Efficient Dropout-resilient Aggregation for Privacy-preserving Machine Learning

Journal:	*Transactions on Information Forensics & Security*
Manuscript ID	T-IFS-13394-2021.R1
Manuscript Type:	Regular Paper
Date Submitted by the Author:	09-Jan-2022
Complete List of Authors:	Liu, Ziyao; Nanyang Technological University, School of Computer Science and Engineering Guo, Jiale; Nanyang Technological University, School of Computer Science and Engineering Lam, Kwok-Yan; Nanyang Technological University Zhao, Jun; Nanyang Technological University, School of Computer Science and Engineering; Nanyang Technological University
Subject Category Please select at least one subject category that best reflects the scope of your manuscript:	CYBERSECURITY, ANONYMIZATION AND DATA PRIVACY
EDICS:	ADP-PPRO-Privacy protection < ADP-ANONYMIZATION AND DATA PRIVACY, ADP-PMOD-Privacy modeling and analysis < ADP-ANONYMIZATION AND DATA PRIVACY, ADP-APPS-Applications and systems < ADP-ANONYMIZATION AND DATA PRIVACY, CYB-APP-DIST-Distributed computing systems < CYB-CYBERSECURITY

Efficient Dropout-resilient Aggregation for Privacy-preserving Machine Learning

Ziyao Liu§, Jiale Guo§, Kwok-Yan Lam, and Jun Zhao

Abstract—Machine learning (ML) has been widely recognized as an enabler of the global trend of digital transformation in recent years. With the increasing adoption of data-hungry machine learning algorithms, personal data privacy has emerged as one of the key concerns that could hinder the success of digital transformation. As such, Privacy-Preserving Machine Learning (PPML) has received much attention of the machine learning community, from academic researchers to industry practitioners to government regulators. However, organizations are faced with the dilemma that, on the one hand, they are encouraged to share data to enhance ML performance, but on the other hand, they could potentially be breaching the relevant data privacy regulations. Practical PPML typically allows multiple participants to individually train their ML models, which are then aggregated to construct a global model in a privacy-preserving manner, e.g., based on multi-party computation or homomorphic encryption. Nevertheless, in most important applications of large-scale PPML, e.g., by aggregating clients' gradients to update a global model for federated learning, such as consumer behavior modeling of mobile application services, some participants are inevitably resource-constrained mobile devices, which may drop out of the PPML system due to their mobility nature. Therefore, the resilience of privacy-preserving aggregation has become an important problem to be tackled because of its real-world application potential and impacts. In this paper, we propose a scalable privacy-preserving aggregation scheme that can tolerate dropout by participants at any time, and is secure against both semi-honest and active malicious adversaries by setting proper system parameters. By replacing communication-intensive building blocks with a seed homomorphic pseudo-random generator, and relying on the additive homomorphic property of Shamir secret sharing scheme, our scheme achieves a significantly smaller cost and provides stronger dropout-resilience than existing schemes. The simplicity of our scheme makes it attractive both for implementation and for further improvements.

1 Introduction

With the widespread adoption of data-hungry machine learning algorithms and increasing concerns for personal data privacy protection, Privacy-Preserving Machine Learning (PPML) has emerged as an important area that received much attention of the machine learning community, from academic researchers to industry practitioners to government regulators [29, 37, 45]. PPML allows multiple participants, e.g., data owners, to jointly solve a machine learning problem while preserving their data privacy. Traditional PPML typically enables data owners to individually perform their ML to train a model using their local data, i.e., compute the gradients, which are then aggregated to construct a global model. However, as pointed out in [48], local data of an individual participant could be revealed through a small portion of gradients of its local model. This deep leakage from gradients even allows the attacker to recover images with pixel-wise accuracy and texts with token-wise matching. In this connection, cryptographic mechanisms such as secure multi-party computation (MPC) [19] and homomorphic encryption (HE) [18] have been proposed to enhance PPML by aggregating the local models in a privacy-preserving manner.

In general, one can enhance PPML by constructing a secure[1] aggregation scheme that protects the local models by privacy-preserving technology (PPT) such as differential privacy (DP) [15] and the aforementioned cryptographic mechanisms (i.e., MPC and HE). For example, DP can be applied to clients' gradients before they are uploaded to the aggregation server [47]. In this way, the privacy of gradient information is protected while the server can still aggregate the perturbed gradients to obtain an approximate result according to the properties of DP. Nevertheless, this method suffers from the trade-off between privacy protection and data usability, hence model performance. Whereas, for HE-based aggregation schemes such as [3, 12, 38], participants perform compute-intensive algorithms to encrypt their gradients and send to the server. Then the server aggregates the protected gradients by performing arithmetic operations on the ciphertext and sends them back to the participants for decryption. For example, about 1404 hours is required for training a convolutional neural network (CNN) [38].

With the rapid growth of the digital economy involving billions of users in Cyberspace, to ensure business sustainability in a highly competitive environment, most application service providers have adopted large-scale PPML to model user behavior and preferences in order to provide improved user experiences. It is worth noting that, a large number of users are likely to be on mobile devices, so PPML needs to be highly resilient and cope with the dynamic connectivity of mobile devices. For example, Gboard on Android (the Google Keyboard) [46] has deployed a PPML on mobile phones to improve the accuracy of suggested

• *Ziyao Liu, Jiale Guo, Kwok-Yan Lam, and JunZhao are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore, 50 Nanyang Ave, 639798.*
E-mail: {ziyao002, jiale001} @e.ntu.edu.sg, {kwokyan.lam, junzhao} @ntu.edu.sg

Manuscript received February 01, 2022; revised November 30, 2021.

§. Equal contribution

1. We use the terms secure and privacy-preserving interchangeably.

queries for the user's current typing context. Furthermore, researchers from Apple have used PPML deployed on iPhones to enhance the performance of speaker verification [21]. In this case, large-scale PPML inevitably involves resource-constrained mobile devices which may drop out of the system due to their mobility nature.

In this connection, dropout-resilient privacy-preserving aggregation, e.g., by aggregating clients' gradients to update a global machine learning model in a privacy-preserving manner for Federated Learning [27], has attracted tremendous attention of the research community because of its real-world application potential and impacts. For example, [12] proposed a scheme, based on threshold HE cryptosystem, to securely share the private key and to deal with dropout clients. However, this scheme requires all the clients to contribute to several expensive building blocks such as distributed key generation and decryption [23], which is not practical for large-scale PPML. Compared with [12], the scheme in [3] is more efficient by having the secret key held by every participant, though it sacrifices the privacy requirements of PPML that the privacy of gradient must be kept by the corresponding participant. On the other hand, a pure MPC-based aggregation scheme is not suitable for a large-scale PPML due to the huge communication overheads when evaluating complex functions such as a deep neural network (DNN), and the problem is further exacerbated by the fact that a lot of the PPML clients are resource-constrained mobile devices. For example, training a simple CNN one epoch requires about 7 hours in WAN setting [30]. Server-aided MPC-based schemes such as [34, 44] achieve good efficiency relying on a set of non-colluding servers. However, for PPML systems that deployed by a single agent such as a commercial company or government, while the trust distribution is limited to the number of aided servers, it is not easy to guarantee that the involved aided servers are non-colluding from a game-theoretical perspective.

Recent advancements in dropout-resilient secure aggregation significantly improved the protocol efficiency by leveraging on pair-wise additive masking and Shamir secret sharing as proposed in the pioneer work SecAgg [9]. Assume there is a set of client $\mathcal{U}$, and let each client $u_i \in \mathcal{U}$ holds a vector $\boldsymbol{x}_i$, the goal is to calculate $\sum \boldsymbol{x}_i$ while preserving the privacy of $\boldsymbol{x}_i$. In [9], a pair-wise additive mask is added to $\boldsymbol{x}_i$ (assume a total order on clients) and the client u_i uploads $\boldsymbol{y}_i$ to the central server rather than $\boldsymbol{x}_i$:

$$\boldsymbol{y}_i = \boldsymbol{x}_i + \sum_{i<j} \mathrm{PRG}(s_{i,j}) - \sum_{i>j} \mathrm{PRG}(s_{j,i})$$

where PRG denotes a pseudorandom generator that is able to generate a sequence of random numbers using the seed $s_{i,j}$. It is obvious from the equation that, when aggregating all the $\boldsymbol{x}_i$, the masks will be canceled such that

$$\sum_{u_i \in \mathcal{U}} \boldsymbol{y}_i = \sum_{u_i \in \mathcal{U}} \left(\boldsymbol{x}_i + \sum_{i<j} \mathrm{PRG}(s_{i,j}) - \sum_{i>j} \mathrm{PRG}(s_{j,i}) \right) = \sum_{u_i \in \mathcal{U}} \boldsymbol{x}_i$$

Furthermore, the seeds are shared among the clients using the standard t-out-of-n Shamir secret sharing scheme (see Section 2.1) to handle the dropout clients. To generate the pair-wise masks, each pair of client (u_i, u_j) are involved with Diffie-Hellman (DH) based key exchange protocol to make an agreement on the seed $s_{i,j}$. Note that, with an n-clients PPML system, running a pair-wise DH protocol is not inexpensive as it has $\mathcal{O}(n^2)$ communication-round complexity. The follow-up works such as [31] and [22] also show its inefficiency. Besides, the runtime of the whole protocol increases rapidly as the fraction of dropped participants (dropout rate) increases, as shown in Figure 1. In this case, active malicious dropping which slows down the system could even be exploited as a kind of attack.

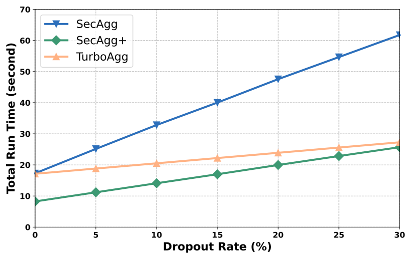


Fig. 1. The runtime of the state-of-the-art aggregation protocols including TurboAgg [43], SecAgg [9] and SecAgg+ [7] with the increase of the dropout rate.

To improve the efficiency of the SecAgg scheme, there are three possible directions proposed by previous works:

- Communicate across only a subset of clients: For example, the variant TurboAgg [43] divides clients into l groups with n_l clients of each group and follows a multi-group circular structure. Both CCESA scheme [14] and SecAgg+ [7] replaces the star topology of the communication network in SecAgg with random subgroups of clients, i.e., a k-regular graph such as Erdos-Renyi graph and Harray graph.
- Optimize the computational complexity: For example, FastSecAgg [26] substitutes standard Shamir secret sharing with a more efficient FFT-based multi-secret sharing scheme.
- Replace pair-wise DH protocol with other schemes: For example, NIKE [32] adopts a non-interactive key exchange protocol.
- Compress the gradient vector: For example, SAFER [6] proposes a MPC-friendly coding method to compress the gradient vector.

However, TurboAgg sacrifices some security for efficiency purposes, compared to SecAgg and SecAgg+. Beside, in both TurboAgg, CCESA and SecAgg+ schemes, each client u shares its secret across only a subset of $\mathcal{U}$, and thus their dropout-resilience regarding Shamir secret sharing are downgraded. Similarly, FastSecAgg achieves a trade-off between the efficiency, privacy and dropout tolerance. NIKE involves two non-colluding servers to construct a 2-out-of-3 Shamir secret sharing scheme, which is not suitable for PPML setting (See Table 1 for comparison). SAFER does not support large-scale systems and has no dropout-resilience. We note that a more generic scheme SAFELearn [16] can be used to construct efficient private FL systems, of which the complexity is determined by its cryptographic instantiation. Therefore, we choose SecAgg as the baseline and SecAgg+ as state-of-the-art for its most client-friendly computation and communication overheads, compared to other above-mentioned works.

In this paper, leveraging on homomorphic pseudorandom generator (HPRG) and Shamir secret sharing scheme, we propose an efficient and dropout-resilient aggregation scheme for PPML. In general, the improved efficiency and dropout-resilience mainly come from the application of HPRG and the sophisticated integration with Shamir secret sharing and the building blocks of SecAgg-based schemes. Specifically, we replace the Diffie-Hellman key exchange protocol for seed agreement in existing schemes with an HPRG-based scheme. In this case, no interaction is needed for the clients to construct the seed used for generating masks, which significantly reduces the communication overheads compared to previous works. Furthermore, since the connected clients do not need to send additional Shamir shares to the server to reconstruct the dropped users' secrets, compared to SecAgg-based schemes, less interaction over clients leads to lower computation and communication overheads. Meanwhile, the server needs only one computation for seed reconstruction rather than the number of dropout clients in SecAgg-based schemes. Thus, our proposed scheme has a stronger dropout-resilience that the runtime of the whole protocol decreases with the increase of the dropout rate.

Compared with existing works, our contributions are as follows:

- The proposed scheme is more efficient, which significantly reduces the communication and computation overheads.
- The proposed scheme has a stronger dropout-resilience that the runtime of the whole protocol decreases with the increase of dropout rate. At the same time, the t-out-of-n Shamir secret sharing scheme is kept, and no extra trusted third party is needed.
- The simplicity of the proposed scheme makes it attractive both for implementation and for further improvements.

Moreover, it can be proven that our scheme is secure against both semi-honest and active malicious adversaries by setting proper system parameters even if a set of clients drops out of the protocol at any time.

2 Underlying Cryptographic Primitives

This section briefly describes the preliminaries of Shamir secret sharing scheme, homomorphic pseudorandom generator, and signature schemes.

2.1 Shamir Secret Sharing

Shamir secret sharing scheme [39] divides a secret S into n pieces of data called shares such that (i) the secret S can be efficiently reconstructed by any combination of t data pieces, and (ii) the secret S cannot be reconstructed by any set of data pieces of which the number is less than t. Such scheme is called a t-out-of-n or (t, n) threshold scheme.

In specific, for a standard (t, n) Shamir secret sharing scheme, the secret S and shares $S_1, \dots, S_n$ are the elements in a finite field $\mathbb{Z}_P$ for some prime P where $0 < t \le n < P$. We assume that there is one secret holder u_s and n participants $\{u_1, \dots, u_n\}$. The scheme works as follows:

1) Preparation: The secret holder u_s randomly chooses $t-1$ positive integers $a_1, \dots, a_{t-1}$ from $\mathbb{Z}_P$ and $a_0 = S$ to define a polynomial of degree $t-1$, i.e., $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{t-1} x^{t-1} \bmod P$.
2) Secret sharing: The secret holder u_s randomly chooses n points $x_1, \dots, x_n$ to retrieve $\{x_i, f(x_i)\}$ for $i \in \{1, 2, \dots, n\}$, and sends them to the corresponding participants u_i.
3) Secret reconstructing: Given any t of $\{x_i, f(x_i)\}$, the secret holder is able to calculate the coefficients $a_0, \dots, a_{t-1}$ of the polynomial $f(x)$ using Lagrange interpolation, and the constant term a_0 is the secret. A more efficient method to directly reconstruct the secret is to calculate $S = a_0 = \sum_{j=0}^{t} f(x_j) \prod_{m=0, m\neq j}^{t} \frac{x_m}{x_m - x_j} \bmod P$.

For simplicity, we denote $\mathcal{F}_{gen}$ as the share generation function to generate shares $\{x_i, f(x_i)\}$, and $\mathcal{F}_{rec}$ as the secret reconstruction function to reconstruct the secret S for (t, n) standard Shamir secret sharing.

Shamir secret sharing scheme is additive homomorphic [19]. For example, assume that the party P_1 has a secret S_1. To secretly share S_1 among n parties $P_1, P_2, \dots, P_n$, the secret holder P_1 chooses a $t-1$ degree polynomial $f(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1} \bmod P$ where $a_0 = S_1$ and $a_1, \dots, a_{t-1}$ are randomly chosen from $\mathbb{Z}_P$, then computes each Shamir share of S_1 to be sent to P_i as $\{x_i, f(x_i)\}$ for $x_i \in X = \{x_1, x_2, \dots, x_n\}$. Similarly, another secret holder P_2 who has the secret S_2 chooses a $t-1$ degree polynomial $g(x) = b_0 + b_1 x + \cdots + b_{t-1} x^{t-1}$, where $b_0 = S_2$ and $b_1, \dots, b_{t-1}$ are randomly chosen from $\mathbb{Z}_P$, then computes n shares of S_2 as $\{x_i, g(x_i)\}$ for $x_i \in X$ that $\{x_i, g(x_i)\}$ is sent to P_i. In this case, each party P_i can locally compute $\{x_i, f(x_i) + g(x_i)\}$, and then cooperates with other parties of which the number is greater than t to reconstruct the secret $S_1 + S_2$ using Lagrange interpolation over $\{x_i, f(x_i) + g(x_i)\}$ for $x_i \in X$. We note that addition over Shamir shares works only when P_1 and P_2 agree on the same set X, which is usually assigned to be $\{1, 2, \dots, n\}$ in Shamir based applications [7, 9, 43]. Besides, when context is clear, we abuse $f(x_i)$ to denote the share rather than $\{x_i, f(x_i)\}$.

2.2 Homomorphic Pseudorandom Generator

A pseudorandom function (PRF) is an efficient algorithm that approximately maps two distinct sets $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ such that a uniform $k \in \mathcal{K}$, a uniform function $f : \mathcal{X} \to \mathcal{Y}$, an oracle for $F(k, \cdot)$ is computationally indistinguishable from an oracle for $f(\cdot)$ [20]. Similar to the definition of PRF, a pseudorandom generator (PRG) is an efficient algorithm that is able to generate a sequence of approximate random numbers. More specifically, PRG is an efficient computable function $G : \mathcal{S} \to \mathcal{Y}$ such that for uniform $s \in \mathcal{S}$ and uniform $y \in \mathcal{Y}$, the distribution $\{G(s)\}$ is computationally indistinguishable from the distribution of $\{y\}$.

A pseudorandom function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is said to be key homomorphic if for any $F(k_1, x)$ and $F(k_2, x)$, an efficient algorithm exists to compute $F(k_1 \oplus k_2, x) = F(k_1, x) \otimes F(k_2, x)$ where both $(\mathcal{K}, \oplus)$ and $(\mathcal{Y}, \otimes)$ are groups. In simple words, the PRF is homomorphic with respect to its key. Similarly, a PRG function $G : \mathcal{S} \to \mathcal{Y}$ is said to be seed homomorphic if for any $G(s_1, x)$ and

TABLE 1
Comparison of the computation complexity, communication complexity, dropout resilience and privacy guarantee between SecAgg [9], TurboAgg [43], CCESA [14], SecAgg+ [7] and FastSecAgg [26]. Here n is the total number of clients, and m is the length of each client's vector. In TurboAgg, n_l is set to be $\log n$ as the group size for the best trade-off. k is set to be $\mathcal{O}(\log n)$ as the degree of each client in k-regular graph in SecAgg and $\mathcal{O}(\sqrt{n/\log n})$ in CCESA, δ is the dropout rate. In FastSecAgg, d can be set up to $\frac{n}{2}$. Note that if the protocol listed in the table provides security against malicious adversaries, its security is also guaranteed against semi-honest adversaries. Since the maximum number of dropout clients is not the same for different privacy guarantees, we only give the comparison in a semi-honest setting. Note that some of complexity analysis are extracted from SAFELearn paper [16]

Protocol		SecAgg [9]	TurboAgg [43]	CCESA [14]	SecAgg+ [7]	FastSecAgg [26]	Our's
Computation complexity	Server	$\mathcal{O}(mn^2)$	$\mathcal{O}(m \log n \log^2 \log n)$	$\mathcal{O}(mn \log n)$	$\mathcal{O}(mn \log n + n \log^2 n)$	$\mathcal{O}(m \log n)$	$\mathcal{O}(n)$
	Client	$\mathcal{O}(n^2 + mn)$	$\mathcal{O}(m \log n \log^2 \log n)$	$\mathcal{O}(n\sqrt{n \log n} + mn)$	$\mathcal{O}(m \log n + \log^2 n)$	$\mathcal{O}(m \log n)$	$\mathcal{O}(n^2 + m)$
Communication complexity	Server	$\mathcal{O}(n^2 + mn)$	$\mathcal{O}(mn \log n)$	$\mathcal{O}(n \log n + m\sqrt{n \log n})$	$\mathcal{O}(mn + n \log n)$	$\mathcal{O}(n^2 + mn)$	$\mathcal{O}(n^2 + mn)$
	Client	$\mathcal{O}(m + n)$	$\mathcal{O}(m \log n)$	$\mathcal{O}(\sqrt{n \log n + m})$	$\mathcal{O}(m + \log n)$	$\mathcal{O}(m + n)$	$\mathcal{O}(m + n)$
Dropout resilience	Scheme	(t, n)	$(\frac{n_l}{2}, n_l)$	(t, k)	(t, k)	$(n - d, n)$	(t, n)
	Max drop	$n - 1$	$\frac{n}{2} - 1$	δn	δn	$\frac{n}{2} - 1$	$n - 1$
Communication rounds		4	$n/\log n$	3	3	3	3
Privacy		malicious	semi-honest	semi-honest	malicious	semi-honest	malicious

$G(s_2, x)$, we have $G(s_1 \oplus s_2, x) = G(s_1, x) \otimes G(s_2, x)$. Such type of PRG is called homomorphic PRG (HPRG).

As described in [5], constructing an HPRG is quite straightforward in random oracle model. Let $(\mathbb{G}, q)$ be a finite cyclic group of prime order q, and $H : \mathcal{X} \to \mathbb{G}$ be a hash function modeled as random oracle, and the function $F : \mathbb{Z}_q \times \mathcal{X} \to \mathbb{G}$ be

$$F(k, x) = H(x)^k$$

We can observe that F is key homomorphic:

$$F(k_1 + k_2, x) = F(k_1, x) \cdot F(k_2, x)$$

It is proved in [35] that if the Decision Diffie-Hellman (DDH) assumption holds in $\mathbb{G}$, such function F is secure in random oracle model that if k is uniform in $\mathcal{K}$, then $F(k, \cdot)$ is indistinguishable from a random sample in $\mathbb{G}$. Therefore, a simple version of HPRG can be constructed by using the seed s for k and indices for x. In specific, HPRG $G(s)$ can generate a sequence of elements, i.e., $F(s, 1), F(s, 2), F(s, 3)$.... The length of such sequence is determined by the vector to be masked. For example, if the input vector $\boldsymbol{x} = [x_1, x_2, ..., x_m]$ has m entries, then $G(s)$ generates $F(s, 1), F(s, 2), ..., F(s, m)$, and the masking operations are done element-wisely.

2.3 Signature Scheme

A signature scheme is used to prove the origin of a message. If a message is signed by Alice's secret key, then the message must have come from Alice. A signature scheme is a tuple of algorithm $(\mathcal{F}_{kg}, \mathcal{F}_{sig}, \mathcal{F}_{vrfy})$ such that

- $\mathcal{F}_{kg}$ is a randomized algorithm that outputs a secret key sk and a public key pk.
- $\mathcal{F}_{sig}$ is a randomized algorithm that receives the secret key sk and a message m and outputs a signature σ.
- $\mathcal{F}_{vrfy}$ is a deterministic algorithm that takes in the public key pk, the message m, and the signature σ, and returns one if σ is a signature on m and 0 otherwise.

The signature scheme achieves security against universal forgery under chosen message attack (UF-CMA). It means that someone without the secret key can not create a valid signature on a message he has not seen signed before. In other words, the probability of the adversary to construct a pair (m^*, σ^*) without knowing the secret key sk that σ^* is a valid signature on m^* and m^* has never been seen before is negligible.

3 High-level Overview

Similar to the SecAgg scheme [9], we divide participants into two classes: (i) a central server $\mathcal{S}$ that acts as a coordinator to aggregate inputs from n clients $\mathcal{U}$, and (ii) each client $u_i \in \mathcal{U}$ holds a locally trained model $\boldsymbol{x}_i$. The goal of our scheme is that the server can compute the sum of clients' models as $z = \sum_{u_i \in \mathcal{U}} \boldsymbol{x}_i$ while keeping the privacy of $\boldsymbol{x}_i$ only to the client u_i. Besides, the scheme should be dropout-resilient as clients may drop out at any time.

Threat model. In many PPML applications such as FL, the participants can be individual users or competitive business entities and are required to comply with the data privacy regulations. Besides, some participants may cooperate in invading other's data privacy for their benefit. Due to these natures, we consider two threat models. In the first threat model, adversary participants are semi-honest that will not deviate from the protocol but tries to infer the honest parties' information. In the second threat model, adversary participants can be active malicious and may collude and send fraudulent messages to others. We note that our proposed protocol is secure against semi-honest adversaries, and provides additional security against active malicious adversaries by adopting specific security protocols of which the number of adversaries has an upper bound (see Section 4.2 for the details). Our security definition is based on the Universal Composability (UC) framework, and we refer interested readers to [11] for the details.

3.1 Masking Models

Since we aim to protect the clients' locally trained models from information leakage, the most intuitive method is to mask the model using an one-time pad before the models are submitted to the central server. Note that the mask is added to the model element-wisely to permute the model's distribution. Otherwise, since the range of the model's elements is known for a fixed ML model, adversaries can easily reconstruct the original model if all the model's elements are masked by the same random value. Let the client u_i's model be $\boldsymbol{x}_i$, and its self-generated mask be $\boldsymbol{r}_i$, then client u_i's masked model can be locally calculated by:

$$\boldsymbol{y}_i = \boldsymbol{x}_i + \boldsymbol{r}_i$$

After that, the client sends $\boldsymbol{y}_i$ to the server. Assume that the server already had the sum of $\boldsymbol{r}_i$ for all the clients as $\boldsymbol{R} = \sum_{u_i \in \mathcal{U}} \boldsymbol{r}_i$, the server can compute:

$$\boldsymbol{z} = \sum_{u_i \in \mathcal{U}} \boldsymbol{y}_i - \boldsymbol{R} = \sum_{u_i \in \mathcal{U}} \boldsymbol{y}_i - \sum_{u_i \in \mathcal{U}} \boldsymbol{r}_i = \sum_{u_i \in \mathcal{U}} \boldsymbol{x}_i$$

The naive method to get $\boldsymbol{R}$ is to let all the clients send their masks $\boldsymbol{r}_i$ to the server for aggregation. However, this leaks information about $\boldsymbol{x}_i$ as the server can easily compute $\boldsymbol{x}_i = \boldsymbol{y}_i - \boldsymbol{r}_i$. To deal with this issue, the intuitive idea is to let the clients send partial information of $\boldsymbol{R}$ as $\boldsymbol{R}^i$ rather than $\boldsymbol{r}_i$, which can be used to reconstruct $\boldsymbol{R}$ by computing $\boldsymbol{R} = \mathcal{F}(\boldsymbol{R}^1, \ldots, \boldsymbol{R}^n)$ where $\mathcal{F}$ is the reconstruction function. Note that $\boldsymbol{R}^i$ and the process of computing $\boldsymbol{R}^i$ and $\boldsymbol{R}$ should not leak any information of $\boldsymbol{r}_i$. For example, if $\mathcal{F}$ is defined as summation and

$$\boldsymbol{R}^i = \sum_{u_j \in \mathcal{U}} \boldsymbol{r}_j^i, \text{ where } \boldsymbol{r}_j = \sum_{u_i \in \mathcal{U}} \boldsymbol{r}_j^i$$

Here, $\boldsymbol{r}_j^i$ is the partial information of $\boldsymbol{r}_j$ hold by client u_i. Then, we can have:

$$\boldsymbol{R} = \sum_{u_j \in \mathcal{U}} \boldsymbol{r}_j = \sum_{u_i \in \mathcal{U}} \boldsymbol{R}^i$$

Since the client u_i knows only $\boldsymbol{r}_j^i$, and server $\mathcal{S}$ knows only $\boldsymbol{R}^i$, none of them can learn $\boldsymbol{r}_i$ nor $\boldsymbol{x}_i$. Note that the scheme mentioned above cannot tolerate dropout clients as the server can reconstruct $\boldsymbol{R}$ only when all the clients are connected, i.e., all the $\boldsymbol{R}^i$ are received by the server.

3.2 Handling dropout clients

To handle the scenario where the clients may drop out during aggregation, the server should be able to learn the sum of $\boldsymbol{x}_i$ from only a fraction of the clients. In specific, if there are $n - t$ dropout clients, i.e., t connected users, the server can still reconstruct $\boldsymbol{R}$ from $\boldsymbol{R}^i$ from connected clients. Such scheme can be achieved by using a t-out-of-n Shamir secret sharing scheme.

Following the description in Section 2.1, we assume a total order on clients. For each client u_j, let $\mathcal{F}_{gen}$ be the share generation function and $\mathcal{F}_{rec}$ be the secret reconstruction function for (t, n) standard Shamir secret sharing that $\mathcal{F}_{gen}(\boldsymbol{r}_j) \rightarrow \left\{i, \boldsymbol{r}_j^i\right\}_{u_i \in \mathcal{U}}$ where $i \in \{1, 2, \ldots, n\}$. Here $\boldsymbol{r}_j^i$ is the share of $\boldsymbol{r}_j$ hold by client u_i such that $\mathcal{F}_{rec}(\left\{i, \boldsymbol{r}_j^i\right\}_{u_i \in \mathcal{V}}) \rightarrow \boldsymbol{r}_j$ where $|\mathcal{V}| \geq t$. Then, each client u_i can locally compute $\boldsymbol{R}^i$ as the share of $\boldsymbol{R}$ that

$$\boldsymbol{R}^i = \sum_{u_j \in \mathcal{U}} \boldsymbol{r}_j^i \bmod P$$

Therefore, relying on the additive homomorphic property of Shamir scheme (see Section 2.1), if the number of received $\boldsymbol{R}^i$ is greater than the threshold t, the server can reconstruct $\mathcal{F}_{rec}(\left\{i, \boldsymbol{R}^i\right\}_{u_i \in \mathcal{V}}) \rightarrow \boldsymbol{R}$ where $|\mathcal{V}| \geq t$.

3.3 More efficiently generating masks

We notice that in the SecAgg scheme [9], the communication cost can be further reduced by having the clients agree on common seeds or keys. In that case, the mask can be generated by a common seed using a pseudorandom generator (PRG). Thus each pair of clients only need to transmit one seed rather than the entire mask. A similar approach can also be applied to our scheme by having each client u_i hold a seed s_i such that the mask vector $\boldsymbol{r}_i = \text{PRG}(s_i)$. However, this trick does not work for our scheme as

$$\text{PRG}(\sum_{u_i \in \mathcal{U}} s_i) \neq \sum_{u_i \in \mathcal{U}} \text{PRG}(s_i)$$

which means

$$\text{PRG}(\sum_{u_i \in \mathcal{U}} s_i) = \boldsymbol{R} \neq \sum_{u_i \in \mathcal{U}} \boldsymbol{r}_i = \sum_{u_i \in \mathcal{U}} \text{PRG}(s_i)$$

In this case, since $\boldsymbol{y}_i = \boldsymbol{x}_i + \boldsymbol{r}_i$, the server cannot correctly cancel the mask to get the real aggregation results by computing $\boldsymbol{z} = \sum_{u_i \in \mathcal{U}} \boldsymbol{y}_i - \boldsymbol{R}$. Luckily, relying on the homomorphic pseudorandom generator (HPRG), this trick still works to reduce the communication costs. As illustrated in Section 2.2, additive homomorphism holds for HPRG such that for any two seeds s_a, $s_b \in \mathcal{K}$, we have

$$\text{HPRG}(s_a + s_b) = \text{HPRG}(s_a) \cdot \text{HPRG}(s_b)$$

Note that HPRG is constructed by using the structure introduced in Section 2.2 such that

$$\text{HPRG}(k) \rightarrow [F(k, 1), F(k, 2), F(k, 3), ..., F(k, m)]$$

where $F(k, x) = H(x)^k$ and m is the vector size of $\boldsymbol{x}_i$. Therefore, in our scheme:

$$\text{HPRG}(\sum_{u_i \in \mathcal{U}} s_i) = \prod_{u_i \in \mathcal{U}} \text{HPRG}(s_i)$$

and component-wisely:

$$\boldsymbol{R} = \prod_{u_i \in \mathcal{U}} \boldsymbol{r}_i = \prod_{u_i \in \mathcal{U}} \text{HPRG}(s_i)$$

To keep the consistency between the operations regarding HPRG and our scheme, we adapt the masking as:

$$\boldsymbol{y}_i = g^{\boldsymbol{x}_i} \cdot \boldsymbol{r}_i$$

Here, $\boldsymbol{r}_i$ is randomly picked from the finite cyclic group $\mathbb{G}$ of which q is the order and g is the generator that all clients agree on. Note that since the distribution of $\boldsymbol{y}_i$ is identical with that of $\boldsymbol{r}_i$, the multiplicative mask $\boldsymbol{r}_i$ still guarantees the security. In other words, the mask $\boldsymbol{r}_i$ hides all information about $\boldsymbol{x}_i$. Besides, to keep the aggregation results meaningful, the value of P for the Shamir scheme and q for the finite cyclic group $\mathbb{G}$ need to satisfy that $P > q > n * \max(x_i)$ in order to avoid the overflow where n is the number of clients, i.e., the number of vectors to be aggregated. As a result, with the sum of $\boldsymbol{y}_i$ and the product of $\boldsymbol{r}_i$, we can compute

$$g^{\boldsymbol{z}} = \prod_{u_i \in \mathcal{U}} \boldsymbol{y}_i / \boldsymbol{R} = \prod_{u_i \in \mathcal{U}} \boldsymbol{y}_i / \prod_{u_i \in \mathcal{U}} \boldsymbol{r}_i = g^{\sum_{u_i \in \mathcal{U}} \boldsymbol{x}_i}$$

Because the input range of x_i is fixed which is not very large, computing the discrete logarithms of $g^{\boldsymbol{z}} = g^{\sum_{u_i \in \mathcal{U}} \boldsymbol{x}_i}$ base g to decrypt the sum $\sum_{u_i \in \mathcal{U}} \boldsymbol{x}_i$ is affordable. By using Pollard's lambda method [33], it requires roughly square root of time in the plaintext space. If the plaintext range of each element of $\boldsymbol{x}_i$ is in $\{0, 1, 2, \ldots, n\alpha\}$, computing $\boldsymbol{x}_i$ requires roughly $\sqrt{n\alpha}$ time. Such consideration is practical in privacy-preserving aggregation for FL as pointed out in [41]. For example, for 64-bit plaintext space (enough for many image classification tasks), computing a 1024-bit discrete logarithms with $n = 500$ for a vector with 50K entries takes about 12.5 seconds on a 64-bit server. Note that in this section, we only describe the high-level overview and omit some details for simplicity, and thus we refer the readers to Section 3.5 for the full specification.

3.4 Putting it all together

We summarize the protocols regarding masking as follows:

1 Each client u_i randomly selects a seed $s_i \in \mathcal{K}$.

2 Using (t, n) Shamir secret sharing scheme, each client u computes n shares of s_i as $\mathcal{F}_{gen}(s_i) \rightarrow \left\{j, s_i^j\right\}_{u_j \in \mathcal{U}}$ and then sends $\{j, s_i^j\}$ to the client $u_j \in \mathcal{U}$.

3 Each client u_i computes the masked update $y_i = g^{\boldsymbol{x}_i} \cdot \boldsymbol{r}_i$ and sends it to the server. Then the server receives the updates from all connected clients (denoted as $\mathcal{V}$), and computes $\prod_{u_i \in \mathcal{V}} \boldsymbol{y}_i = g^{\sum_{u_i \in \mathcal{V}} \boldsymbol{x}_i} \cdot \prod_{u_i \in \mathcal{V}} \boldsymbol{r}_i = g^{\sum_{u_i \in \mathcal{V}} \boldsymbol{x}_i} \cdot \boldsymbol{R}$.

4 Each client u_j locally computes s_R^j as the share of seed s_R for generating $\boldsymbol{R}$ that $s_R^j = \sum_{u_i \in \mathcal{V}} s_i^j$, and sends s_R^j to the server.

5 The server reconstructs s_R under (t, n) Shamir secret sharing scheme that $\mathcal{F}_{rec}(\left\{j, s_R^j\right\}_{u_j \in \mathcal{V}}) \rightarrow s_R$, then calculates the mask $\boldsymbol{R}$ using HPRG such that $\boldsymbol{R} = \mathrm{HPRG}(s_R)$.

Note that the list of connected clients $\mathcal{V}$ is kept by the server. The clients need to fetch the list of connected clients before calculating s_R^j. We refer readers to the Section 3.5 for the details. We can observe from the proposed protocol that to mask the clients' model, we rely on the additive property of the Shamir scheme rather than the pair-wise masking as used in SecAgg and SecAgg+. Thus, the communication overheads are significantly reduced. Furthermore, as long as a sufficient number of Shamir shares of the final mask have been collected by the server, the connected clients do not need to send additional Shamir shares to the server, compared to SecAgg-based schemes, which leads to a stronger dropout-resilience with respect to the efficiency.

3.5 Proposed secure aggregation protocol

Our aggregation protocol involves one single server and a set of n clients. Each client u_i has an input vector $\boldsymbol{x}_i$ as its locally trained model or gradient. The vector $\boldsymbol{x}_i$ consists of m elements from field $\mathbb{Z}_q$ for some q. The HPRG is under DDH assumption regarding the algorithm $(\mathbb{G}, g, q, H)$ for HPRG which samples a group $\mathbb{G}$ of order q with generator g and Hash function H (see Section 2.2). Similar to [7, 9], the communication channels between the server and clients are assumed to be synchronous, which means that message delivery time is bounded, e.g., if the server does not receive an uploaded model from a client within a time limit, it can assume that this client drops out of the system [27]. For simplicity, we assume a public-key crypto-system between any pair of clients, and abuse the notation and use $Enc(msg, pk)$ and $Dec(msg, sk)$ as the encryption and decryption on message msg using the public key pk and the secret key sk. The clients may drop out of the protocol at any time. However, if the number of connected clients is greater than a threshold t, the server can still learn the correct output as the aggregation result.

The detailed description of our protocol is given in Protocol 1. Specifically, if a client u_k has uploaded the masked model $\boldsymbol{y}_k$ in Step 2 and dropped out in the following steps, $\boldsymbol{y}_k$ still can be unmasked correctly and contribute to the aggregated model as the client has already shared its seed s_k with other clients in Step 1. In other words, when more than t clients send the shares of $\boldsymbol{R}$ to the server, i.e., $|\mathcal{U}_4| > t$, the server can compute an aggregation of all the models uploaded by clients in U_2 (U_3 under malicious threat model). Note that if the client u_k has shared the seed s_k successfully but drops out before sending its masked model, i.e., $u_k \in \mathcal{U}_1 \setminus \mathcal{U}_2$, the mask vector $\boldsymbol{r}_k$ of the client u_k will not be included in $\boldsymbol{R}$. As shown in Step 4, each connected client only computes the sum of shares of the seeds getting from the clients in $\mathcal{U}_2$. In this scenario, the share of client u_k's seed s_k is not added and thus $\boldsymbol{r}_k$ will not be reconstructed by the server. Like other mechanisms based on Shamir secret sharing [7, 9], to reconstruct secrets, the proposed protocol is inevitably subject to delay since clients have to wait for the $\mathcal{U}_2$ list, which contains the identifier of the clients from whom the server has received the masked models. Nevertheless, as the protocol is assumed to run on a synchronous channel as mentioned earlier, this delay is bounded and the client is considered dropped out of the protocol if it does not upload the masked model within a predefined time limit.

We can observe that compared with the SecAgg scheme, since the most communication-intensive building block, i.e., the pair-wise DH key agreement protocol for seed agreement, is removed, our scheme achieves a much simpler structure.

4 Security Analysis

In this section, we provide the security claims along with their proofs for the protocols proposed in Section 3.5. Recall that the involved participants are a single central server $\mathcal{S}$ and a set of clients $\mathcal{U}$ with their locally trained models $x_{\mathcal{U}}$. We consider that the central server is always online while the clients may abort, e.g., drop out, from the protocol at any point. We denote the connected clients in each step as $\mathcal{U}_i$ from the receivers' angle (refer to Protocol 1). The underlying cryptographic building blocks are instantiated with the security parameter κ.

We assume a group of adversaries consisting of a subset of clients whose number is less than a threshold t, and

Protocol: HPRG based Secure Aggregation

Participants: A single central server $\mathcal{S}$ and a set of clients $\mathcal{U}$.
Private inputs: Each client u_i has a locally trained model or gradient represented as a vector $\boldsymbol{x}_i$, a secret key for constructing authenticated channels csk_i, and a secret key for signature ssk_i. The server has a secret key for authenticated channels with clients csk_s and a secret key for signature ssk_s.
Public inputs: The number of clients $n = |\mathcal{U}|$, the threshold $t < n$, the field $\mathbb{Z}_P$ for some P for Shamir secret sharing scheme with function $\mathcal{F}_{gen}$ and $\mathcal{F}_{rec}$, the algorithm $(\mathbb{G}, g, q)$ for HPRG which samples a finite cyclic group $\mathbb{G}$ of prime order q with generator g, and the security parameter κ. Each client u_i's and server's public key for constructing secure channels cpk_i, cpk_s, and their public keys for signature spk_i, spk_s. Note that $P > q > n * \max(x_i)$.
Outputs: The aggregation result of the locally trained models from the set of connected clients $\mathcal{U}_2 \subseteq \mathcal{U}$: $\sum_{u_i \in \mathcal{U}_2} \boldsymbol{x}_i$ ($\mathcal{U}_3$ under malicious threat model)

---

* **Step 1 - Sharing seeds:**
Client u_i:
a) The client randomly picks s_i from $\mathbb{Z}_q$ where q is the order of the finite cyclic group $\mathbb{G}$ from which all clients agree on a generator g. Generates (t, n) Shamir secret shares of $s_i \in \mathbb{Z}_P$, i.e., $\mathcal{F}_{gen}(s_i) \rightarrow (\left\{j, s_i^j\right\}_{u_j \in \mathcal{U}})$
b) Sends $(Enc(s_i^j, cpk_j), \sigma_{i,j}^1)$ to client $u_j \in \mathcal{U}$, where the signature $\mathcal{F}_{sig}(Enc(s_i^j, cpk_j), ssk_i) \rightarrow \sigma_{i,j}^1$ (Denote $\mathcal{U}_1$ as the set of clients $u_i \in \mathcal{U}$ that at least t shares of s_i have been received by other clients).
c) Receives $(Enc(s_j^i, cpk_i), \sigma_{j,i}^1)$ from all the clients $u_j \in \mathcal{U}$, and then computes $s_j^u = Dec(Enc(s_j^u, cpk_i), csk_i)$. If $\mathcal{F}_{vrfy}(Enc(s_j^i, cpk_i), spk_j, \sigma_{j,i}^1) = 0$, aborts.

* **Step 2 - Collecting masked models:**
Client u_i:
a) Generates the mask vector $\boldsymbol{r}_i$ using HPRG with the seed s_i, i.e., $\boldsymbol{r}_i = \text{HPRG}(s_i)$.
b) Computes the model masked by $\boldsymbol{r}_i$, i.e., $\boldsymbol{y}_i = g^{\boldsymbol{x}_i} \cdot \boldsymbol{r}_i$.
c) Sends $\boldsymbol{y}_i$ with the signature $\mathcal{F}_{sig}(\boldsymbol{y}_i, ssk_i) \rightarrow \sigma_i^2$ to the server $\mathcal{S}$.
Server $\mathcal{S}$:
a) Receives all $\boldsymbol{y}_i$ with the signature σ_i^2 from clients (denote with $\mathcal{U}_2 \subseteq \mathcal{U}_1$ this set of clients). If $\mathcal{F}_{vrfy}(\boldsymbol{y}_i, spk_i, \sigma_i^2) = 0$, remove client u_i from $\mathcal{U}_2$.

* **Step 3 - Checking consistency:**
Client u_i:
a) Fetches the list of $\mathcal{U}_2$ from the server $\mathcal{S}$ with the server's signature $\mathcal{F}_{sig}(\mathcal{U}_2, ssk_s) \rightarrow \sigma_s^3$. If $\mathcal{F}_{vrfy}(\mathcal{U}_2, spk_s, \sigma_s^3) = 0$, aborts.
b) Sends $\mathcal{F}_{sig}(\mathcal{U}_2, ssk_i) \rightarrow \sigma_i^4$ to the server $\mathcal{S}$.
Server $\mathcal{S}$:
a) Receives σ_i^4 from at least t clients (Denote with $\mathcal{U}_3 \subseteq \mathcal{U}_2$ this set of clients) and forwards to the clients in $\mathcal{U}_3$.

* **Step 4 - Unmasking:**
Client u:
a) If the protocol does not consist of step 3 for consistency checking, fetches the list of $\mathcal{U}_2$ from the server $\mathcal{S}$. Otherwise, if $|\mathcal{U}_3| < t$ or for all $u_j \in \mathcal{U}_3$, $\mathcal{F}_{vrfy}(\mathcal{U}_2, spk_j, \sigma_j^4) = 0$, aborts.
b) Computes the sum of shares of s_j^i from all the clients $u_j \in \mathcal{U}_2$, i.e., $s_R^i = \sum_{u_j \in \mathcal{U}_2} s_j^i \bmod P$. ($\mathcal{U}_3$ under malicious threat model)
c) Sends s_R^i with the signature $\mathcal{F}_{sig}(Enc(s_R^i, cpk_s), ssk_i) \rightarrow \sigma_j^5$ to the server $\mathcal{S}$.
Server $\mathcal{S}$:
a) Receives s_R^j from the clients (Denote with $\mathcal{U}_4$ this set of clients). If $\mathcal{F}_{vrfy}(Enc(s_R^j, cpk_s), spk_j, \sigma_j^5) = 0$, remove client j from $\mathcal{U}_4$. Proceed until $|\mathcal{U}_4| > t$.
b) Reconstructs the seed for unmasking, i.e., $\mathcal{F}_{rec}(\left\{j, s_R^j\right\}_{u_j \in \mathcal{U}_4}) \rightarrow s_R \bmod P$.
c) Generates the mask $\boldsymbol{R}$ using HPRG with the seed s_R, i.e., $\boldsymbol{R} = \text{HPRG}(s_R)$.
d) Computes $g^{\boldsymbol{z}} = \prod_{u \in \mathcal{U}_2} \boldsymbol{y}_i / \boldsymbol{R} = \prod_{u_i \in \mathcal{U}_2} \boldsymbol{y}_i / \prod_{u_i \in \mathcal{U}_2} \boldsymbol{r}_i = g^{\sum_{u_i \in \mathcal{U}_2} \boldsymbol{x}_i}$. ($\mathcal{U}_3$ under malicious threat model)
e) Computes and outputs $\boldsymbol{z} = \log_g(g^{\boldsymbol{z}}) = \sum_{u_i \in \mathcal{U}_2} \boldsymbol{x}_i$. ($\mathcal{U}_3$ under malicious threat model)

Protocol 1. Detailed description of HPRG based Secure Aggregation protocol for one FL round. The underlined parts are only for active malicious threat model.

with or without the central server. The security definition requires that any group of adversaries will learn nothing about the remaining clients' values. For example, if the threshold $t > 2$, a group of adversaries consisting of client u_1, u_2 and the central server $\mathcal{S}$, they will not learn any information about u_3's locally trained model. More specifically, given any subset $\mathcal{C} \subseteq \mathcal{U}$ of the adversaries where $|\mathcal{C} \setminus \{\mathcal{S}\}| < t$, the resulting values of honest participants $\mathcal{U} \setminus \mathcal{C}$ should look uniformly random.

Let $\mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,\kappa}(x_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3)$ be a random variable representing the joint views of participants in $\mathcal{C}$ in real execution of our proposed protocol, and $\mathsf{SIM}_{\mathcal{C}}^{\mathcal{U},t,\kappa}(x_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3)$ be another combined views of participants in $\mathcal{C}$ simulating the protocol that the inputs of honest participants are selected randomly and uniformly denoted with $x_{\mathcal{C}}$. Following above-mentioned idea, the distribution of $\mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,\kappa}(x_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3)$ and $\mathsf{SIM}_{\mathcal{C}}^{\mathcal{U},t,\kappa}(x_{\mathcal{C}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3)$ should be indistinguishable.

4.1 Semi-honest Model

We consider two cases: (i) a subset of clients are semi-honest colluding adversaries, and the server is honest (ii) the server is additionally semi-honest adversarial and colludes with a subset of clients. We provide the security claims along with the proofs in Theorem 4.1 and Theorem 4.2 respectively.

Theorem 4.1 (Security against semi-honest clients, with honest server). *For all $\mathcal{U}, t, \kappa$ with $|\mathcal{C}| < t$, $x_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3$, and $\mathcal{C}$ where $\mathcal{C} \subseteq \mathcal{U}$ and $\mathcal{U}_3 \subseteq \mathcal{U}_2 \subseteq \mathcal{U}_1 \subseteq \mathcal{U}$, there exists a probabilistic polynomial-time (PPT) simulator* SIM *such that*

$$\mathsf{SIM}_{\mathcal{C}}^{\mathcal{U},t,\kappa}(x_{\mathcal{C}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3) \equiv \mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,\kappa}(x_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3)$$

where $\equiv$ denotes that the distributions are identical.

Proof. Since the server is honest, the combined views of the participants in $\mathcal{C}$ are independent of that of the participants who are not in $\mathcal{C}$. This means by letting all semi-honest participants having their actual inputs and other participants having dummy inputs, the SIM can perfectly simulate the views of the participants in $\mathcal{C}$. As only the list of specific participants will be revealed to semi-honest participants, the simulator SIM can set the message uploaded from honest participants who are not in $\mathcal{C}$ as dummy values. Therefore, the simulated combined views of the participants in $\mathcal{C}$ is identical to that in REAL. □

Theorem 4.2 (Security against semi-honest adversaries, including the server). *For all $\mathcal{U}, t, \kappa$ with $|\mathcal{C} \setminus \{\mathcal{S}\}| < t$, $x_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3$, and $\mathcal{C}$ where $\mathcal{C} \subseteq \mathcal{U} \cup \{\mathcal{S}\}$ and $\mathcal{U}_3 \subseteq \mathcal{U}_2 \subseteq \mathcal{U}_1 \subseteq \mathcal{U}$, there exists a probabilistic polynomial-time (PPT) simulator* SIM *such that*

$$\mathsf{SIM}_{\mathcal{C}}^{\mathcal{U},t,\kappa}(x_{\mathcal{C}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3) \equiv \mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,\kappa}(x_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3)$$

where $\equiv$ denotes that the distributions are identical.

Proof. We use a standard hybrid argument to prove the theorem. We define a sequence of hybrid distributions $H_0, H_1, \ldots$ to construct the simulator SIM by the subsequent modifications to the random variable REAL. In other words, if any two subsequent hybrids are computationally indistinguishable, the distribution of simulator SIM as a whole is also identical to the real execution REAL.

* H_0: In this hybrid, the distribution of the combined views of $\mathcal{C}$ of SIM is exactly the same as that of REAL.
* H_1: In this hybrid, for each client $u_i \in \mathcal{U}_1 \setminus \mathcal{C}$, we replace s_i^j, i.e., the share of simulated honest client u_i's seed s_i that sent to adversarial client u_j, with a randomly selected element in the corresponding field. Note that since the adversaries in $\mathcal{C}$ do not receive any additional shares of s_i where $u_i \in \mathcal{U}_1 \setminus \mathcal{C}$, the combined view of adversaries has only $|\mathcal{C} \setminus \{\mathcal{S}\}| < t$ shares of each seed s_i. According to the security property of the Shamir secret sharing scheme, the adversaries learn nothing about the seed s_i. Therefore, the distribution of this hybrid is identical to the previous one.
* H_2: In this hybrid, instead of computing the mask $\boldsymbol{r}_i$ using HPRG, the mask $\boldsymbol{r}_i$ of each simulated client u_i is replaced with a randomly selected number in the appropriate length. Since in the previous hybrid, the seed s_i is chosen uniformly and randomly by letting its shares be selected uniformly at random, and the adversaries' seeds are set to be 0, the output of HPRG does not depend on its seed. Therefore, the security of HPRG leveraging the Decisional Diffie-Hellman assumption guarantees the identical distribution of this hybrid to the previous one.
* H_3: In this hybrid, we substitute each of the honest clients u_i's masked model $\boldsymbol{y}_i$ with a uniformly random value. Since in the previous hybrid, $\boldsymbol{r}_i$ is chosen uniformly at random and is used as a one-time pad to mask $\boldsymbol{x}_i$, it is obvious that SIM can simulate REAL without knowing any information about $\boldsymbol{x}_i$, and thus this hybrid is identically distributed to the previous one.
* H_4: In this hybrid, for the honest clients $u_i \in \mathcal{U}_2 \setminus \mathcal{C}$, instead of sending $g^{\boldsymbol{x}_i} \cdot \boldsymbol{r}_i \rightarrow \boldsymbol{y}_i$, we send $g^{\boldsymbol{w}_i} \cdot \boldsymbol{r}_i \rightarrow \boldsymbol{y}_i$ where $\boldsymbol{w}_i$ is uniformly sampled at random from $\mathbb{Z}_q$, subject to

$$\sum_{u_i \in \mathcal{U}_2 \setminus \mathcal{C}} \boldsymbol{w}_i = \sum_{i \in \mathcal{U}_2 \setminus \mathcal{C}} \boldsymbol{x}_i \bmod q$$

  We can observe that the distribution of $g^{\boldsymbol{x}_i} \cdot \boldsymbol{r}_i$ is identical to that of $g^{\boldsymbol{w}_i} \cdot \boldsymbol{r}_i$ subject to the above equation.

By defining such PPT simulator SIM as described in the last hybrid, the semi-honest adversaries' combined views of SIM is computationally indistinguishable from that of the real execution REAL, and thus the proof is completed. □

4.2 Active Malicious Model

Next, we discuss the security of the active malicious threat model. Note that in such a threat model, correctness cannot be guaranteed as active adversaries $\mathcal{C}$ can deviate from the protocol at any time by sending fraudulent messages, distorting the outputs, etc. In this case, only the privacy of honest clients' inputs is considered to be guaranteed. The difference between the semi-honest model and malicious model for our protocol can be summarized as follows:

- The adversaries $\mathcal{C}$ can simulate a specific honest client u, and thus receive all the related information about u to recover u's inputs. This malicious behavior is so-called the Sybil attack.
- The malicious server $\mathcal{S}$ can actively send a different list of connected clients to the honest clients. For example, the server sends $\mathcal{U}$ to client u, and $\mathcal{V} \subseteq \mathcal{U}$ to client v. Then during the execution of the protocol, the information

about $\mathcal{U} \setminus \mathcal{V}$ would be leaked, which can be used to recover the private inputs of honest clients' from $\mathcal{U} \setminus \mathcal{V}$.

- The malicious participants (server) is able to dynamically set any honest client to be dropped out from the protocol at any round of protocol execution, and thus the proof for the semi-honest threat model is no longer correct. The reason is that the simulator SIM knows only the sum of honest clients' inputs. If some honest clients are set to be dropped, the SIM cannot simulate the rest of honest clients' behaviors as it knows nothing about the private inputs of dropped clients.

The first difference regarding the Sybil attack can be solved by using a standard signature scheme (see Section 2.3) which can be used to prove the origin of a message. In specific, a message signatured by client u must have come from u, and its origin can be verified by any participants. Thus the messages sent from honest clients cannot be modified or substituted by malicious adversaries.

However, even with such authenticated channels, the malicious server can still give a different view of dropped clients (connected clients) to the honest clients for malicious purposes, as pointed out in the second difference. Therefore, we have to check the consistency between the list of connected clients sent to each client. In general, after receiving the list of the connected clients $\mathcal{V}$ from the server, each client u_i in $\mathcal{V}$ generates a signature σ_i on $\mathcal{V}$ and sends it to the server. Then the server forwards all the σ_i to the clients in $\mathcal{V}$ to have them checking the consistency between $\mathcal{V}$ and $\{\sigma_i\}_{u_i \in \mathcal{V}}$, i.e., to verify if each σ_i is actually the client u_i's signature on $\mathcal{V}$. As a result, the same view of connected clients lists to all the clients in $\mathcal{V}$ can be guaranteed. Note that the consistency check costs one communication round, which may cause more dropped clients, and thus needs to maintain an extra list of connected clients (see the details in Protocol 1).

For the last difference, we take a similar approach in [9] to adopt the proof to be performed in random oracle (RO). In such an RO model, the simulator SIM is able to send a query to an ideal functionality to learn the sum of a dynamically selected subset of honest clients. In other words, by reprogramming the RO such that the subset of honest clients is chosen dynamically, the combined view of adversaries in the real protocol execution $\mathsf{REAL}(\mathsf{M}_\mathsf{C})$ is indistinguishable from that of the simulator SIM. Here M_C is a probabilistic polynomial-time algorithm that denotes the "next message" function of participants in $\mathcal{C}$, which enables participants in $\mathcal{C}$ to dynamically choose (i) their inputs at any round of the protocol execution and (ii) the list of connected participants.

We first give an ideal functionality $\mathcal{F}_{HSecAgg}$ in Func. 1 that describes how a fully trusted third party $\mathcal{T}$ would compute each participant's output from the inputs, i.e., calculate the sum of each client's model. In this case, our proposed protocol is secure if a simulator can simulate any information that the malicious adversaries can learn from the protocol in such a way that it is indistinguishable from what they can learn from the ideal functionality $\mathcal{F}_{HSecAgg}$, i.e., $\mathsf{SIM}(\cdot) \equiv \mathsf{REAL}(\cdot)$. To enable the comparison between our scheme and previous schemes, including SecAgg and SecAg+, we provide similar security claims under two settings, i.e., active malicious clients with and without honest server, along with the proofs in Theorem 4.3 and Theorem 4.2, respectively.

Func. 1 Functionality $\mathcal{F}_{HSecAgg}$. HPRG based Secure Aggregation Scheme Overview.

Participants:

• A single central server $\mathcal{S}$ and a set of clients $\mathcal{U}$.

Inputs:

• Private gradient vector of each client $\boldsymbol{x}_i$; Private seed of each client s_i; Public number of clients $n = |\mathcal{U}|$; Public (t, n) Shamir secret sharing scheme.

Outputs:

• The server receives the aggregation result of the gradient vectors from a of clients $\mathcal{U}_3 \subseteq \mathcal{U}$.

```
Trusted party $\mathcal{T}$ executes the following steps:
1: $\mathcal{T}$ receives the seeds from a set of clients $\mathcal{U}_1 \subseteq \mathcal{U}$. If $|\mathcal{U}_1| < t$, aborts.
2: $\mathcal{T}$ computes and sends the Shamir shares of received seeds and the list of $\mathcal{U}_1$ to the clients in $\mathcal{U}_1$.
3: $\mathcal{T}$ receives the masked models $\boldsymbol{y}_i$, which are constructed based on seeds' Shamir shares and $\boldsymbol{x}_i$, from a set of clients $\mathcal{U}_2 \subseteq \mathcal{U}_1$. If $|\mathcal{U}_2| < t$, aborts.
4: $\mathcal{T}$ sends the list of $\mathcal{U}_3 \subseteq \mathcal{U}_2$ to the clients in $\mathcal{U}_2$. If $|\mathcal{U}_3| < t$, aborts. (This step involves consistency check.)
5: $\mathcal{T}$ receives Shamir shares constructed based on seeds' Shamir shares for unmasking from a set of clients $\mathcal{U}_4 \subseteq \mathcal{U}_3$. If $|\mathcal{U}_4| < t$, aborts.
6: $\mathcal{T}$ calculates the sum of $\boldsymbol{x}_i$ for $\mathcal{U}_3$ based on the received $\boldsymbol{y}_i$ from $\mathcal{U}_3$ and the received Shamir shares from $\mathcal{U}_4$, then send it to the server.
```

Theorem 4.3 (Security against active malicious clients, with honest server)**.** *For all $\mathcal{U}, t, \kappa$ with $|\mathcal{C}| < t$, $x_{\mathcal{U} \setminus \mathcal{C}}$, and $\mathcal{C}$ with the algorithm M_C where $\mathcal{C} \subseteq \mathcal{U}$, the protocol 1 is a secure protocol for computing $\mathcal{F}_{HSecAgg}$, i.e., there exists a probabilistic polynomial-time (PPT) simulator* SIM *such that*

$$\mathsf{SIM}_{\mathcal{C}}^{\mathcal{U},t,\kappa}(M_C, x_{\mathcal{U}\setminus\mathcal{C}}) \equiv \mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,\kappa}(M_C)$$

where $\equiv$ denotes that the distributions are identical.

Proof. The proof is identical to that for Theorem 4.1. The reason is that even with M_C, which enables participants in $\mathcal{C}$ to choose their inputs at any round of the protocol execution, participants in $\mathcal{C}$ learn nothing about $x_{\mathcal{U}\setminus\mathcal{C}}$ rather than the list of connected participants. Therefore, the simulator SIM can let all active malicious participants having their actual inputs and other participants having dummy inputs to perfectly simulate the views of the participants in $\mathcal{C}$. In this case, the simulated combined views of the participants in $\mathcal{C}$ are identical to that in REAL. □

However, for the threat model including both active malicious clients and server, the proof is different from that for Theorem 4.2 as the sum of $\boldsymbol{x}_i$ is no longer available as the input to the simulator SIM (see H_4 in the proof for Theorem 4.2). Thus, we allow SIM to learn the sum by making a query to an ideal function I in RO for dynamically chosen subset of honest participants denoted as $\mathcal{L}$ at any round of the protocol execution. More precisely, the ideal function I takes $\mathcal{L}, \mathcal{U}, \mathcal{C}$ and a lower bound of the number of honest participants δ as inputs, and outputs $\sum_{u_i \in \mathcal{L}} \boldsymbol{x}_i$ if $\mathcal{L} \subseteq (\mathcal{U} - \mathcal{C})$ and $|\mathcal{L}| \geq \delta$, and aborts otherwise.

Theorem 4.4 (Security against active malicious clients, including the server). *For all $\mathcal{U}, t, \kappa$ with $|\mathcal{C} \setminus \{\mathcal{S}\}| < t$, $x_{\mathcal{U}\setminus\mathcal{C}}, \mathcal{C}$ with the algorithm M_C where $\mathcal{C} \subseteq \mathcal{U} \cup \{\mathcal{S}\}$, and $\delta = t - |\mathcal{C} \cap \mathcal{U}|$, the protocol 1 is a secure protocol for computing $\mathcal{F}_{HSecAgg}$, i.e., there exists a probabilistic polynomial-time (PPT) simulator* SIM *such that*

$$\mathsf{SIM}_{\mathcal{C}}^{\mathcal{U},t,\kappa}(M_C, x_{\mathcal{U}\setminus\mathcal{C}}) \equiv \mathsf{REAL}_{\mathcal{C}}^{\mathcal{U},t,\kappa,I}(M_C)$$

where $\equiv$ denotes that the distributions are identical.

Proof. Similar to the proof for Theorem 4.2, we use a standard hybrid argument to prove the theorem. By defining a sequence of modifications to the random variable REAL, we can construct the simulator SIM with a sequence of hybrid distributions. If any two subsequent hybrids are computationally indistinguishable, the distribution of simulator SIM as a whole is identical to the real execution REAL.

* H_0: In this hybrid, the distribution of the combined views of M_C of SIM is exactly the same as that of REAL.
* H_1: In this hybrid, we substitute all the shares s_i^j for each client $u_i \in \mathcal{U}_1 \setminus \mathcal{C}$, with a randomly selected element in $\mathbb{Z}_P$. The security of Shamir secret sharing scheme guarantees identical distribution from the previous one.
* H_2: In addition to the previous hybrid, the SIM aborts if M_C provides any incorrect $\sigma_{i,j}^1$. Since this is equivalent to breaking the security of the signature scheme, this hybrid is identical to the previous one.
* H_3: In this hybrid, the mask $\boldsymbol{r}_i$ of each simulated client u_i is substituted with a randomly selected number in appropriate length, and the adversaries' masks are set to be 0. The security of HPRG leveraging the Decisional Diffie-Hellman assumption guarantees the identical distribution of this hybrid to the previous one.
* H_4: In addition to the previous hybrid, the SIM aborts if M_C provides any incorrect σ_i^2. Since this is equivalent to breaking the security of the signature scheme, this hybrid is identical to the previous one.
* H_5: In addition to the previous hybrid, the SIM aborts if M_C provides any incorrect σ_s^3. Because of the security of the signature scheme that guarantees that forgeries can happen only with negligible probability, this hybrid is indistinguishable from the previous one.
* H_6: In addition to the previous hybrid, the SIM aborts if M_C provides any incorrect σ_i^4. Because of the security of the signature scheme that guarantees that forgeries can happen only with negligible probability, this hybrid is indistinguishable from the previous one.
* H_7: Denote the list of $\mathcal{U}_2$ fetched from the server as $\mathcal{Q}$. The SIM aborts if two different $\mathcal{Q}$ are signed by the clients. Since this amounts to breaking the security discussed in Section 4.2.3, and the server cannot forge signatures on behalf of the honest clients, this hybrid is indistinguishable from the previous one.
* H_8: In addition to the previous hybrid, the SIM aborts if M_C provides any incorrect σ_i^4. Because of the security of the signature scheme that guarantees that forgeries can happen only with negligible probability, this hybrid is indistinguishable from the previous one.
* H_9: In this hybrid, the simulator SIM does not receive the inputs of the honest participants. Instead, it learns the required value $\boldsymbol{w}_i$ for the set $(\mathcal{Q} \setminus \mathcal{C})$ with respect to $\sum_{u_i \in \mathcal{Q}\setminus\mathcal{C}} \boldsymbol{w}_i = \sum_{u_i \in \mathcal{Q}\setminus\mathcal{C}} \boldsymbol{x}_i$ by making a query to the ideal function I. Note that according to the discussion in H_7 and Section 4.2.3, the ideal function I will not abort and does not modify the joint view of $\mathcal{C}$. Thus, this hybrid is indistinguishable from the previous one.

By defining such PPT simulator SIM as described in the last hybrid, the active malicious adversaries' combined views of SIM is computationally indistinguishable from that of the real execution REAL, and thus the proof is completed. This means the active malicious participants learn nothing except for the sum of $\boldsymbol{x}_i$ where $u_i \in \mathcal{L}$ and $|\mathcal{L}| \geq \delta$. □

Note that for active adversaries, the security of the consistency check can be guaranteed only when the threshold t with respect to δ is set to be a proper value. Next, we discuss the minimum value of t required for security in three threat models as follows.

4.2.1 Malicious clients with honest server

It is evident that with an honest server, each honest client views the correct list of connected clients, which is guaranteed by the protocol of consistency check. This means that for any $t > 1$, the client learns nothing about the information of other's inputs.

4.2.2 Honest clients with malicious server

Compared with the previous model, malicious server makes the situation a bit more complicated. The reason is that for some specific threshold t, the server is able to learn the information of the inputs, while passing the consistency check. For example, there are four clients u_1, u_2, u_3, u_4 with threshold $t = 2$, the malicious server actively sends the connected client list $l_1 = \{u_1, u_2, u_3\}$ to u_1, u_2, and $l_2 = \{u_1, u_2, u_3, u_4\}$ to u_3, u_4, then following the consistency check protocol, u_1 and u_2 receive l_1 with their signatures on l_1, say $\sigma_1\{l_1\}$ and $\sigma_2\{l_1\}$. Since they can verify the origin of the messages and $|\{\sigma_1\{l_1\}, \sigma_2\{l_1\}\}| = 2 \geq t$, u_1 and u_2 do not abort during the consistency check, and thus u_3 and u_4 are viewed as dropped clients from the angle of u_1 and u_2. Consequently, in our protocol, the server will receive the product of masks from connected clients $R_1 = r_1 r_2 r_3$ where r_i is the mask generated by u_i. Similarly, the server will also receive $R_2 = r_1 r_2 r_3 r_4$. As a result, r_4 can be easily obtained by computing $\frac{R_1}{R_2}$ which causes leakage of u_4's input. To avoid such different views for n clients, the threshold t has to be set to be greater than $\frac{n}{2}$. In this case, since all the clients are honest, no one will make the signature twice, the number of signatures equals to n. This means that for two fraudulent list l_1 and l_2, if l_1 passes the consistency check with $|\{\sigma\{l_1\}\}| > t$, the list l_2 cannot pass the consistency check as $|\{\sigma\{l_2\}\}| = n - |\{\sigma\{l_1\}\}| < t$. Therefore, the security is guaranteed for $t \geq \lfloor \frac{n}{2} \rfloor + 1$.

4.2.3 Malicious clients and server

Different from the assumption in the previous model, for malicious clients, they are able to make the signature on the list l any number of time. For example, there are six clients $u_1, u_2, u_3, u_4, u_5, u_6$ with the threshold $t = 4$ where

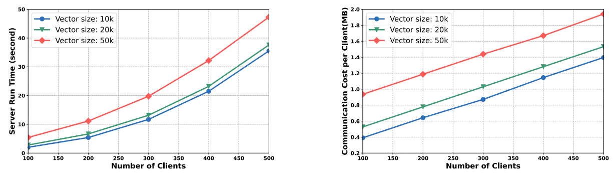


Fig. 1. Server's (total) runtime and client's communication cost as the number of clients increases. No dropout client.

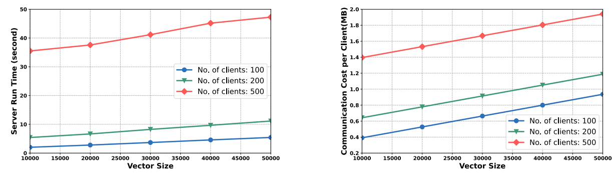
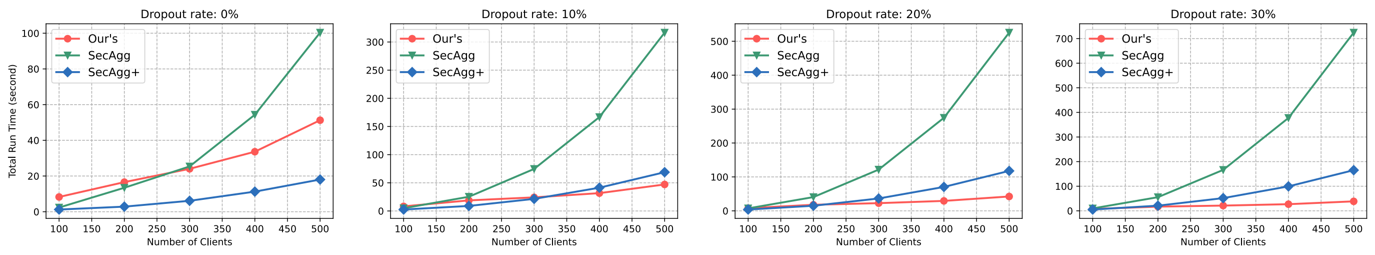
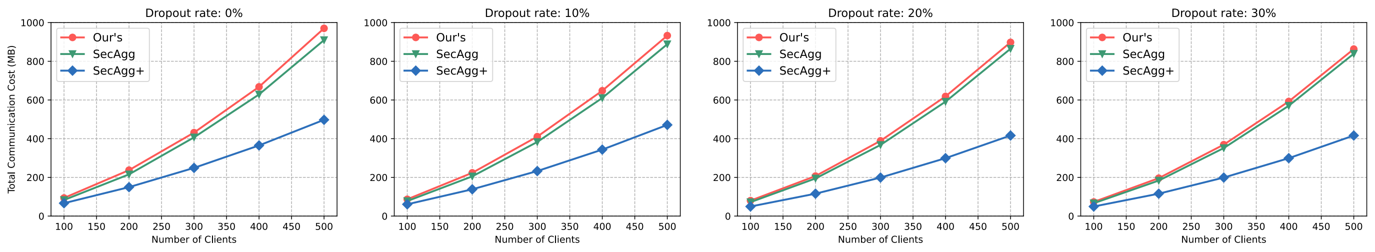


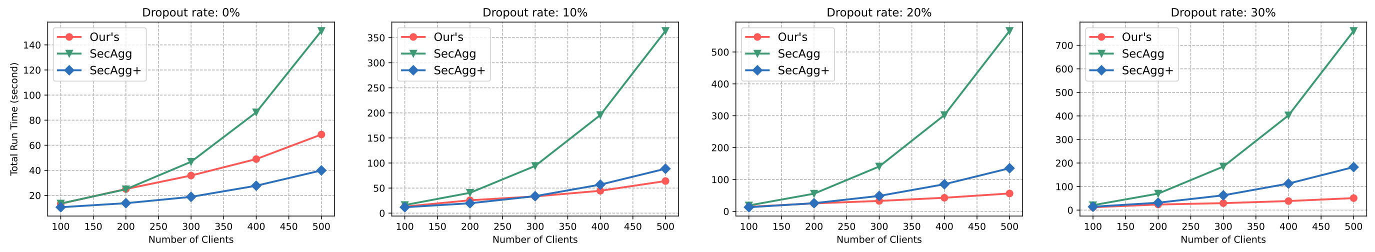
Fig. 2. Server's (total) runtime and client's communication cost as the vector size increases. No dropout client.



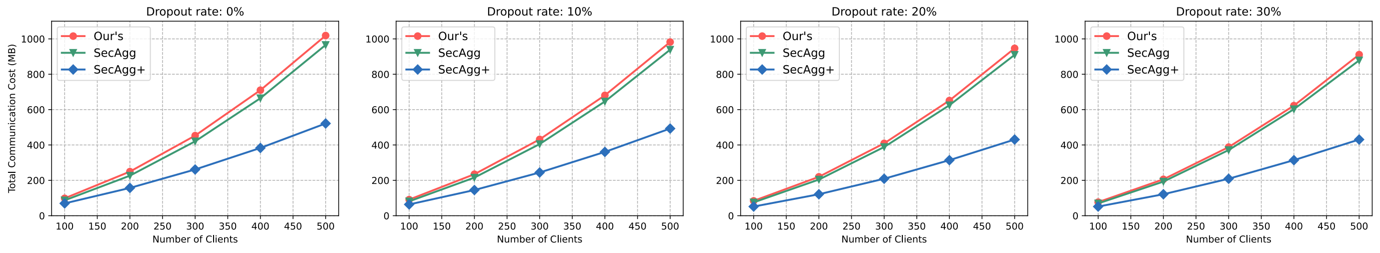
(a) Runtime in LAN setting.



(b) Communication cost in LAN setting.

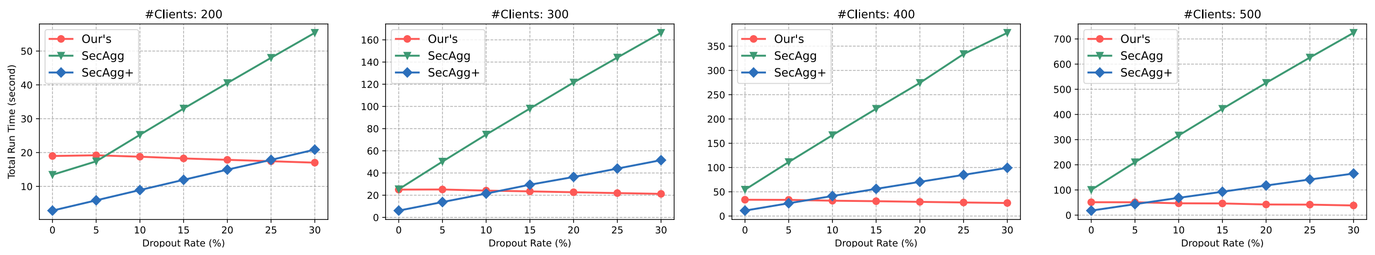


(c) Runtime in WAN setting.

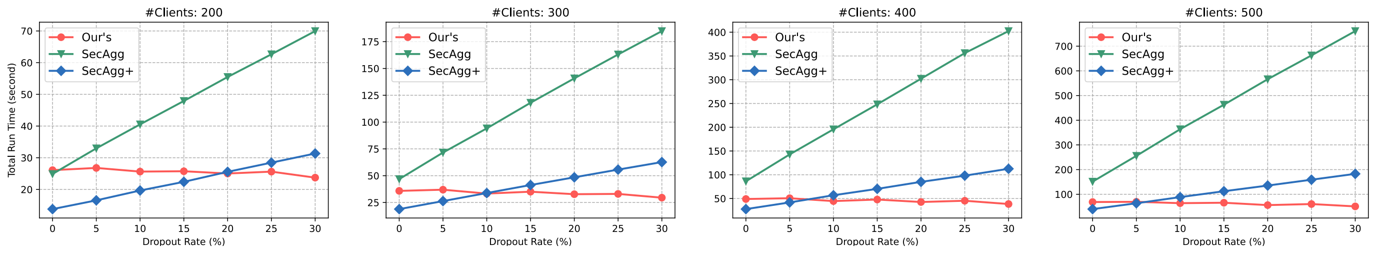


(d) Communication cost in WAN setting.

Fig. 3. Total runtime and communication cost with different dropout rates. The vector size is fixed to 50K.



(a) LAN setting.



(b) WAN setting.

Fig. 4. Total runtime as the dropout rate increases in different network settings. The vector size is fixed to 50K.

u_1 and u_2 are active malicious clients. The client u_3 and u_4 receive the list $l_1 = \{u_1, u_2, u_3, u_4, u_5\}$ with the signatures $\sigma_1 = \{\sigma_1\{l_1\}, \sigma_2\{l_1\}, \sigma_3\{l_1\}, \sigma_4\{l_1\}\}$ while the client u_5 and u_6 receive the list $l_2 = \{u_1, u_2, u_3, u_4, u_5, u_6\}$ with the signatures $\sigma_2 = \{\sigma_1\{l_2\}, \sigma_2\{l_2\}, \sigma_5\{l_2\}, \sigma_6\{l_2\}\}$. Since the signatures are correctly made and $|\sigma_1| \geq t$, $|\sigma_2| \geq t$, no one will abort during the consistency check, and thus the input of client u_6 will be leaked. We can observe that client u_1 and u_2 have made their signatures on both l_1 and l_2. Therefore, even with $t \geq \lfloor \frac{n}{2} \rfloor + 1$, the privacy of clients' inputs can be compromised in the setting where both the server and clients are active malicious.

Now we proceed to discuss the lower bound of threshold t in this model. Recall that we are considering the setting where they are n clients in total including n_c malicious clients with the threshold t. For two fraudulent lists l_1 and l_2, let the number of signatures on l_1 and l_2 from honest clients be n_1 and n_2 respectively, and the number of signatures from malicious clients be n_c. In order to have two fraudulent lists l_1 and l_2 passing the consistency check, the number of effective signatures on l_1 and l_2 should be not less than t such that

$$t \leq n_1 + n_c, t \leq n_2 + n_c$$

while $n_1 + n_2 + n_c \leq n$, we can observe that for $t \leq \frac{n+n_c}{2}$, it is possible to construct two different views to clients. Therefore, to guarantee the security, the threshold t must be greater than $\frac{n+n_c}{2}$. Furthermore, as the number of adversaries n_c cannot be greater than $\frac{n}{3}$ (refer to [9] for the details), we can have the minimum threshold t for the required security is $\lfloor \frac{2n}{3} \rfloor + 1$.

5 Performance Analysis

In this section, we first analyze the computation and communication cost of the client and server, respectively. Then, we implement a prototype to show the performance of our proposed protocol.

5.1 Complexity analysis

We summarize the analysis of complexity and dropout-resilience compared to other existing protocols in Table 1.

Computation overheads: Each client's computation cost mainly depends on (i) computing n (t, n) Shamir secret sharing with $\mathcal{O}(n^2)$ complexity and (ii) generating m mask for the input vector using HPRG with $\mathcal{O}(m)$ complexity. Thus the total computation complexity per client is $\mathcal{O}(n^2+m)$. Since the server involves only one reconstruction from n (t, n) Shamir secret shares, the overhead is $\mathcal{O}(n)$. Note that we adopt similar method in SecAgg to precompute Lagrange basis polynomials (see Section 2.1 and [9] for the details) for Shamir secret sharing, computational cost for one reconstruction results in $\mathcal{O}(n)$ complexity rather than $\mathcal{O}(n^2)$ in standard Shamir scheme.

Communication overheads: Communication cost of each client consists of sending n encrypted shares and one masked model with m elements, which causes $\mathcal{O}(m + n)$ complexity. The communication cost of the server is dominated by forwarding messages between every pair of clients with $\mathcal{O}(n^2)$ complexity and receiving masked models from the clients with $\mathcal{O}(mn)$ complexity, which is $\mathcal{O}(n^2 + mn)$ complexity in total. Note that compared with SecAgg, our scheme does not involve pair-wise DH protocol, thus reducing $2n$ key exchange overhead for each client and $2n^2$ message forwarding for the server.

Dropout resilience: First, we note that the maximum number of dropout clients that our proposed scheme can tolerate is the same as that of the SecAgg scheme since both of them use a (t, n) Shamir secret sharing scheme, which does not sacrifice any dropout-resilience compared to SecAgg+ scheme. Furthermore, by replacing communication-intensive Diffie-Hellman protocol adopted in SecAgg and SecAgg+ with HPRG and additive operation based on Shamir secret sharing scheme, the runtime of our proposed protocol decreases with the increase of the dropout rate, rather than the increasing runtime of previous

TABLE 2
The client's and server's (total) runtime for different steps of the proposed protocol in LAN/WAN settings under semi-honest model. The vector size is fixed to 50K with 64 bits length.

	Num. clients	Dropout rate	Sharing seeds	Collecting Masked models	Unmasking	total runtime
Client	500	0%	2.28s/7.76s	0.78s/3.24s	0.03s/0.84s	3.09.s/11.84s
Server	500	0%	2.61s/7.76s	0.83s/3.24s	44.45s/57.56s	47.88s/68.56s
Server	500	10%	2.47s/8.26s	0.78s/3.25s	38.96s/52.48s	42.21s/63.99s
Server	500	20%	2.47s/6.63s	0.78s/2.62s	35.33s/46.88s	38.59s/56.13s
Server	500	30%	2.47s/6.16s	0.79s/2.63s	31.08s/42.11s	34.34s/50.90s
Client	1000	0%	8.36s/16.77s	0.78s/3.91s	0.23s/1.85s	9.37s/22.53s
Server	1000	0%	8.81s/16.77s	0.79s/3.91s	145.34s/242.37s	154.94s/263.05s
Server	1000	10%	8.83s/14.15s	0.78s/3.69s	129.10s/224.01s	138.71s/241.85s
Server	1000	20%	8.87s/14.23s	0.78s/3.57s	108.17s/177.22s	117.82s/195.02s
Server	1000	30%	8.85s/17.31s	0.83s/3.68s	91.95s/147.71s	101.63s/168.70s

TABLE 3
The client's and server's (total) runtime for different steps of the proposed protocol in LAN/WAN settings under active malicious model. The vector size is fixed to 50K with 64 bits length.

	Num. clients	Dropout rate	Sharing seeds	Collecting Masked models	Checking consistency	Unmasking	Total runtime
Client	500	0%	5.08s/9.54s	0.84s/2.65s	0.03s/0.87s	0.23s/0.12s	6.18s/13.18s
Server	500	0%	5.26s/9.54s	0.83s/2.65s	0.03s/0.87s	88.58s/131.14s	94.70s/144.20s
Server	500	10%	5.26s/11.55s	0.85s/3.69s	0.03s/0.87s	73.23s/106.59s	79.36s/122.70s
Server	500	20%	5.26s/11.46s	0.84s/3.72s	0.03s/0.87s	60.34s/90.28s	66.48s/106.33s
Server	500	30%	5.26s/12.75s	0.83s/4.34s	0.02s/0.86s	48.90s/75.07s	55.03s/93.03s
Client	1000	0%	14.79s/20.19s	0.79s/3.33s	0.10s/0.95s	5.75s/14.87s	21.43s/39.34s
Server	1000	0%	14.81s/20.19s	0.79s/3.33s	0.10s/0.95s	406.40s/594.11s	422.10s/619.30s
Server	1000	10%	14.85s/22.53s	0.79s/3.04s	0.09s/0.94s	311.71s/482.24s	327.44s/508.75s
Server	1000	20%	14.88s/23.79s	0.79s/3.61s	0.09s/0.93s	238.46s/391.50s	254.21s/419.39s
Server	1000	30%	14.83s/23.65s	0.79s/3.44s	0.08s/0.92s	174.12s/307.38s	189.81s/335.39s

works, which implies the stronger dropout-resilience of our scheme.

5.2 Experiments

Our prototype is tested on two c5.4xlarge AWS EC2 instances running Ubuntu 18.04 with 16 vCPUs and 32GB memory. The central server executes on one instance while the clients run parallel in the other instance. In the LAN setting, the instances are both hosted within the same region, i.e., Singapore (ap-southeast-1). In the WAN setting, the central server executes on an instance located in Singapore, and the clients execute on another instance located in northern Virginia, US (us-east-1). Our main scheme is implemented in Python using Gmpy2, Cryptography library, and several other standard libraries. The Pollard's lambda method for computing discrete logarithms is implemented in C++ using NTL library. For cryptographic primitives, we adopt AES-GCM with 128-bit keys for authenticated encryption, standard (t, n) Shamir secret sharing scheme to deal with dropped clients, and DDH based HPRG constructed with an SHA-256 hash to generate masks. We note that our proposed scheme is usable for all kinds of machine learning models, as we only focus on the part for secure aggregation. Thus, our scheme does not affect any performance of the ML model, and there is no deviation due to the usage of cryptographic techniques.

As noted earlier, our focus is on PPML systems that clients are resource-constrained mobile devices, hence it is necessary to investigate the communication cost from the client perspective. In addition, since the new FL round begins only after the server obtains the aggregation result in the last FL round, the total runtime from the server perspective may also affect user experience. Thus, we first evaluate the server's runtime and client's communication cost with different numbers of clients and vector sizes in a no-dropout setting, of which the experimental results are given in Fig. 1 and Fig. 2. We can observe that with 500 clients and 50K vector size comparable to LeNet [28], only about 2MB communication cost is required for each client, and the whole aggregation can be done in about 1 minute, which implies the efficiency of our proposed scheme.

Furthermore, taking dropout clients into account, we investigate the total runtime and client's communication cost with different dropout rates in both LAN and WAN settings, of which the experimental results are given in Fig. 3. We can observe that our scheme involves a comparable total communication cost of the whole aggregation to that of SecAgg, as the same (t, n) Shamir scheme is kept, but with less total runtime. Such observation keeps consistency in both LAN and WAN settings. Specifically, for large-scale systems, say with 500 clients, when the dropout rate is large, say greater than 10%, our scheme's efficiency performance outperforms SecAgg and SecAgg+. Fig. 4 also supports this point. The total runtime of our proposed scheme decreases with the increase of the dropout rate, rather than the increasing runtime of SecAgg and SecAgg+, which implies the stronger dropout-resilience of our scheme compared to previous works.

Besides, the client's and server's runtime for different steps of our proposed protocol under semi-honest setting and active malicious setting are given in Table. 2 and Table. 3

TABLE 4
Runtime of computing discrete logarithms using Pollard's lambda method with different vector size. The order of involved finite cyclic group is set to be a 1024-bit prime.

Vector Size	Model Type	Runtime
72	Linear Regression[10]	30ms
7850	Logistic Regression [28]	1.9s
35K	SVM [1]	8.6s
50K	LeNet [28]	12.5s
2.5M	MobileNet V3 small [25]	601.5s
5.5M	MobileNet V3 large [25]	1338.7s
11.7M	ResNet18 [24]	2039.3s

respectively, from which we can observe that the additional cost to guarantee the security under the active malicious setting due to the use of a large number of signatures and verification techniques. Moreover, as mentioned in Section 3.3, the server can accelerate the computation of discrete logarithms in the unmasking step of our protocol by using Pollard's lambda method [33]. In Table. 4, we show the runtime of such computation for different sizes of the clients' gradients comparable to several ML models. We can observe from Table. 4 that the time required is competitive for small traditional ML models and simple neural networks such as LeNet [28], but becomes impractical for large-scale neural networks such as ResNet18 [24]. However, we note that for lightweight neural networks such as MobileNet V3 [25] that can be deployed on resource-constrained mobile devices, the overheads of computing discrete logarithms can be still affordable if more powerful servers and multi-threading implementations are adopted.

5.3 Further discussions

As noted in Section 5.2, we emphasize that our focus is on designing a secure aggregation protocol to protect the privacy of clients' gradient vectors, hence the intermediate and final global models are revealed to all participants. Thus, our proposed scheme is still vulnerable to the membership inference attack [42]. In this case, attackers can determine if a record is in clients' training datasets, given only some global models. Protecting the privacy of global models requires clients to train their ML models over encrypted global models. Existing solutions include HE-based schemes [17, 38] and MPC-based schemes[13, 30]. However, those solutions may involve large overheads for large-scale ML models such as deep neural networks. To improve the efficiency while keeping the privacy of global models, sophisticated integrations of our proposed scheme with existing solutions are required.

Furthermore, we should note that secure aggregation schemes such as ours, SecAgg and SecAgg+ hinder the deployment of the defense methods against so-called poisoning attack or backdoor attack [4, 36]. Existing solutions [2, 8, 37, 40] rely on different metrics to evaluate the quality of clients' locally trained models, hence detect the poisoned models, which means that the server needs to know each client's model. However, since secure aggregation schemes protect the privacy of clients' locally trained models, it is not straightforward to combine them with defense methods against poisoning attacks. Thus, designing secure aggregation protocols compatible with existing defenses against poisoning attacks still remains a topic for further research.

6 Conclusions

We proposed an efficient aggregation protocol to compute the sum of inputs from a set of participants while preserving their input privacy. Our protocol allows participants to drop out from the protocol during the execution and provides stronger dropout-resilience compared to previous works. Thus it is suitable to be applied to large-scale PPML scenarios. Additionally, the security of our protocol is guaranteed against both semi-honest and malicious adversaries by setting proper system parameters. Besides, the simplicity of the proposed scheme makes it attractive both for implementation and for further improvements.

Acknowledgments

We would like to thank three anonymous reviewers for their comments on the earlier versions of this paper. In addition, we thank the authors of TurboAgg and FastSecAgg for their shared materials and helpful discussions. This research is supported by the National Research Foundation, Singapore under its Strategic Capability Research Centres Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

[1] SVM MNIST digit classification in python using scikit-learn. https://github.com/ksopyla/svm_mnist_digit_classification.

[2] Sebastien Andreina, Giorgia Azzurra Marson, Helen Möllering, and Ghassan Karame. Baffle: Backdoor detection via feedback-based federated learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 852–863. IEEE, 2021.

[3] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2017.

[4] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

[5] Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens. Key-homomorphic constrained pseudorandom functions. In *Theory of Cryptography Conference*, pages 31–60. Springer, 2015.

[6] Constance Beguier, Mathieu Andreux, and Eric W Tramel. Efficient sparse secure aggregation for federated learning. *arXiv preprint arXiv:2007.14861*, 2020.

[7] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference*

on Computer and Communications Security, pages 1253–1269, 2020.

[8] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 118–128, 2017.

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

[10] Luis M Candanedo, Véronique Feldheim, and Dominique Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and buildings*, 140:81–97, 2017.

[11] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[12] T-H Hubert Chan, Elaine Shi, and Dawn Song. Privacy-preserving stream aggregation with fault tolerance. In *International Conference on Financial Cryptography and Data Security*, pages 200–214. Springer, 2012.

[13] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*, 2020.

[14] Beongjun Choi, Jy-yong Sohn, Dong-Jun Han, and Jaekyun Moon. Communication-computation efficient secure aggregation for federated learning. *arXiv preprint arXiv:2012.05433*, 2020.

[15] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[16] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, Thomas Schneider, Hossein Yalame, et al. Safelearn: secure aggregation for private federated learning. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 56–62. IEEE, 2021.

[17] David Froelicher, Juan Ramón Troncoso-Pastoriza, Apostolos Pyrgelis, Sinem Sav, Joao Sa Sousa, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Scalable privacy-preserving distributed learning. *Proc. Priv. Enhancing Technol.*, 2021.

[18] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[19] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78, 1998.

[20] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 241–264. 2019.

[21] Filip Granqvist, Matt Seigel, Rogier van Dalen, Áine Cahill, Stephen Shum, and Matthias Paulik. Improving on-device speaker verification using federated learning with privacy. *arXiv preprint arXiv:2008.02651*, 2020.

[22] Jiale Guo, Ziyao Liu, Kwok-Yan Lam, Jun Zhao, and Yiqiang Chen. Privacy-enhanced federated learning with weighted aggregation. In *International Symposium on Security and Privacy in Social Networks and Big Data*, pages 93–109. Springer, 2021.

[23] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient rsa key generation and threshold paillier in the two-party setting. *Journal of Cryptology*, 32(2):265–323, 2019.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[25] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[26] Swanand Kadhe, Nived Rajaraman, O Ozan Koyluoglu, and Kannan Ramchandran. Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning. *arXiv preprint arXiv:2009.11248*, 2020.

[27] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] Xiuhua Li, Luxi Cheng, Chuan Sun, Kwok-Yan Lam, Xiaofei Wang, and Feng Li. Federated-learning-empowered collaborative data sharing for vehicular edge networks. *IEEE Network*, 35(3):116–124, 2021.

[30] Ziyao Liu, Ivan Tjuawinata, Chaoping Xing, and Kwok-Yan Lam. Mpc-enabled privacy-preserving neural network training against malicious attack. *arXiv preprint arXiv:2007.12557*, 2020.

[31] Kalikinkar Mandal and Guang 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*, pages 57–68, 2019.

[32] Kalikinkar Mandal, Guang Gong, and Chuyi Liu. Nike-based fast privacy-preserving highdimensional data aggregation for mobile devices. Technical report, CACR Technical Report, CACR 2018-10, University of Waterloo, Canada, 2018.

[33] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018.

[34] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 35–52, 2018.

[35] Moni Naor, Benny Pinkas, and Omer Reingold. Dis-

tributed pseudo-random functions and kdcs. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 327–346. Springer, 1999.
[36] Thien Duc Nguyen, Phillip Rieger, Markus Miettinen, and Ahmad-Reza Sadeghi. Poisoning attacks on federated learning-based iot intrusion detection system. In *Proc. Workshop Decentralized IoT Syst. Secur.(DISS)*, pages 1–7, 2020.
[37] Thien Duc Nguyen, Phillip Rieger, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Ahmad-Reza Sadeghi, Thomas Schneider, et al. Flguard: Secure and private federated learning. *arXiv preprint arXiv:2101.02281*, 2021.
[38] Sinem Sav, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. POSEIDON: privacy-preserving federated neural network learning. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*, 2021.
[39] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
[40] Shiqi Shen, Shruti Tople, and Prateek Saxena. Auror: Defending against poisoning attacks in collaborative deep learning systems. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 508–519, 2016.
[41] Elaine Shi, TH Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Proc. NDSS*, volume 2, pages 1–17. Citeseer, 2011.
[42] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.
[43] Jinhyun So, Başak Güler, and A Salman Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, 2021.
[44] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *arXiv preprint arXiv:2004.02229*, 2020.
[45] Helin Yang, Jun Zhao, Zehui Xiong, Kwok-Yan Lam, Sumei Sun, and Liang Xiao. Privacy-preserving federated learning for uav-enabled networks: Learning-based joint scheduling and resource management. *IEEE Journal on Selected Areas in Communications*, 2021.
[46] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.
[47] Yang Zhao, Jun Zhao, Mengmeng Yang, Teng Wang, Ning Wang, Lingjuan Lyu, Dusit Niyato, and Kwok-Yan Lam. Local differential privacy-based federated learning for internet of things. *IEEE Internet of Things Journal*, 8(11):8836–8853, 2020.
[48] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pages 14774–14784, 2019.

Ziyao Liu received his B.E. degree from the school of Electronics Information Engineering, Zhengzhou University, Zhengzhou, China, in 2015, and the M.S. degree from Beijing Institute of Technology, Beijing, China, in 2018. He is currently working towards a Ph.D. degree in the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests include privacy-preserving machine learning, multi-party computation, and applied cryptography.

Jiale Guo received her B.S. from the School of Mathematics, Shandon University, Jinan, China, in 2017. She is currently pursuing a Ph.D. degree in the School of Computer Science and Engineering, Nanyang Technological University, Singapore. Her research interests include privacy-preserving machine learning and Cybersecurity.

Kwok-Yan Lam (Senior Member, IEEE) received his B.Sc. degree (1st Class Hons.) from University of London, in 1987, and Ph.D. degree from University of Cambridge, in 1990. He was a Visiting Scientist at the Isaac Newton Institute, Cambridge University, and a Visiting Professor at the European Institute for Systems Security. He has collaborated extensively with law-enforcement agencies, government regulators, telecommunication operators, and financial institutions in various aspects of Infocomm and Cyber Security in the region. From 2002 to 2010, he was a Professor with Tsinghua University, China. Since 1990, he has been a Faculty Member with the National University of Singapore and the University of London. He is currently a Full Professor with Nanyang Technological University, Singapore and the Director of the Strategic Centre for Research in Privacy-Preserving Technologies and Systems (SCRiPTS). From August 2020, Professor Lam is also on part-time secondment to the INTERPOL as a Consultant at Cyber and New Technology Innovation. In 1998, he received the Singapore Foundation Award from the Japanese Chamber of Commerce and Industry in recognition of his research and development achievement in information security in Singapore.

Jun Zhao (S'10-M'15) is currently an Assistant Professor in the School of Computer Science and Engineering (SCSE) at Nanyang Technological University (NTU), Singapore. He received a Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University (CMU), Pittsburgh, PA, USA, in May 2015, and a bachelor's degree in Information Engineering from Shanghai Jiao Tong University, China, in June 2010. One of his papers was a finalist for the best student paper award in IEEE International Symposium on Information Theory (ISIT) 2014. His research interests include A.I. and data science, security and privacy, control and learning in communications and networks.