ShuffleFL: Gradient-Preserving Federated Learning using Trusted Execution Environment

Yuhui Zhang
SKLOIS, Institute of Information Engineering, CAS
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, CHINA
zhangyuhui@iie.ac.cn

Zhiwei Wang
SKLOIS, Institute of Information Engineering, CAS
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, CHINA
wangzhiwei@iie.ac.cn

Jiangfeng Cao
SKLOIS, Institute of Information Engineering, CAS
Beijing, CHINA
caojiangfeng@iie.ac.cn

Rui Hou*
SKLOIS, Institute of Information Engineering, CAS
Beijing, CHINA
hourui@iie.ac.cn

Dan Meng
Institute of Information Engineering, CAS
Beijing, CHINA
mengdan@iie.ac.cn

ABSTRACT

Federated Learning (FL) is a promising approach to privacy-preserving machine learning. However, recent works reveal that gradients can leak private data. Using trusted SGX-processors for this task yields gradient-preserving but requires to prevent exploitation of any side-channel attacks.

In this work, we present ShuffleFL, a gradient-preserving system using trusted SGX, which combines random group structure and intra-group gradient segment aggregation for combating any side-channel attacks. We analyze the security of our system against semi-honest adversaries. ShuffleFL effectively guarantees the participants' gradient privacy. We demonstrate the performance of ShuffleFL and show its applicability in the federated learning system.

CCS CONCEPTS

• **Security and privacy → Domain-specific security and privacy architectures**; *Hardware-based security protocols*; *Security protocols*; *Privacy-preserving protocols*; • **Hardware →** Model checking.

KEYWORDS

federated learning, gradient-preserving, trusted execution environment

ACM Reference Format:
Yuhui Zhang, Zhiwei Wang, Jiangfeng Cao, Rui Hou, and Dan Meng. 2021. ShuffleFL: Gradient-Preserving Federated Learning using Trusted Execution Environment. In *Computing Frontiers Conference (CF '21), May 11–13, 2021, Virtual Conference, Italy*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3457388.3458665

*Corresponding author: Rui Hou (hourui@iie.ac.cn).

CF '21, May 11–13, 2021, Virtual Conference, Italy

ACM ISBN 978-1-4503-8404-9/21/05…$15.00
https://doi.org/10.1145/3457388.3458665

1 INTRODUCTION

Federated Learning is a promising framework that builds a machine learning model collaboratively among multiple nodes (participants) while preserving each node's privacy data[14]. It has been widely believed that gradient information does not disclose private data in the past. By sharing gradients instead of sensitive data, participants can participate in model training through an aggregation server that computes a weighted sum of gradients.

However, recent works have subverted the previous opinion. They showed that a binary classifier trained on gradients can determine whether an exact data record (membership inference[15, 21]) or a data record with certain properties (property inference [15]) is included in the other participants' batch. The GAN model was proposed to generate prototypical samples of the targeted training set that was meant to be private [5, 7, 9, 15]. Phong *et al.* demonstrated that a small portion of gradients might reveal information on local data. Though there are noises on the reconstructed images, the ground truth label of the sample is still recognizable[17]. Zhu *et al.* proved the possibility of obtaining private training data from publicly shared gradients, known as *Deep Leak from Gradient* (*DLG*)[24]. DLG reveals pixel-wise accurate images and token-wise matching texts compared with conventional approaches. Thus, these attacks motivate the importance of protecting gradient information when designing a federated learning system.

Existing solutions to protect gradients need to be improved in terms of model accuracy and/or efficiency. For example, ① *Secure Multi-Party Computation*: Bonawitz *et al.* proposed a secure multi-party computation (SMC) protocols based on secret-sharing, known as *Secure Aggregation*. Although secure aggregation ensures no individual participant's gradients can be inspected, it is confronted with poor robustness issues [3]. ② *Homomorphic Encryption*: Phong *et al.* adopted homomorphic encryption to aggregate gradients[17]. But it requires massive computation resources for cryptographic

operations. ③ *Differential Privacy*: Shokri and Shmatikov took advantage of differential privacy that masks sensitive attributes to make it difficult to recover gradient information, while the noises lead to the loss of accuracy[20].

As a straightforward approach, people can utilize widely-used hardware Trusted Execution Environment (TEE) for federated learning system, e.g. Intel Software Guard eXtensions (SGX)[11]. TEE acts as the aggregation center to protect the gradient from attackers. However, the SGX mechanism has lots of attack surfaces due to resource sharing, such as page table[19, 23], cache[8, 13, 18, 22], CPU internal structures[12] etc.

Our solution addresses these issues by ShuffleFL, a system for efficient, private aggregation of gradient. ShuffleFL utilizes a widely-used hardware TEE, Intel Software Guard eXtensions (SGX)[11]. On the ground that SGX faces side-channel attacks, we adopt a random grouping algorithm to dynamically organize all participants into a hierarchical group structure and combine intra-group gradient segment aggregation to against adversaries. These strategies make it more difficult for gradient-stealing. It is important to shuffle the group structure before each iteration begins. In this way, we greatly reduce the possibility that the adversaries obtain gradient information to recover private data or train an attack model.

The main contributions are:

- We propose ShuffleFL, in which all aggregation operations are running in hardware enclaves. We first adopt a random grouping algorithm and intra-group gradient segment aggregation to prevent any `side-channel attacks`. Further, by aggregating gradient without value modification, our approach has lossless model accuracy. Besides, this system is still dropout-robust, because the aggregation running in enclaves follows the traditional `FedAvg` aggregation protocol.
- We analyze the security of ShuffleFL in three threat models against semi-honest adversaries. The results show that it is reliable when there are no colluding attacks, or the number of corrupt participants is limited.
- We implement and evaluate the performance of ShuffleFL. We deploy a testbed with one server node and multiple participant nodes. All of these nodes own an Intel Xeon E3-1240 v6 processor to support SGX. The results show that ShuffleFL effectively reduces computational cost of gradient aggregation, compared with using SGX as an aggregation center.

2 BACKGROUND

2.1 Federated Learning

To meet the requirements of privacy protection, data security, and government regulations, one of the most popular solutions is training an efficient machine learning model $f_\theta(x)$ with data from multiple institutions, but without data sharing. As a distributed machine learning paradigm, federated learning allows participants to jointly model based on no data sharing. Through sharing gradient instead of private data, participants can exchange their training contribution for joint modeling.

Each participant calculates the gradient $g = \nabla f_{\theta^{(t)}}$ on its local data at the current model $\theta^{(t)}$. After pulling the gradient and the training data size from participants, the server aggregates their gradient to update the model from $\theta^{(t)}$ to $\theta^{(t+1)}$:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \sum_{i=1}^{|K|} \frac{n_i}{n} g_i, t \geq 0 \quad (1)$$

Where K is the collection of participants, η is the step size or learning rate, n is the total data size of all participants, n_i, and g_i respectively are the training data size and gradient of ith participant in K. θ^0 is initialized with a random value or all-zero.

2.2 Trusted Execution Environment

In this work, we use enclaves provided by Intel Software Guard eXtensions (SGX).

Security Properties. Intel SGX, as a practical TEE, protects the confidentiality and integrity of code and data through memory isolation. It is a new extension of the Intel processor, which brings new instruction set and memory control mechanisms. The extension allows applications to implement a container called *enclave*, which separates the protected memory space for applications. Enclave protects applications from malicious software even if it has privilege.

Moreover, SGX provides remote attestation among platforms for users to verify the identity of the enclave. Remote attestation mainly contains the following functions: ① Verify the identity of the enclave. ② Verify that the code running in SGX has not been tampered with. ③ Negotiate a key pair through Intel, which is used for encrypting transmitted data. In our work, each participant depends on remote attestation to prove its identity and establishes secure communication channels with the server and other participants.

Weakness. Although the enclave protects the content of the CPU and memory area, data may still be exfiltrated through side-channel attacks [4, 18, 19, 22, 23]. For instance, using maliciously constructed inputs, timing the max operator can reveal the value of a target input [8]. A similar attack can expose the value of a federated learning floating-point number by measuring the extra cycles taken when the value is subnormal (very near zero) [1].

3 THREAT MODEL

In our work, we assume that a corrupt participant follows the prescribed operations and protocols, but he/she tries to learn unintended information about gradient or honest participants' training data (this adversary is honest but curious or passive, or semi-honest). That is, he will not provide false training data or modify the uploaded gradient. So the data poisoning attack [2] is beyond the scope of our current work. Besides, the administrator of each node still possibly utilize the side channels of SGX to steal the internal sensitive data. The corrupt participants can collaborate.

It can be classified into three threat models: 1) Client-only threat model: some clients are malicious. 2) Server-only threat model: no client is malicious, but the server is malicious. 3) Colluding threat model: the corrupt participants collaborate.

4 OVERVIEW

Assuming that there are $|P|$ participants. We denote the server by S, and a participant by $P_i, i \in [0, |P| - 1]$, who holds the dataset denoted by $\mathcal{D}_i, |\mathcal{D}_i| \geq 1$. We have the following goal:

Goal. The main focus of ShuffleFL is enabling participants to learn a machine learning model collaboratively, without leaking any information about $\mathcal{D}$ to adversaries.

The main challenges while achieving this goal are:

- *Dropout-robust:* ShuffleFL should tolerate the participants to drop out at any time. Therefore dropout-robust training is required.
- *Accuracy:* We have greatly emphasized the accuracy of the model. ShuffleFL should output the correct model, so it will not lose accuracy compared to traditional federated learning.
- *Privacy:* ShuffleFL aims at protecting local private data. The adversaries should learn no information about participants' private data.
- *Efficiency:* The computation and communication costs should be as small as possible.

Basic idea. Sending the local gradient directly to the server may leak participants' gradient which can be used to recover private data. A straightforward idea to prevent such leakage is performing the aggregation task on an SGX-enabled center. The gradient transmitted to the center is in cipher-text, and the aggregation is performed in an enclave. Although participants do not trust the aggregation center, they can review the aggregation code, deploy the code into a processor-protected memory region (called an enclave), perform remote attestation, upload their encrypted gradient just for this task, and finally get the aggregated gradient. Figure 1 provides an overview of this system.

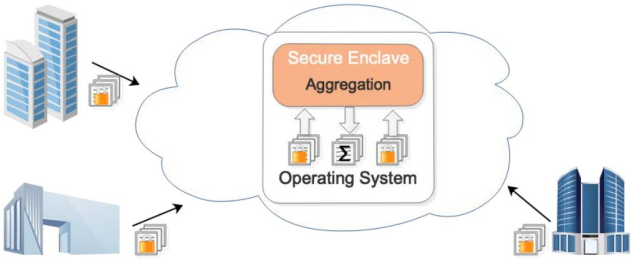


Figure 1: Federated learning using SGX as an aggregation center.

Data in the enclave may still be exfiltrated through side-channel attacks. Thus, the basic idea brings a potential privacy risk: when a side-channel attack occurs, the server can easily access all participants' gradient. Decentralization is a significant trend that extends the security dependency from a server to multiple centers. The centers firstly aggregate the intra-group gradient, therefore a group level gradient is accessible to the server rather than an individual level gradient. However, dividing all participants into fixed groups also has privacy risks. So we introduce a random self-organization grouping algorithm to organize all participants into a hierarchical group structure. Nevertheless, there is still an obvious shortcoming: once the group leader is attacked by side-channel attacks, the gradient of the group members will be leaked without reservation, so we develop a strategy named intra-group gradient segment aggregation.

ShuffleFL Overview. Figure 2 shows the architecture of ShuffleFL. The system is comprised of participant nodes and a server node. The execution of ShuffleFL can be summarized in three phases:

▹ *Phase 1: Random grouping.* The random grouping algorithm dynamically reorganizes participants into a hierarchical group structure before each iteration of training. An example is shown in Figure 2(a). C2 nodes are the leaders of C3 nodes, C1 nodes are the leaders of C2 nodes. Randomization increases the uncertainty of group structure, making it difficult for a corrupt participant to be a teammate with the victim repeatedly.

▹ *Phase 2: Intra-group gradient aggregation.* Figure 2(c) illustrates the workflow of intra-group gradient aggregation which named *segment aggregation.* The gradient is no longer directly sent to the group leader but divided into multiple segments. Then, these segments are distributed to other members in the same group as Figure 2(c1). Group members follow the aggregation process as illustrated by the numbered steps in Figure 2(c2), at each iteration, the members ① update the model with the local data, and meanwhile, ② send the segment pulling requests to other members, once the update is finished, they ③ encrypt and ④ send the segments to the requestors as a response of the pulling requests, when all the pulling requests are satisfied, in enclaves, the members ⑤ decrypt and ⑥ aggregate the gradient segments, and send the aggregated segments to the group leader who then ⑦ put them together to rebuild a complete gradient.

In this phase, the leader's gradient also participates in the aggregation. All aggregation operations are performed within their enclaves for gradient-preserving. As illustrated in Figure2(c2), while the participant is providing the gradient segments to others, the segments are encrypted with the key negotiated during the SGX remote attestation as step ③. All decryption and aggregation operations are performed within the enclaves as step ⑤ and ⑥.

▹ *Phase 3: Inter-group gradient aggregation.* After the accomplishment of all intra-group aggregation tasks among lower-level nodes, new aggregation tasks will be performed among the higher-level nodes, and so on until the top-level. As shown in Figure 2(b), the C2 nodes rebuild the gradient segments produced by C3 nodes into the new gradient, and then participate in the new aggregation tasks as group members of the C1 nodes.

Advantages of ShuffleFL. The advantages are as follows. *(1) Dropout-robust.* The aggregation operation within SGX follows the traditional federated learning's aggregation protocol. Therefore the participants are allowed to drop out at any time. *(2) Accuracy lossless.* The gradient used to update the model is consistent from traditional federated learning, so we believe that there is no loss of the model accuracy compared to traditional federated learning. *(3) Privacy.* Enclave owners can depend on remote attestation to prove their identity and establish secure communication channels with each other. All gradient are plain-text only in enclave. Since SGX may face side-channel attacks, random grouping and segment aggregation alleviate the risk of side-channel attacks.

5 SHUFFLEFL IMPLEMENTATION

5.1 Random Self-Organization Grouping

Group Structure Design. A simple decentralized way is to divide participants according to the number of centers. $|P|$ denotes the number of participants, $|G|$ denotes the number of centers,

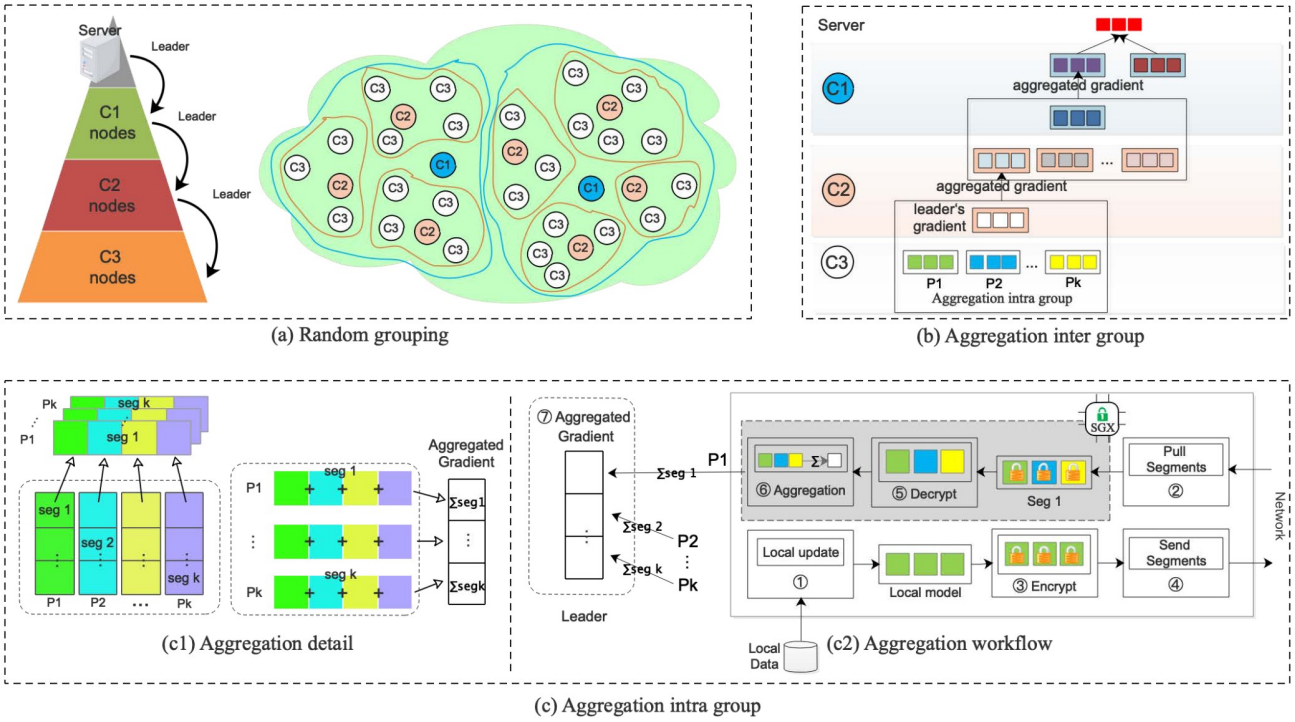


Figure 2: An overview of ShuffleFL. (a) Participants shuffle the group structure by random grouping before each iteration begins, (b) and then start the intra-group gradient aggregation from the bottom layer to the top layer. (c)The gradient within the group is aggregated in segments.

that is, the number of groups. The server manages $|G|$ leaders. A group leader manages $\frac{|P|}{|G|} - 1$ group members. $|G|$ and $\frac{|P|}{|G|} - 1$ are inversely proportional when the number of participants $|P|$ is invariable. However, this system still faces the following challenges:

- *Privacy.* When $\frac{|P|}{|G|} - 1$ is larger, a group leader will manage more members. A corrupt group leader can easily steal more participants' gradient through side-channel attacks.
- *Efficiency.* When $|P|$ is certain, one of $|G|$ and $\frac{|P|}{|G|} - 1$ is easy to be large. The EPC of SGX1.0 is limited to 128 Megabytes, when the memory limit is exceeded, the computational cost increases about tens of times. Therefore, we expect both the number of leaders and the number of group members to be appropriate, although SGX2.0 is available in the future.

A hierarchical group structure is selected to constrain both the group size and the number of groups. As an example in Figure 2(a), all participants and the server are divided into four layers, the server is at the top-level and responsible for C1 nodes, C1 nodes are responsible for C2 nodes, and so on.

Random Self-Organization Grouping Algorithm. Before each iteration begins, the system will shuffle the group structure. All participants will redetermine the group they belong to and their group leaders, as described in Algorithm 1. It is worth noting that the red, underlined parts will be moved in the enclave to obtain the protection of SGX. Benefitting from the integrity protection provided by SGX, the code will be verified. Additionally, exchanging information on a protected channel guarantees reliable random grouping. The given value $GMaxSize$ defines the maximum group size. Then, each participant generates a random value in $[1, GMaxSize]$. Participants with the same value are grouped together and a group leader is randomly voted. If a group's size is larger than $GMaxSize$, this process is recursively executed until all groups' size less than $GMaxSize$. $(pk_{i,j}, sk_{i,j})$ is the pairwise keys established between P_i and P_j during the step of SGX Remote Attestation. P_i holds $pk_{i,j}$ used to encrypt data that is transferred to participant P_j. So, participant P_i maintains a list of public keys $PKList_i = \{pk_{i,j}\}_{j=1}^{|P|}$. Similarly, P_j holds $sk_{i,j}$ used to decrypt the data from participant P_i and maintains a list of private keys $SKList_j = \{sk_{i,j}\}_{i=1}^{|P|}$.

5.2 Intra-group Gradient Aggregation

Prior work reveals that the sparse gradient makes it harder to infer private data. For instance, the maximum tolerance of gradient sparsity is around 20% in DLG[24]. When the pruning ratio is larger, the recovered images are no longer visually recognizable and thus effectively prevent privacy leakage. Inspired by this work, we develop a strategy named intra-group gradient segment aggregation to address the shortcoming of corrupt leaders. Figure 2(c) illustrates this approach. We assume that there are $k-1$ members and a leader in a group.

Gradient Segment, Encryption, and Sending. The members firstly decide how to partition the gradient. They don't have to follow the equal segmentation rule. Still, for simplicity, we assume

Algorithm 1 Random Self-Organization Grouping. (*take participant P_i as an example*)

Require: P: Participant set; $GMaxSize$: Maximum group size; $lastLevelL$: Last level leader;
Ensure: P_i's leader

function GROUPING(P, $lastLevelL$, $GMaxSize$)
▷ **//Generate group companions**
  $RG_i \leftarrow$ Generate a random number in $[1, GMaxSize]$ to indicate a group id that P_i belongs to.
  for P_j in P **do**
    $E_{pk_{i,j}}(RG_i) \leftarrow$ Encrypt RG_i with $pk_{i,j}$ and transmit $E_{pk_{i,j}}(RG_i)$ to P_j
  end for
  $E(RGTable) \leftarrow$ The table of received $E_{pk_{j,i}}(RG_j)$ form P_j, $P_j \in P$
  for $E_{pk_{j,i}}(RG_j)$ in $E(RGTable)$ **do**
    $RG_j \leftarrow Pi$ decrypts $E_{pk_{j,i}}(RG_j)$ use $sk_{j,i}$
    If $(RG_j == RG_i)$ **then** add P_j to the teammate list $sameG$
  end for
▷ **//Vote for the group leader**
  $RL_i \leftarrow$ Generate a random number in $[0, sizeOf(sameG)]$ to indicate a leader id that P_i votes to.
  for P_s in $sameG$ **do**
    $E_{pk_{i,s}}(RL_i) \leftarrow$ Encrypt RL_i with $pk_{i,s}$ and transmit $E_{pk_{i,s}}(RL_i)$ to P_s
  end for
  $E(RLTable) \leftarrow$ The table of received $E_{pk_{s,i}}(RL_s)$ form P_s, $P_s \in sameG$
  P_i decrypts $E(RLTable)$ and obtains $RLTable$
  $MaxVoteNum \leftarrow$ The most appearing leader id in $RLTable$
  $currentL \leftarrow$ Sort $sameG$ by the rules, and take the $MaxVoteNum - th$ participant
▷ **//Determine whether the group size meets the conditions, if not, then continue to group recursively**
  if $(currentL == P_i)$ **then return** $lastLevelL$
  else if $(sizeOf(sameG) <= GMaxSize$ **then return** $currentL$
  else GROUPING($sameG.delete(currentL)$, $currentL$, $GMaxSize$)
  end if
end function

they partition the gradient into k segments equally and each member is responsible for one of the segments. After encrypting with the corresponding key, the members dispatch their gradient segments to the corresponding members.

Let $\mathcal{G}$ denotes the gradient. The group members firstly divide $\mathcal{G}$ into k segments without overlapping:

$$\mathcal{G} = (\mathcal{G}[1], \mathcal{G}[2], ..., \mathcal{G}[k]) \tag{2}$$

For member i who is responsible for segment i, the member chooses a peer teammate which we denote it as j and then actively pulls the corresponding segment $\mathcal{G}_j[i]$ from he/her. Note that this step is parallelized to make full use of the bandwidth. When the member i fetches all the model segments back, a set of segment i can be get such as $\{\mathcal{G}_1[i], \mathcal{G}_2[i], ..., \mathcal{G}_k[i]\}$.

Segments Decryption and Aggregation. While the member is providing the gradient segments to others, it is also receiving the segments it has requested previously. Then, in his/her enclave, this member decrypts and aggregates the received gradient segments and sends the aggregated segment to his/her leader.

Typically the received gradient segments are aggregated by weighted averaging with the group members' dataset size as weight. Let P_i denotes the set of the participants who provide segment to member i, $|\mathcal{D}_j|$ denotes the dataset size of member j. Aggregate segments in $\mathcal{G}_j[i]$:

$$\widetilde{\mathcal{G}}[i] = \frac{\sum_{j \in P_i} |\mathcal{D}_j| \mathcal{G}_j[i]}{\sum_{j \in P_i} |\mathcal{D}_j|} \tag{3}$$

Gradient Rebuilding. Eventually, the group leader will receive the aggregated gradient segment of each group member. The leader put the aggregated gradient segments together into a complete gradient according to the segmentation rule.

After receiving all the aggregated gradient segments, the leader can rebuild the final aggregation result by

$$\widetilde{\mathcal{G}} = (\widetilde{\mathcal{G}}[1], \widetilde{\mathcal{G}}[2], ..., \widetilde{\mathcal{G}}[k]) \tag{4}$$

6 SECURITY ANALYSIS

This section presents the security analysis of ShuffleFL against semi-honest adversaries. We discuss three threat models separately: 1) Security against only client; 2) Security against only server; and 3) Security against colluding attack.

Threat Model 1 (Security against only server). ShuffleFL is not threatened by server-only threat model. Because the gradient of the participant has been aggregated in groups before being sent to the server, the server can only obtain the sum of participants' gradient.

Threat Model 2 (Security against only client). The total number of participants is $|P|$, the group size is $|G|$. In this threat mode,

when the victim and the malicious client are in the same group, the private data may be stolen. The probability of being in the same group is $C_{|P|-2}^{|G|-2}/C_{|P|}^{|G|}$. At this point, the amount of gradient information that the attacker can obtain accounts for $\frac{1}{|G|}$ of the total amount of gradient information.

$$C_{|P|-2}^{|G|-2} = \frac{A_{|P|-2}^{|G|-2}}{(|G|-2)!} = \frac{(|P|-2)!}{(|P|-|G|)!(|G|-2)!} \tag{5}$$

$$C_{|P|}^{|G|} = \frac{A_{|P|}^{|G|}}{|G|!} = \frac{|P|!}{(|P|-|G|)!|G|!} \tag{6}$$

$$\frac{C_{|P|-2}^{|G|-2}}{C_{|P|}^{|G|}} = \frac{|G|(|G|-1)}{|P|(|P|-1)} \tag{7}$$

Users can select the maximum gradient of information they think can be leaked based on the actual scenario, in which case their original data will not be inferred. Assuming that the user thinks that 20% of the gradient information cannot recover the original private data(In our example, we chose a small proportion of gradient information. Actually, it was mentioned in the work of Zhu et.al that the original information could not be recovered when the gradient information was less than 80%[24].), it needs $|G| > 5$ to ensure that each group member can obtain less than 20% of the gradient information of other members through side-channel attack. At this time, when the number of participants $|P| > 45$, the probability of the attacker's success is less than 1%. Therefore, users can adjust the group size according to the tolerable leakage of gradient information, the number of participants and other parameters to achieve an almost zero probability of being attacked.

Threat Model 3 (Security against colluding attack). We do not consider server-client collusion attack, because the server can only obtain the aggregated gradient, and we assume that the aggregated gradient obtained by the server is not a secret. Only when the victim is a leaf node or the parent of a leaf node, can malicious clients obtain gradient information. Because the information is aggregated multiple times as it passes to higher layers, the original information cannot be restored. The total number of participants is $|P|$, of which the number of corrupt participants is $|P_c|$, the group size is $|G|$.

(1) Scenario 1: As a leaf node. In the group where the victim is located, the number of malicious clients(including victim's parent) is $|G_a|$, then through a collusion attack, the proportion of gradient information that may be leaked at most is $\frac{|G_a|}{|G|+1}$, when $|G_a| < |G|$. When $|G_a| = |G|$, the proportion is 1.

(2) Scenario 2: As a leaf nodes' parent. The number of malicious child nodes is $|G_a|$, the proportion of gradient information that may be leaked at most is $\frac{|G_a|}{|G|+1}$, when $|G_a| <= |G|$.

Simulation Results Analysis. We simulated the process with different numbers of participants $|P|$ and corrupt participants $|P_c|$. Each experiment iterated this process 10,000 times to get the probability of the honest client being attacked. As shown in Table 1, we derive the relationship between the proportion of corrupt participants and the proportion of the honest participant's gradient information that may be leaked. We can find that even when the proportion of malicious colluding clients reaches 60%, the probability of leaking half of the gradient information is only about 30%. Although the probability of obtaining fewer gradients will increase, the difficulty of recovering private data will greatly increase. It's almost impossible when the proportion of colluding malicious clients decreases, or when they want to get more gradients information. So when the number of malicious participants is less than 2/3, the system can effectively protect private data.

Table 1: Colluding attack: probability of being attacked.

Participants	Corruption Ratio	Leakage Ratio of the Whole Gradient				
		50%	60%	70%	80%	90%
100	20%	0.018	0	0	0	0
	40%	0.062	0.030	0.016	0.01	0.01
	60%	0.344	0.094	0.098	0.052	0.027
500	20%	0.004	0	0	0	0
	40%	0.067	0.021	0.002	0	0
	60%	0.318	0.086	0.09	0.019	0.002
1000	20%	0	0	0	0	0
	40%	0.062	0.009	0	0	0
	60%	0.33	0.091	0.093	0.006	0

7 PERFORMANCE EVALUATION

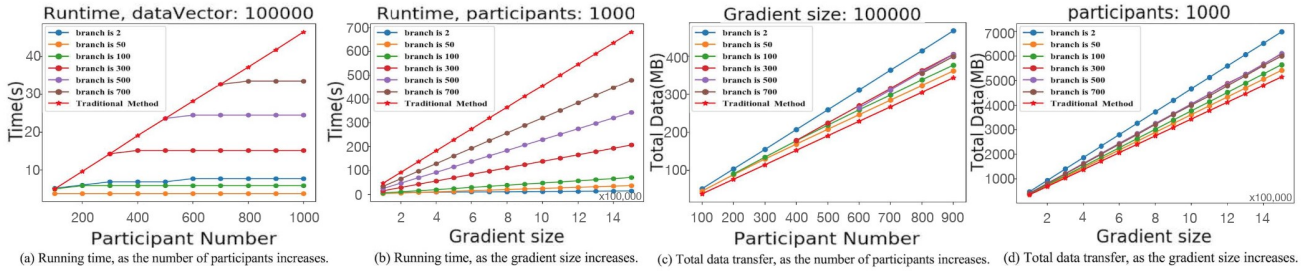
7.1 Experiment Setup

To ascertain the performance, we benchmark ShuffleFL on the axes of computation cost and communication cost. We no longer measure the accuracy of the model, because the gradient used to update the model is consistent with traditional federated learning[14], there is no loss of the accuracy. The training process of all participants is carried out synchronously, and we assume that the parameters of each participant are the same as traditional FL, so we ignore the time cost of the training process. For both the participants and the server, almost all of the cost comes from aggregating gradient and random grouping. We evaluate the performance of different numbers of participant nodes, which ranges from 100 to 1000. And the number of gradient size ranges from 100,000 to 1,500,000. The parameter *GMaxSize* (branches) of random grouping algorithm ranges from 0 to the number of participant nodes. The specific settings are listed as follows:

▹ *Execution environment settings.* The experiments were conducted on the machines with the major configurations listed in Table 2. This processor limits the amount of platform memory that can be reserved for enclaves to 94 Megabytes (out of a total of 128 Megabytes of protected memory).

Table 2: System Configuration

Configuration	Server Node	Participant Node
CPU	Intel(R) Xeon(R) CPU E3-1240 v6 @ 3.70GHz	
OS	Ubuntu 18.04 64-bit	
Memory	24GiB	8GiB
Disk	40GiB	40GiB



(a) Running time, as the number of participants increases. (b) Running time, as the gradient size increases. (c) Total data transfer, as the number of participants increases. (d) Total data transfer, as the gradient size increases.

Figure 3: Performance.

▹ *Comparison settings.* We compare ShuffleFL with a traditional federated learning system[14]. In traditional federated learning system, the SGX-enable server plays the role of `aggregation center` and aggregates gradient pulled from all participants in their enclave.

7.2 Computation Cost

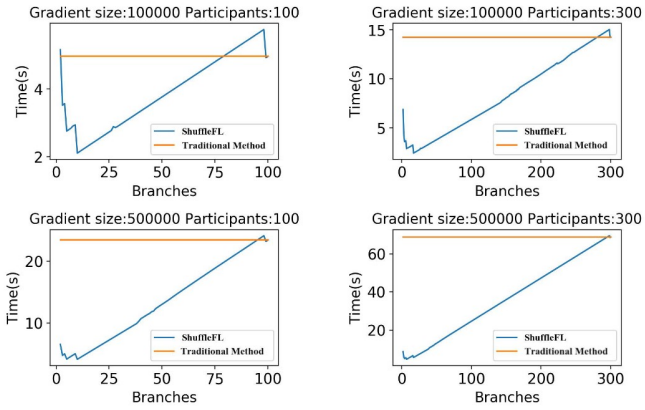


Figure 4: Running time. **(Sensitivity on the factor of branches)**

As seen in Figure 4, the running time of traditional federated learning with SGX is not affected by branches. For the running time of ShuffleFL, it is possible to exceed the running time of traditional federated learning with SGX only when the branch value is large enough or small enough. Still, as the gradient size and the number of participants increases, this possibility becomes smaller. And under the condition of optimal branching, the running time of ShuffleFL is much less than that of traditional federated learning with SGX.

This conclusion is also reflected in Figure 3(a)(b), where the running time of ShuffleFL increases with the number of participants, but the growth rate of the running time is far less than that of traditional federated learning with SGX. And as the aggregated gradient size becomes more extensive, the running time increases linearly, although as the branch value increases, its growth rate is in close proximity to the growth rate of the running time of traditional federated learning with SGX.

Therefore, under the condition of selecting an appropriate branch, ShuffleFL's computational cost is always less than that of traditional federated learning with SGX. The larger the gradient size (the model) and/or the greater the number of participants, the more obvious the advantage.

7.3 Communication Cost

Figure 5 shows that, as the number of branches increases, the total data size transferred of ShuffleFL decreases and then stabilizes. In the case of different numbers of participants and different gradient sizes, we statistically conclude that the data size transferred in ShuffleFL at the optimal branch is about 1.1 times that of the traditional method.

The communication cost also reflected in Figure 3(c)(d). The total transferred data size increases as the number of participants increases and gradient size but the change is not obvious compared to the traditional method when the number of branches changes. So, we can choose a suitable branch to obtain a tolerable transferred data size.

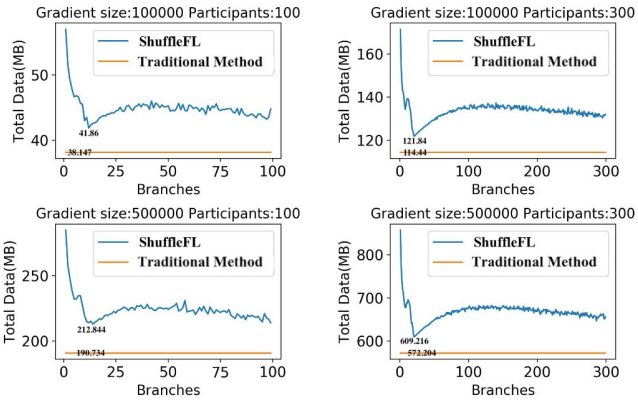


Figure 5: Total data transfer. **(Sensitivity on the factor of branches)**

8 RELATED WORK

Privacy-preserving federated learning generally falls into two categories: based on cryptography and trusted execution environment.

Existing Crypto-based Solutions Gradient security is a challenge to the security of federated learning. A main kind of approaches to protect gradient information are based on cryptography, such as Multi-Party Computation, Homomorphic Encryption, and Differential Privacy. ①*Multi-Party Computation*: Secure Aggregation proposed by [3] masks gradients in the exchanging stage. Then the gradients are unmasked in the aggregation process to acquire valid result. However, this approach is facing two challenges. First, it is vulnerable when the proportion of adversaries is more than 1/3. Second, the inputs must be well-formed, otherwise it will cause other clients' tasks to abort. ②*Homomorphic Encryption*: [17]'s approach proceeds directly on encrypted data without decrypting it.

It still requires massive computational resources for cryptographic operations which affect efficiency. For instance, [6] run a fully homomorphic encryption scheme on an IBM System x3500 server. Consequently, it takes 2 hours to generate a public key whose size is up to 2.3GB when the key's dimension is 32768. ③*Differential Privacy*: Shokri and Shmatikov add noises to sensitive attributes to make it difficult to recover gradient information, yet the accuracy is less than 60% when the privacy budget per parameter is 0.001.

TEE-based Solutions Some works adopt TEE, specifically SGX to ensure each party's data private[10, 16]. The common of these works is that each participant transfers their encrypted dataset to a centralized server to train a model. Because of the protection of SGX, contents inside SGX is unreachable to outside. These approaches rely on the security of SGX, which is its last line of defense. Because there are attacks towards SGX, such as the side-channel attack, SGX is not absolutely safe. They have proposed different solutions to address side-channel attacks. However, because the enclave of SGX1.0 has a memory limit of 128 Megabytes, it is not practical to train a model in the enclave, though this limitation may be addressed in SGX2.0.

9 CONCLUSION

In this paper, we proposed a new and practical distributed federated learning system based on SGX for securely aggregating gradients. To mitigate the risk of SGX being attacked by side-channel attacks, we apply two techniques, random grouping structure and intra-group gradient segment. This system makes it difficult for the adversary to obtain the complete gradient of victim. Besides, At the beginning of each iteration, this system will shuffle the group structure, this strategy greatly reduces the likelihood that the opponent will gain enough gradient to recover the original data. Moreover, we deploy a testbed supporting the proposed system for performance evaluation. Experimental results demonstrate the performance of *ShuffleFL* and show its applicability.

REFERENCES

[1] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. 2015. On subnormal floating point and abnormal timing. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 623–639.

[2] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2018. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459* (2018).

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

[4] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure: {SGX} cache attacks are practical. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*.

[5] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1322–1333.

[6] Craig Gentry and Shai Halevi. 2011. Implementing gentry's fully-homomorphic encryption scheme. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 129–148.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[8] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security*. ACM, 2.

[9] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep models under the GAN: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 603–618.

[10] Nick Hynes, Raymond Cheng, and Dawn Song. 2018. Efficient deep learning on multi-source private data. *arXiv preprint arXiv:1807.06689* (2018).

[11] Intel. 2013. Software Guard Extensions Programming Reference. Reference no. 329298-001US.

[12] Sangho Lee, Ming Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2016. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing. (2016).

[13] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. [n.d.]. Last-Level Cache Side-Channel Attacks are Practical. ([n. d.]).

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

[15] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2018. Inference Attacks Against Collaborative Learning. *CoRR* abs/1805.04049 (2018). arXiv:1805.04049 http://arxiv.org/abs/1805.04049

[16] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 619–636. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko

[17] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. 2018. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security* 13, 5 (2018), 1333–1345.

[18] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware guard extension: Using SGX to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 3–24.

[19] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. 2016. Preventing page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 317–328.

[20] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 1310–1321.

[21] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.

[22] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. 2017. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2421–2434.

[23] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 640–656.

[24] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep Leakage from Gradients. (2019).