult because it has biases large enough to mislead the judgment. Therefore, we evaluate the communication overhead with the number of communications. We conducted experiments to count up the communications and tried to compare the overheads of DemoFL and Bonawitz's secure aggregation scheme. Since there are no dropouts or crashes, the number of communications in each epoch is fixed. Therefore, we computed the ratio of overheads of Bonawitz's secure aggregation scheme and DemoFL with the number of clients set as 100 and number of leaders as 3, and the result is 1.184. And if the number of leaders gets larger, this ratio will decrease towards 1. Since the number of leaders will never be set too large, the communication overhead of DemoFL is advantageous.

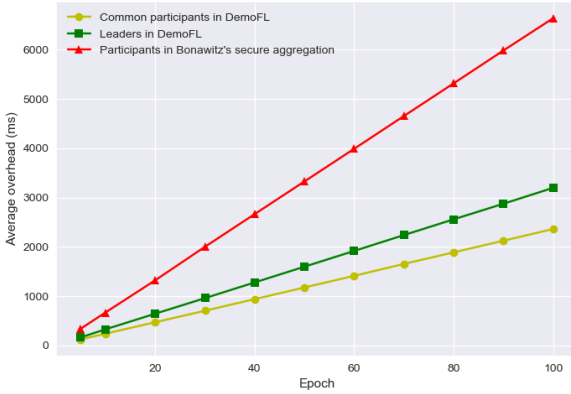


Fig. 8. The average running time per participant in the learning process VS epoch. Without dropouts and reorganization processes.

### E. Robustness

1) *Crash*: The robustness of DemoFL is mainly based on the reorganizing process, which happens when a leader is crashed. We discuss crashes for leaders because a crash of a common user is equivalent to a dropout. Crashes can hardly be completely avoided, therefore, we introduced *crash\_rate*, which is the possibility for each leader that it would crash during one epoch, to help to measure the robustness. Generally, we consider *crash\_rate* is quite small because compared

to dropouts, crashes rarely happen in nowadays smart devices/servers. Therefore we can consider *crash\_rate* would not be larger than 10%. Since a crash causes a reorganizing process, which helps the system by communications, we still adopt the number of communications to evaluate the extra overhead. We conducted several experiments on how *crash\_rate* impacts efficiency. First, we set the number of clients to 100 and the *crash\_rate* to 10%, which is the worst case. Then we counted up the number of communications at a situation without crashes. Finally we counted up communications at other situations with different number of clients and different number of leaders. We consider the ratio of communications in these situations with the origin one as their overheads, and the result is shown in Figure 9.

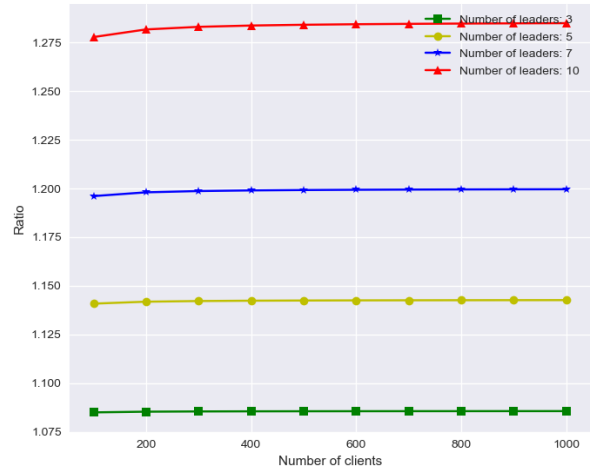


Fig. 9. The ratio of communications VS number of clients and leaders.

The number of clients hardly affects the overheads. In contrast, the number of leaders is very influential. Therefore, we suggest setting the number of leaders as smaller as possible. Generally, 3 or 5 could be sufficient choices.

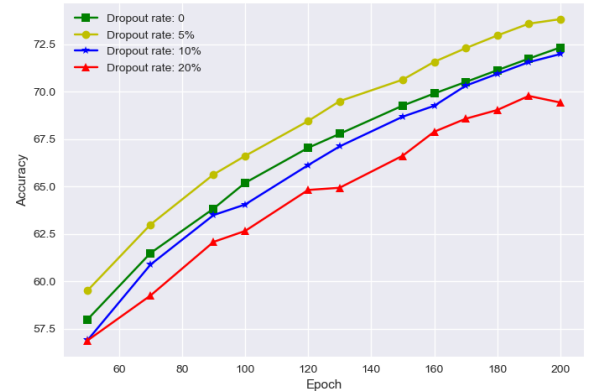


Fig. 10. The accuracy VS different *dropout\_rates*.

2) *Network Delay and Dropout*: Sometimes several packets cannot be received in time due to network delays or dropouts. Bonawitz *et al.*'s method [16] has considered this issue and it has a "double-masking" scheme to address it, which has a high computation overhead. In DemoFL's secure learning process, a leader will not be waiting for participants' parameters permanently. It has a time limit, over which the leader will abandon waiting and send the current  $B_j$  to the server (introduced in Section IV). Under this circumstance, some common parties will be recognized as having lost connection and removed from the participant-group without contributing to the learning process. However, if a party does not lose the connection while its message reached the leader late because of network delay, discarding it may influence the target model's accuracy. Since the leaders will compute the intersection no matter there are dropouts or not, the dropouts have no negative effect on efficiency. We set the *dropout\_rate* to show the probability that a common party fails to send its parameters to leaders due to packet loss or network delay in one epoch. We then carried out several experiments to observe how package losses impact the learning process. The result is shown in Figure 10. The result indicates that the influence of dropouts on accuracy is acceptable when *dropout\_rate* is no more than 10%: when the *Epoch* gets larger than about 160, the accuracy gap between no-dropout models and with-dropout models becomes insignificant. In addition, the accuracy of model learned in the situation with 5%*dropout\_rate* is even better than the model learned in the situation without *dropout\_rate*, which may be caused by biases, or the inherent defects of federated learning's aggregation scheme, and this might be one of our future researchs. In summary, DemoFL has a strong resistance to packet loss considering that most machine learning tasks have a large enough *Epoch<sub>max</sub>*.

## VII. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a novel, efficient, and privacy-preserving federated learning framework named Democratic Federated Learning. DemoFL adopted MPC-based secret sharing methods to achieve privacy-preserving against honest-but-curious adversaries. It also employs consensus algorithms based leader election algorithms to reduce the communication cost and provide higher efficiency and robustness. Our experiments showed that DemoFL has a linear time complexity VS either the number of clients or training rounds. And it has a high resistance to member changes. DemoFL is lightweight and easy-deploying for many existing federated frameworks such as Pysyft and FATE.

However, it remains a problem for federated learning that malicious attackers are more harmful and difficult to defend against compared to honest-but-curious attackers. A malicious attacker can deviate from designed protocols or cheat on data that would go through it. Enhanced MPC protocols such as SPDZ [23] may help to deal with malicious attackers, which could be future work. Therefore, detecting malicious nodes is challenging work. In addition, other federated learning schemes besides FedAvg may contain algebraic calculus more than only addition and multiplication, which means existing MPC based methods will gain high overhead on those

schemes. Some blockchain techniques are inspiring to solve malicious attacks such as poisoning and adversarial examples, which may be adopted to enhance DemoFL to provide higher security by preventing fraud. It would be significant and interesting for future work to find more approaches to improve the security and efficiency of federated learning.

## REFERENCES

- [1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016.
- [2] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," 2018.
- [3] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2512–2520.
- [4] Z. Li, Z. Huang, C. Chen, and C. Hong, "Quantification of the leakage in federated learning," *arXiv preprint arXiv:1910.05467*, 2019.
- [5] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 739–753.
- [6] E. Shi, T. H. Chan, E. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in *Proc. NDSS*, vol. 2. Citeseer, 2011, pp. 1–17.
- [7] K. Pillutla, S. M. Kakade, and Z. Harchaoui, "Robust aggregation for federated learning," 2019.
- [8] K. Bonawitz, F. Salehi, J. Konečný, B. McMahan, and M. Gruteser, "Federated learning with autotuned communication-efficient secure aggregation," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, 2019, pp. 1222–1226.
- [9] K. Mandal, G. Gong, and C. Liu, "Nike-based fast privacy-preserving highdimensional data aggregation for mobile devices," CACR Technical Report, CACR 2018-10, University of Waterloo, Canada, Tech. Rep., 2018.
- [10] K. Mandal and G. Gong, "Privfl: Practical privacy-preserving federated regressions on high-dimensional data over mobile networks," in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, ser. CCSW'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 57–68. [Online]. Available: <https://doi.org/10.1145/3338466.3358926>
- [11] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, Jul. 2018. [Online]. Available: <https://doi.org/10.1145/3214303>
- [12] J. Weng, J. Zhang, M. Li, Y. Zhang, and W. Luo, "Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2019.
- [13] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Blockchain and federated learning for privacy-preserved data sharing in industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 6, pp. 4177–4186, 2020.
- [14] H. Kim, J. Park, M. Bennis, and S. Kim, "Blockchained on-device federated learning," *IEEE Communications Letters*, vol. 24, no. 6, pp. 1279–1283, 2020.
- [15] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 1982, pp. 160–164.
- [16] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1175–1191. [Online]. Available: <https://doi.org/10.1145/3133956.3133982>
- [17] R. Kanagavelu, Z. Li, J. Samsudin, Y. Yang, F. Yang, R. S. M. Goh, M. Cheah, P. Wiwatphonthana, K. Akkarajitsakul, and S. Wangz, "Two-phase multi-party computation enabled privacy-preserving federated learning," 2020.

- [18] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theor.*, vol. 22, no. 6, p. 644–654, Sep. 2006. [Online]. Available: <https://doi.org/10.1109/TIT.1976.1055638>
- [19] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," 2019.
- [20] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, Nov. 1979. [Online]. Available: <https://doi.org/10.1145/359168.359176>
- [21] M. O. Rabin, "How to exchange secrets with oblivious transfer," 2005, harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005. [Online]. Available: <http://eprint.iacr.org/2005/187>
- [22] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "SoK: general-purpose compilers for secure multi-party computation," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.
- [23] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012 - Volume 7417*. Berlin, Heidelberg: Springer-Verlag, 2012, p. 643–662. [Online]. Available: [https://doi.org/10.1007/978-3-642-32009-5\\_38](https://doi.org/10.1007/978-3-642-32009-5_38)
- [24] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology — CRYPTO '91*, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 420–432.
- [25] A. Beimel, "Secret-sharing schemes: A survey," in *Coding and Cryptology*, Y. M. Chee, Z. Guo, S. Ling, F. Shao, Y. Tang, H. Wang, and C. Xing, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 11–46.
- [26] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, p. 133–169, May 1998. [Online]. Available: <https://doi.org/10.1145/279227.279229>
- [27] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06. USA: USENIX Association, 2006, p. 307–320.
- [28] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 305–319. [Online]. Available: <https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro>
- [29] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *CoRR*, vol. abs/1711.10677, 2017. [Online]. Available: <http://arxiv.org/abs/1711.10677>
- [30] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 493–506. [Online]. Available: <https://www.usenix.org/conference/atc20/presentation/zhang-chengliang>
- [31] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. Vincent Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.
- [32] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," 2017.
- [33] A. Triastecyn and B. Faltings, "Federated learning with bayesian differential privacy," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 2587–2596.
- [34] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, ser. AISec'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–11. [Online]. Available: <https://doi.org/10.1145/3338501.3357370>
- [35] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, "Hybridalpha: An efficient approach for privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, ser. AISec'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 13–23. [Online]. Available: <https://doi.org/10.1145/3338501.3357371>
- [36] "Fate (federated ai technology enabler)," <https://fate.fedai.org/>, accessed July 4, 2020.
- [37] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *CoRR*, vol. abs/1811.04017, 2018. [Online]. Available: <http://arxiv.org/abs/1811.04017>
- [38] W. Du and M. J. Atallah, "Secure multi-party computation problems and their applications: a review and open problems," in *NSPW '01*, 2001.
- [39] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 805–817. [Online]. Available: <https://doi.org/10.1145/2976749.2978331>
- [40] H. Zhu, Z. Li, M. Cheah, and R. S. M. Goh, "Privacy-preserving weighted federated learning within oracle-aided mpc framework," 2020.