Electronic Preprint for Journal of Information Processing Vol.30

Regular Paper

SPGC: Integration of Secure Multiparty Computation and Differential Privacy for Gradient Computation on Collaborative Learning

Kazuki Iwahana[1,a)] Naoto Yanai[1,b)] Jason Paul Cruz[1,c)] Toru Fujiwara[1,d)]

Received: May 31, 2021, Accepted: December 3, 2021

Abstract: Achieving *differential privacy* and utilizing *secure multiparty computation* are the two primary approaches used for ensuring privacy in privacy-preserving machine learning. However, the privacy guarantee by existing integration protocols of both approaches for collaborative learning weakens when more participants join the protocols. In this work, we present *Secure and Private Gradient Computation* (*SPGC*), a novel collaborative learning framework with a strong privacy guarantee independent of the number of participants while still providing high accuracy. The main idea of SPGC is to *create noise for the differential privacy within secure multiparty computation*. We also created an implementation of SPGC and used it in experiments to measure its accuracy and training time. The results show that SPGC is more accurate than a naive protocol based on local differential privacy by up to 5.6%. We experimentally show that the training time increases in proportion to the noise generation and then demonstrate that the privacy guarantee is independent of the number of participants as well as the accuracy evaluation.

Keywords: collaborative learning, privacy-preserving machine learning, secure multiparty computation, differential privacy

1. Introduction

1.1 Backgrounds

Collaborative learning also known as federated learning is a kind of machine learning where multiple participants pool all training data, and a model is trained on this communal pool. In general, each participant trains a local model with its data and then periodically exchanges and updates model parameters with other participants. Therefore, privacy is an essential issue in collaborative learning because training data are collected and pooled from many sources. When sensitive data, such as genome data or private images are utilized as training data, their privacy should be protected. Inference attacks on training data privacy that reveal information about the training data [39], [46] have been found in the past years.

A typical approach to guarantee privacy in collaborative learning is to achieve *differential privacy* [19], that is, informally, each participant gives data noise in local [17], [61] and then updates a model with the data from all participants. However, such an approach often generates significant amounts of noise that degrade the accuracy of its resultant model. In contrast, the privacy guarantee weakens when the noise generated in local is slight because the training data can be inferred from model parameters [39], [46].

[1] Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[a)] k-iwahana@ist.osaka-u.ac.jp
[b)] yanai@ist.osaka-u.ac.jp
[c)] cruz@ist.osaka-u.ac.jp
[d)] fujiwara@ist.osaka-u.ac.jp

A potential solution to the privacy issues mentioned above is the integration [51] of differential privacy and *secure multiparty computation*, which computes a function on inputs given by participants without revealing the inputs themselves. In the integration approach, model parameters are protected by the use of secure multiparty computation before they are exchanged among participants, and small amounts of noise for differential privacy are generated and added when the model is updated [65], [67]. Intuitively, the integration approach enables participants to adjust the total amount of noise they generate and thus supports both the privacy guarantee and the inference accuracy.

However, the privacy guarantee in existing integration protocols [65], [67] on collaborative learning weakens when the number of participants increases. In collaborative learning, many participants are expected to join the network, given that the primary motivation for collaborative learning is to collect data from many sources. Therefore, a protocol that offers a privacy guarantee independent of the number of participants is desirable. This paper aims to answer the following fundamental question: *Can a new integrated protocol of differential privacy and secure multiparty computation for collaborative learning which maintains a strong privacy guarantee and accuracy without depending on the number of participants be constructed?*

This paper is a full version of our previous work published at DPM 2021 [28]. We proposed an integrated protocol of differential privacy and secure multiparty computation for collaborative learning, SPGC, and analyzed the privacy guarantee of SPGC formally. We also evaluated the accuracy and training time via experiments in our previous work. In the current version, the lack of

the correctness of the protocol by Chase et al. [9], which is essential existing work, is presented formally. We have also presented training loss values to analyze the training time of SPGC in detail. Furthermore, related works are discussed in terms of developments on privacy-preserving machine learning, secure multiparty computation, and differential privacy, which were not presented in the previous work.

1.2 Contribution

In this paper, we present *SPGC*, pronounced as "speak" and which is an abbreviation for Secure and Private Gradient Computation. SPGC is an integrated protocol of differential privacy and secure multiparty computation for collaborative learning that provides a privacy guarantee independent of the number of participants. As another contribution, we conducted extensive experiments on SPGC with academic and medical diagnosis datasets to evaluate its performance.

The construction of SPGC is significantly different from existing works [65], [67]. Existing works adopt local differential privacy [17] (LDP) in which each participant generates noise in local and then *removes* the noise via secure multiparty computation. However, an LDP-based construction will weaken the privacy guarantee even by the use of *any* secure multiparty computation. In contrast, the main idea behind SPGC is the *creation of noise within secure multiparty computation*.

The idea mentioned above was inspired by the collaborative gradient computation (CGC) protocol of Chase et al. [9], in which the noise is generated by using secure multiparty computation. However, we note that CGC is incomplete, and that SPGC is quite an improvement over CGC. In particular, we found a negative case in the training method of CGC where the positive and negative gradients become indistinguishable from each other. SPGC can avoid such an issue as well. Moreover, Chase et al. did not discuss experimental evaluation in detail, more specifically they only provided the inference accuracy on the MNIST dataset and they did not provide training time. Detailed experimental evaluation is crucial for an integration protocol for differential privacy and secure multiparty computation machine learning. How noise on the differential privacy affects evaluation on each dataset is essential for accuracy, and communication overhead on the secure multiparty computation is significant for training time. We formally show that SPGC overcomes the negative case described, and then present the experimental evaluation in detail including evaluations on medical diagnosis datasets in addition to the MNIST dataset and those training times. (See Section 4 and Section 5 for more detail). We then show that the training time increases with respect to the noise generation through the experimental results. We also demonstrate that the privacy guarantee is independent of the number of participants compared to an LDP-based construction as a naive approach. (See Section 6 for more details).

1.3 Paper Organization

The remaining parts of this paper are organized as follows. Background knowledge to help understand this work is presented in Section 2, and the main problem statement is presented in Section 3. The design of SPGC is presented in Section 4, and then the details of our experiments are presented in Section 5. A discussion on the results is presented in Section 6, and related works of SPGC are presented in Section 7. Finally, a conclusion is presented in Section 8.

2. Preliminaries

In this section, we provide backgrounds on deep learning and collaborative learning. We then describe differential privacy and secure multiparty computation as building blocks of SPGC.

2.1 Deep Learning and Collaborative Learning

Deep learning is composed of a *training* phase to find optimal weight parameters on a model to be trained and an *inference* phase to solve a task via inference on the trained model. The goal of the training phase is to find weight parameters that yield an acceptably small loss on a loss function between an output of the model and its training data. Let the current weight parameters be w_t. A loss function L is defined as an average of outputs of the function with data samples $\{x_1, x_2, \cdots, x_u\}$, and thus the function is defined as $L(w_t) = \frac{1}{u}\sum_{i=1}^{u} L(w_t, x_i)$. The stochastic gradient descent (SGD) algorithm is often utilized to minimize L. The SGD algorithm computes a gradient $g = \frac{1}{u}\sum_{x_i \in X} \nabla_{w_t} L(w_t, x_i)$ and updates the current weight parameter w_t to $w_{t+1} = w_t - \eta g$, where η is a learning rate. In general, the update process of weight parameters is done for each data group called a *batch*. One of the well-known implementations to instantiate the SGD algorithm is `AdamOptimizer()`, which is also utilized in this work.

A typical deep learning algorithm considers a centralized setting whereby a central server gathers data for training. In contrast, a deep learning algorithm to train a model distributively by multiple participants is called *collaborative learning*.

The motivation for collaborative learning is to train a model so that even if training data is distributed to multiple participants, each participant trains a local model on its data and then exchanges model parameters with other participants. The main advantage of collaborative learning is to collect training data from multiple participants. In other words, in a situation where training data is distributed to each participant, a participant can share the training with the other participants. To do this, for each step in training a model, each participant sends model parameters, such as gradients or weights, to n central servers. Hereafter, we focus on gradients as model parameters.

For a set $P = \{p_1, p_2, \cdots, p_k\}$ of participants and a set $X = \{X_{p_1}, X_{p_2}, \cdots, X_{p_k}\}$ of data, a participant p_j for any $j \in [1, k]$ computes a gradient $g(X_{p_j})$ for data X_{p_j} in local and then sends it to the central servers.

Roughly speaking, collaborative learning updates a model $Y = f(g(X_{p_1}), g(X_{p_2}), \cdots, g(X_{p_k}))$ of gradients sent from all the participants on n central servers $H = \{H_1, H_2, \cdots, H_n\}$. Afterwards, the servers return Y to all the participant. After receiving the model Y from the central servers, the participants utilize Y in the next training. At the end of the training, each participant owns the same model as the other participants and the central servers.

After the training phase of the collaborative learning is finished, the final model Y often becomes publicly available for in-

ference by a client. In doing so, a client can give the model Y a query on inference with any data z and then obtain the inference from Y. Hereafter, we denote by $Y(z)$ inference on Y with z for a client.

2.2 Differential Privacy

Differential privacy [19] is a mathematical notion that guarantees privacy theoretically. Recall the definition below.

Definition 1. *A randomized mechanism $M : D \to R$ with domain D and range R satisfies (ϵ, δ)-differential privacy if, for any two adjacent inputs $d, d' \in D$ and for any subset of outputs $S \subseteq R$, the following equation holds:*

$$\Pr(M(d) \in S) \leq \exp(\epsilon) \Pr(M(d') \in S) + \delta.$$

Note that ϵ-differential privacy is identical to a special case for $\delta = 0$ in the definition above. When a deterministic real-valued function is defined as $f : D \to R$, a typical way to satisfy the differential privacy for f is to perturb the output of a function f by adding noise. More precisely, a mechanism M is instantiated via additive noise calibrated to f's sensitivity S_f, which is defined as the maximum of the absolute distance $|f(d) - f(d')|$ where d and d' are adjacent inputs. The Gaussian mechanism or the Laplace mechanism is often utilized for generating noise. Hereafter, we focus on the Gaussian mechanism which is defined by $M(d) = f(d) + \mathcal{N}(0, (S_f\sigma)^2)$, where $\mathcal{N}(0, (S_f\sigma)^2)$ is the Gaussian distribution with mean 0 and standard deviation $S_f\sigma$. According to Theorem 3.22 in [20], the Gaussian mechanism for function f of the sensitivity S_f satisfies (ϵ, δ)-differential privacy for $\delta \leq \frac{4}{5} \exp\left(-(\sigma\epsilon)^2/2\right)$ and $\epsilon < 1$.

2.3 Secure Multiparty Computation

Secure multiparty computation is a cryptographic tool that is commonly used for evaluating a function between multiple participants without leaking any information beyond what is revealed by the output of the computation. We describe garbled circuits [69] and secret sharing [59] as building blocks of the secure multiparty computation below.

2.3.1 Garbled Circuit

A garbled circuit is a secure multiparty computation protocol used for evaluating any function as well as preserving the privacy of inputs. Garbled circuits are often utilized in a two-party setting. In this protocol, a function is presented as Boolean circuits to be encrypted. TinyGarble [64] is a publicly available library of garbled circuits.

2.3.2 Secret Sharing

Secret sharing is a cryptographic tool used for encoding data into multiple shares such that each share reveals nothing about the original data. The original data itself can then be recovered when shares more than a threshold designated in advance are gathered.

We describe how smod operations defined by Chase et al. [9] can be computed with secret sharing below. The smod operation is defined as $x \bmod C = ((x + C) \bmod 2C) - C$. For instance, for any integer $x \in [-C, C)$, shares of x are generated by $\langle x \rangle_1 = (x + \langle x \rangle_2) \bmod C$, where $\langle x \rangle_2$ is uniformly distributed in $[-C, C)$. The resulting share $\langle x \rangle_1, \langle x \rangle_2$ then reveals nothing about x. Then, x is recovered by computing $x = \langle x \rangle_1 - \langle x \rangle_2 \bmod C$.

3. Problem Description

In this section, we describe privacy-preserving collaborative learning as the primary problem setting and its technical difficulty to be solved in this paper.

3.1 Privacy-preserving Collaborative Learning

Privacy-preserving collaborative learning is collaborative learning where the privacy of training data provided by k participants $P = \{p_1, \cdots, p_k\}$ is preserved against n servers in the training phase and a client in the inference phase. For a privacy-preserving mechanism M and a model $Y = f(g(X_{p_1}), g(X_{p_2}), \cdots, g(X_{p_k}))$ trained by collaborative learning for the entire training datasets $X = \{X_{p_1}, X_{p_2}, \cdots, X_{p_k}\}$, all participants try to compute $M(Y)$ and a client can utilize $M(Y)$ for an inference on any chosen input z, that is, $M(Y(z))$ is computed.

In this paper, the requirements for achieving training data privacy against an adversary are described below. These requirements are the same as in HybridAlpha [67]. In the following requirements, we assume that a participant p_1 and a server H_1 are honest for the sake of convenience.

Privacy of computation: An adversary can collude with $n - 1$ servers and $k - 1$ participants in the training phase, and the servers and participants follow a protocol under the honest-but-curious setting. Then, any information except for $M(Y)$ concerning training data X_{p_1} provided by an honest participant p_1 is not revealed to the adversary.

Privacy of output: An adversary can collude with a client who follows a protocol under the honest-but-curious setting in the inference phase. Then, no information with respect to training data X_{p_1} provided by p_1 is revealed to the adversary except for $M(Y(z))$ for any chosen z.

As discussed in Section 1, attacks that infer training data from gradients during training have been proposed [39], [46]. To protect training data from such attacks, the privacy to protect gradients in the training phase, or more specifically the privacy of computation, is necessary. In addition, even when the privacy of computation is achieved, some attacks [22], [62] can reveal the original training data from inference results. This means that privacy in the inference phase, or in other words the privacy of output, should be guaranteed to protect the training data. Therefore, by achieving both the privacy of computation and the privacy of output, a collaborative learning algorithm that achieves the privacy guarantee for training data in the training and inference phases can be realized.

3.2 Technical Difficulty

The privacy guarantee of existing integration protocols [65], [67] of secure multiparty computation and differential privacy for collaborative learning weakens as the number of participants increases. These protocols allow each participant to perturb model parameters locally by adding noise to these parameters. These then require a central server to compute the mean of the perturbed parameters via secure multiparty computation. Then, the amount of noise in the existing protocols is is often excessively reduced instead of improving accuracy. Consequently, the pri-

vacy guarantee becomes weaker in exchange for improved accuracy. However, the privacy must be guaranteed to prevent inference attacks [22], [62] as described in the previous section.

The aforementioned problem on existing protocols [65], [67] is difficult to solve. Local differential privacy is used to guarantee privacy in collaborative learning [17] in general. Indeed, the existing protocols [65], [67] are extensions of local differential privacy. According to their experimental results [65], [67], accuracy of the local differential privacy decreases significantly in comparison with other settings, for example the non-privacy setting and an integration protocol with secure multiparty computation. More specifically, the existing protocols have utilized secure multiparty computation to remove the noise of local differential privacy for maintaining accuracy. In other words, if local differential privacy is adopted, removing noise by using secure multiparty computation is necessary to maintain accuracy because each user individually generates noise. Besides, when an adversary colludes with participants, the adversary may request the participants to reduce noise in order to infer target participant p_1's training data X_{p_1}. Consequently, an approach based on removing the noise will blatantly reveal X_{p_1}.

Thus, besides the existing works [65], [67] described above, any protocol based on local differential privacy will sacrifice either the privacy guarantee or the accuracy *even if it uses secure multiparty computation*.

4. Design of SPGC

In this section, we present SPGC, a privacy-preserving collaborative deep learning protocol based on the integration of differential privacy and secure multiparty computation. We first describe an overview of SPGC, including the main idea for overcoming the problems described in the previous section and then show its construction and privacy analysis.

4.1 Overview

The key concept of SPGC is to perturb gradients by *creating noise for gradients within secure multiparty computation on servers* in contrast to the existing works [65], [67] which *remove* the noise via secure multiparty computation. Intuitively, both privacy guarantee and accuracy of SPGC are independent of the number of participants because it does not include a process for removing noise. Thus, SPGC can maintain a strong privacy guarantee even if many participants join the protocol. We show the overview of SPGC in **Fig. 1**.

SPGC allows each participant to conceal gradients, which are computed local, via secret sharing at the beginning of the protocol. Then, SPGC requires central servers to gather and aggregate the concealed gradients via secret sharing without recovering the original gradients. The aggregation process for gradients can be executed fast because the secret sharing provides servers with fast arithmetic operations for secure multiparty computation.

Next, the servers recover the aggregated gradients and simultaneously add noise to the gradients within a garbled circuit. The process of recovering the aggregated gradients and simultaneously adding noise to the gradients involves complicated computations on the secret sharing. Thus, we focused on using garbled circuits because they can evaluate any function, and the number of servers is two in SPGC because a garbled circuit is a fast protocol between two parties.

Based on the aforementioned construction, SPGC can provide both the privacy guarantee independent of the number of participants and a practical accuracy in comparison with the use of the local differential privacy.

The construction described above was inspired by the collaborative gradient computation (CGC) by Chase et al. [9], but CGC cannot distinguish positive and negative gradients from each other when a gradient is equal to a modulus of the protocol. We show the incorrectness of CGC in Appendix A.1. In contrast,

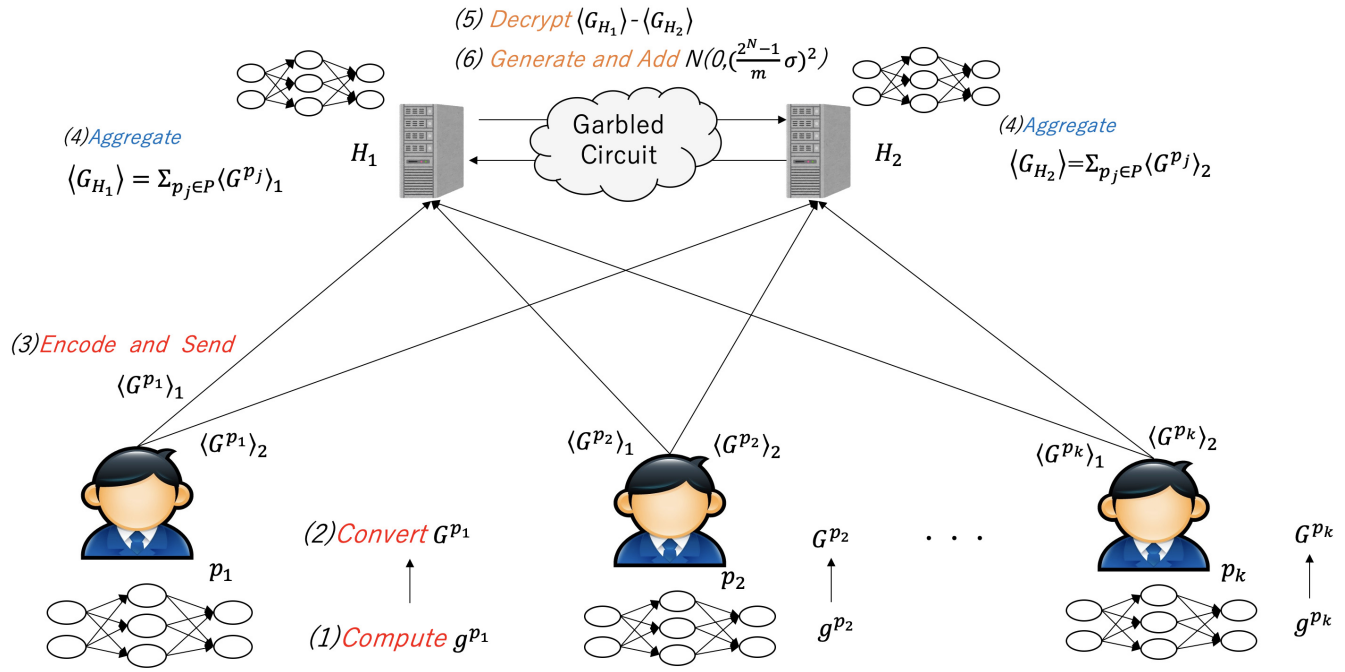


Fig. 1 Overview of SPGC protocol.

Algorithm 1 Participant-Side Process of SPGC.

Input: The current batch B_{t,p_j} in a training step, the current weight w_t in a training step, the entire batch size m, gradient norm bound $C > 0$, modulus 2^N where $N > \log_2(mC + 1)$, loss function $L(\cdot, \cdot)$ used in training.

Output: $\langle G \rangle_1$ sent to a server H_1, $\langle G \rangle_2$ sent to a server H_2.

1: **for** $x_i \in B_{t,p_j}$ **do**
2: $\quad \overline{g}(x_i) = \nabla_{w_t} L(w_t, x_i)$
3: $\quad \tilde{g}(x_i) = \min(1, \frac{C}{\|\overline{g}(x_i)\|_2}) \overline{g}(x_i)$
4: **end for**
5: $g = \sum_{x_i \in B_{t,p_j}} \tilde{g}(x_i)$
6: $G = \frac{2^N - 1}{mC} g$
7: $r = Uniform[-2^N, 2^N)$
8: $\langle G \rangle_1 = G + r \bmod 2^N$
9: $\langle G \rangle_2 = r \bmod 2^N$

SPGC is constructed in a manner where the gradients are *always* smaller than the value of a modulus and therefore it can always train a model correctly.

4.2 Construction

SPGC consists of two algorithms, more specifically one for the participant-side and another for the server-side. Hereafter, we assume the following preconditions of SPGC:

- each participant $p_j \in P$ utilizes a dataset X_{p_j} as a form of batches $\{B_{1,p_j}, B_{2,p_j}, ..., B_{\ell,p_j}\}$ of random examples where ℓ is the number of batches;
- let the entire batch size obtained from P be $m = \sum_{j \in [1,k]} |B_{t,p_j}|$ for any $t \in [1, \ell]$;
- each participant p_j for $j \in [1, k]$ owns an initial model, whose weights w_0 and architecture are identical to those of each other.

The update of weights, more specifically the update of the model itself, is conducted on each batch. In other words, we describe only a gradient computation for a single step as the algorithms of SPGC below.

4.3 Participant-side Process

All the participants run Algorithm 1 in parallel. A participant $p_j \in P$ computes their per-example gradient $\overline{g}(x_i)$ in local. Then, p_j clips the L_2 norm for each gradient $\overline{g}(x_i)$ and computes a summation g of the gradients. This clipping decides the sensitivity S_f, which is utilized for the range of noise, and this is $S_f = C$ in SPGC. Then, p_j converts a floating-point gradient g into a fixed-point gradient G in line 8 because of secret sharing for training. Therefore, p_j encodes G into two shares $\langle G^{p_j} \rangle_1$ and $\langle G^{p_j} \rangle_2$. Finally, outputs $\langle G^{p_j} \rangle_1$ and $\langle G^{p_j} \rangle_2$ of Algorithm 1 are sent to servers H_1 and H_2, respectively.

4.4 Server-side Process

After all the participants have run Algorithm 1, the servers H_1 and H_2 run Algorithm 2. H_1 and H_2 receive shares $\langle G^{p_j} \rangle_1$ and $\langle G^{p_j} \rangle_2$ from each participant p_j, respectively. In doing so, H_1 and H_2 need to wait until they receive the shares from all participants. Moreover, H_1 and H_2 compute $\langle G_{H_1} \rangle_1$ and $\langle G_{H_2} \rangle_2$, respectively. Then, H_1 and H_2 execute garbled circuits.

Next, in line 5, both the recovery of gradients and the generation of noise are executed within the garbled circuits. In particular, the gradients are recovered by computing $(\langle G_{H_1} \rangle_1 - \langle G_{H_2} \rangle_2) \bmod 2^N$. Simultaneously, via a xor computation of s_1 and s_2, a seed for generating noise of the differential privacy is computed. The xor computation within the garbled circuit is performed to conceal the generated noise itself from each server. G^{DP} is then obtained with a noise generated from the seed $s_1 \oplus s_2$ and the recovered gradient $(\langle G_{H_1} \rangle_1 - \langle G_{H_2} \rangle_2) \bmod 2^N$. Finally, g^{DP} is obtained as a floating-point value.

Algorithm 2 Server-Side Process of SPGC.

Input: The participant sets P, $\langle G^{p_j} \rangle_1$ output from participant $p_j \in P$, $\langle G^{p_j} \rangle_2$ output from participant $p_j \in P$, the deviation σ of the Gaussian distribution, the entire batch size m, gradient norm bound $C > 0$, modulus 2^N where $N > \log_2(mC + 1)$.

Output: $g^{DP} = \sum_{p_j \in P} g^{p_j} + \mathcal{N}(0, C^2\sigma^2)$

1: H_1 **:** $\langle G_{H_1} \rangle_1 = \sum_{p_j \in P} \langle G^{p_j} \rangle_1 \bmod 2^N$
2: H_1 **:** generate a random seed s_1
3: H_2 **:** $\langle G_{H_2} \rangle_2 = \sum_{p_j \in P} \langle G^{p_j} \rangle_2 \bmod 2^N$
4: H_2 **:** generate a random seed s_2
5: $G^{DP} = \left(\langle G_{H_1} \rangle_1 - \langle G_{H_2} \rangle_2 \bmod 2^N \right) + \mathcal{N}_{s_1 \oplus s_2}(0, (\frac{2^N - 1}{m} \sigma)^2)$
6: $g^{DP} = \frac{mC}{2^N - 1} G^{DP}$

Note: SPGC may look similar to CGC [9], but its construction is strictly different from CGC, which contains a theoretical problem. CGC might compute a gradient imprecisely when the value of a gradient is equal to a modulus α. To overcome this problem, SPGC restricts the range of fixed-point gradients to be smaller than a modulus by setting $2^N > mC + 1$. Therefore, SPGC can always compute gradients because their values are not equal to a modulus. We also prove in Theorem 1 that SPGC can compute gradients precisely. Our idea enables SPGC to achieve higher accuracy than CGC as well. Furthermore, by creating a modulus in the smod operation in the form of 2^N, the smod operation within garbled circuits becomes faster.

4.5 Correctness

The correctness of gradient computation in SPGC is proven as shown in Theorem 1. To prove Theorem 1, we first present the following lemma.

Lemma 1. *For any modulus $\alpha \in \mathbb{N}$ such that $\alpha > mC > 1$ holds, $|\sum_{i=1}^{m} \frac{\alpha - 1}{mC} \tilde{g}(x_i)| < \alpha$ holds.*

Proof. The proof is shown in Appendix A.2. □

The above lemma restricts a summation of gradients in SPGC to the range of values less than $\pm\alpha$. Here, a modulus α is defined as 2^N in SPGC as shown in Section 4.2. Then, the following theorem is proven.

Theorem 1. *If all servers and participants follow the SPGC protocol, an output of Algorithm 2 is $\sum_{p_j \in P} g^{p_j} + \mathcal{N}(0, C^2\sigma^2)$.*

Proof. H_1 and H_2 compute as follows:

$$\begin{aligned} \langle G_{H_1} \rangle_1 &= \sum_{p_j \in P} \langle G^{p_j} \rangle_1 \bmod 2^N \\ &= \sum_{p_j \in P} (G^{p_j} + r^{p_j} \bmod 2^N) \bmod 2^N \\ &= (\sum_{p_j \in P} G^{p_j} \bmod 2^N + \sum_{p_j \in P} r^{p_j} \bmod 2^N) \bmod 2^N, \end{aligned}$$

and

$$\langle G_{H_2}\rangle_2 = \sum_{p_j\in P}\langle G^{p_j}\rangle_2 \bmod 2^N = \sum_{p_j\in P} r^{p_j} \bmod 2^N.$$

Moreover, the following equation is computed within garbled circuits:

$$(\langle G_{H_1}\rangle_1 - \langle G_{H_2}\rangle_2) \bmod 2^N = \left(\sum_{p_j\in P} G^{p_j} \bmod 2^N\right) \bmod 2^N = \sum_{p_j\in P} G^{p_j} \bmod 2^N \quad (1a) = \sum_{p_j\in P} G^{p_j}. \quad (1b)$$

For each batch, a summation of gradients provided by each participant p_j is equal to a summation of per-example gradients computed from Algorithm 1. That is, $\sum_{i=1}^{m}\frac{\alpha-1}{mC}\tilde{g}(x_i) = \sum_{p_j\in P} G^{p_j}$ holds. Then, the conversion from Eq. (1a) to Eq. (1b) is based on Lemma 1 described above and Lemma 4 in Ref. [9].

Next, the noise $\mathcal{N}(0, d^2)$ generated from the Gaussian distribution is added to the recovered gradients $(\langle G_{H_1}\rangle_1 - \langle G_{H_2}\rangle_2) \bmod 2^N$. The output of G^{DP} is expanded by $\frac{2^N-1}{mC}$, which means to scale back a floating point by multiplying $\frac{mC}{2^N-1}$. After scaling back G^{DP} to g^{DP}, the generated noise is almost equal to the noise generated from the Gaussian distribution $\mathcal{N}(0, C^2\sigma^2)$.

An approximate computation of $g^{DP} = \sum_{p_j\in P} g^{p_j} + \mathcal{N}(0, C^2\sigma^2)$ is then obtained. □

4.6 Privacy Analysis

We analyze the privacy of computation and the privacy of output for SPGC below.

4.6.1 Privacy of Computation

First, we assume that both the garbled circuits and secret sharing achieve privacy of computation. Then, the privacy of computation in the training phase of the SPGC protocol can be achieved following the composition theorem by Kushilevitz et al. [35]. In other words, in Algorithm 2, the computations in lines 1 and 3 are executed via secret sharing, while the computation in line 5 is executed within a garbled circuit. Unless an adversary colludes with both H_1 and H_2, the adversary cannot identify any value except for g^{DP}. In addition, since noise with differential privacy is generated within garbled circuits, an adversary cannot know the accurate noise value. Consequently, no information with respect to training data X_{p_1} is revealed even if an adversary colludes with either H_1 or H_2 and any participant $p_{j\neq 1}$. More formally, following the proof of CGC [9], we can prove the following theorem.

Theorem 2. *If all servers and participants follow the protocol, no information except for an output $g^{DP} = \sum_{p_j\in P} g^{p_j} + \mathcal{N}(0, C^2\sigma^2)$ of Algorithm 2 is revealed to an adversary.*

Proof. In this proof, we discuss from two standpoints, more specifically the participant-side and server-side. On the participant-side, any participant will obtain the end result g^{DP} from H_1, H_2 and nothing else. Hence, without colluding with H_1 and H_2, an adversary cannot reveal any information about g^{p_1}. On the server-side, $\langle G^{p_j}\rangle_1 = G^{p_j} + r^{p_j} \bmod 2^N$ and $\langle G^{p_j}\rangle_2 = r^{p_j} \bmod 2^N$ are uniformly distributed values by virtue of secret sharing as shown in Section 2. If there exists a mechanism M' controlled by an adversary, which is given g^{DP} and $G^{p_j} + r^{p_j}$ and then predicts any property p about G^{p_1}, then the following distribution is obtained from the view of the adversary:

$$\begin{aligned}&\Pr[M'(g^{DP}, \langle G^{p_j}\rangle_1) = p(G^{p_1})]\\ &= \Pr\left[M'(g^{DP}, \langle G^{p_j}\rangle_1) = p(G^{p_1}) | \langle G^{p_j}\rangle_1\right]\\ &= \Pr\left[M('g^{DP}, \langle G^{p_j}\rangle_1) = p(G^{p_1}) | \langle G^{p_j}\rangle_1 \sim \mathrm{Uniform}[-2^N, 2^N)\right].\end{aligned} \quad (2)$$

Therefore, no mechanism M' can reveal any more information other than that viewed by any participant and server, who observe only g^{DP}. □

4.6.2 Privacy of Output

To achieve privacy of output, SPGC needs to satisfy the differential privacy [48], [52]. When adding noise with (ϵ, δ)-differential privacy in one step of training, SPGC constantly satisfies $(\tilde{\epsilon}, \tilde{\delta})$-differential privacy after T epochs for the number of participants k based on the Theorem 3. In contrast, the existing works [65], [67] satisfy $(\sqrt{k}\tilde{\epsilon}, \tilde{\delta})$-differential privacy. In other words, unlike existing works, SPGC can guarantee privacy independent of the number of participants.

Theorem 3. *If all servers and participants follow the protocol and (ϵ, δ)-differential privacy at each batch of the training, SPGC satisfies $(\tilde{\epsilon}, \tilde{\delta})$-differential privacy in the entire process of the training after T epochs, where*

$$\begin{aligned}\tilde{\delta} &= 1 - (1-\delta)^T(1-\delta),\\ \tilde{\epsilon} &= \min\left\{T\epsilon, \frac{(\mathrm{e}^\epsilon - 1)T\epsilon}{\mathrm{e}^\epsilon + 1} + \epsilon\sqrt{2T\log\left(\mathrm{e} + \frac{\sqrt{T\epsilon^2}}{\tilde{\delta}}\right)}, \right.\\ &\qquad \left.\frac{(\mathrm{e}^\epsilon - 1)T\epsilon}{\mathrm{e}^\epsilon + 1} + \epsilon\sqrt{2T\log\left(\frac{1}{\tilde{\delta}}\right)}\right\}.\end{aligned} \quad (3)$$

Proof. For computation on each batch, $\sum_{p_j\in P} g^{p_j} + \mathcal{N}(0, C^2\sigma^2)$ is obtained from an output of Algorithm 2. In doing so, σ is chosen so that (ϵ, δ)-differential privacy is satisfied with each batch. Since a training dataset for each participant is disjointedly divided into ℓ batches, each sample appears at once. Then, a perturbed gradient based on (ϵ, δ)-differential privacy is computed for each batch, and then a model is updated with the gradient. In doing so, SPGC satisfies (ϵ, δ)-differential privacy for each epoch following the parallel composition theorem [27], [38]. Therefore, from Theorem 3.4 by Kairouz et al. [32], an output of the algorithms of SPGC after T epochs satisfies the $(\tilde{\epsilon}, \tilde{\delta})$-differential privacy described above. □

5. Experiments

In this section, we present experiments for measuring the accuracy and training time of SPGC.

5.1 Implementation

We conducted experiments with several academic and medical

diagnosis datasets to measure the performance of SPGC. All the experiments were conducted on a computer having Ubuntu 18.04, 83 GB RAM, Intel Xeon(R) CPU E5-2630 v3 2.40 GHz, and no GPU.

5.1.1 Training

SPGC is implemented mainly with the TensorFlow library [*1] in python, which is an open-source platform developed by Google. Keras in particular is utilized with a TensorFlow backend for training a neural network model. Meanwhile, to restrict the sensitivity during the training, gradients are computed with the `per_example_gradient` operator [25] and then clipped in TensorFlow. When this is easily implemented, the computational complexity becomes $O(m)$ relative to batch size m. Surprisingly, by utilizing the gradient computation proposed by Goodfellow [25], the process is vectorized, more specifically, computations become faster because parallel computations are available. The TensorFlow library contains the `vectorized_map` function for computing the gradient mentioned above, and it is utilized in our implementation.

5.1.2 Server Communication

A secure multiparty computation protocol often needs a heavy communication overhead during protocol execution in addition to the computation itself. The communication process of SPGC is implemented with the gRPC library [*2] in python. The gRPC library is an open-source framework that can design a communication environment with a remote computer and measure training rigorously. Thus, the actual performance of SPGC can be measured via environments with LAN and WAN settings. Meanwhile, garbled circuits used between servers are implemented with the TinyGarble library [64] in C++. The noise generation for differential privacy is directly embedded in the garbled circuits so that the circuits themselves contain the Gaussian distribution, which is identical to a random seed. Consequently, noise identical to a random seed is generated in garbled circuits.

5.2 Experimental Setup

5.2.1 Purpose of Experiments

The purpose of our experiments is to evaluate the accuracy and the training time of SPGC for academic and medical diagnosis datasets. To do this, we first evaluate the accuracy and training time of SPGC relative to the amount of noise for differential privacy. Then, we compare the accuracy and training time of SPGC with the baselines, namely, non-privacy setting, the use of differential privacy, and secure multiparty computation.

5.2.2 Baseline

The following settings are utilized as baselines to compare with the performance of SPGC.

- **Non-privacy**: Train a model without the privacy guarantee. Effects on accuracy and computational overhead for training are evaluated compared to the performance of the Non-privacy setting. The Non-privacy setting significantly differs from the setting of SPGC in that it assumes the use of a single server, which aggregates gradients sent from participants.
- **Local differential privacy (LDP)**: Train a model with a mechanism satisfying local differential privacy (LDP) [17]. LDP is a trivial way for the privacy guarantee in collaborative learning and is also faster than SGPC because it does not use secure multiparty computation. The performance improvement of accuracy and its related overhead on the training phase for SPGC can be evaluated by comparing them with the accuracy and training time for LDP.
- **Only secure multiparty computation (Only-MPC)**: For SPGC, train a model with only garbled circuits and secret sharing instead of noise generation. Original data is recovered from shares within the garbled circuits. The Only-MPC setting is identical to a classic construction [43] based on secure multiparty computation. Accordingly, effects on accuracy and training time with respect to noise generation for the differential privacy in SPGC can be evaluated.

Although one might think to compare SPGC with the existing works [65], [67], their implementations have not been released. An implementation of CGC [9] was not released, but CGC is the motivation of SPGC. Thus, based on experimental results described below, we compare SPGC with CGC by referring to the performance described in Ref. [9] in Section 6. As described by more details later on, we compare only the accuracy of SPGC with CGC since CGC evaluated only the accuracy on the MNIST dataset.

5.2.3 Choice of Parameters in SPGC

On Algorithm 1 and Algorithm 2, let the number of participants be 3, the gradient norm bound C be 1, and N be 16, more specifically a 16-bit integer is utilized. The standard deviation of a noise for differential privacy is computed as $\sigma = \sqrt{2\log(\frac{1.25}{\delta})}/\epsilon$ relative to parameters ϵ and δ of the differential privacy [20]. Meanwhile, noise parameters are $\epsilon = 0.5, 2.0$, and 8.0, which are in common with CGC [9]. These parameters are used also in existing privacy-preserving machine learning [1], [71], for instance the MNIST evaluation in Ref. [1] utilized $\epsilon = 0.5, 2.0$, and 8.0 whereas the Cancer evaluation in Ref. [71] utilized $\epsilon = 8.0$. Likewise, let δ be 10^{-3}. Parameters ϵ and δ in SPGC indicate the privacy level at one step in training. Meanwhile, ϵ and δ in LDP indicate the privacy level when a participant generates noise locally. The privacy levels in both settings may seem to have different values, but an adversary can discover a perturbed gradient by an honest participant in SPGC if the adversary colludes with all participants except for the honest participant. In particular, SPGC provides the same privacy level as when only an honest participant adds noise with a gradient with LDP. We can, therefore, fairly compare the privacy level of SPGC with that of LDP at one step in training.

5.2.4 Datasets and Their Architectures

The datasets used in the experiments and their architectures are shown below.

MNIST: The MNIST dataset contains 70,000 images of handwritten digits from 0 to 9. In particular, the dataset has 60,000 training samples and 10,000 test samples, each with 784 features representing 28×28 pixels in the image. In this experiment, the training samples are equally divided among three participants or more specifically 20,000 samples per participant for training.

[*1] TensorFlow: https://www.tensorflow.org/
[*2] gRPC: https://grpc.io/docs/tutorials/basic/python/

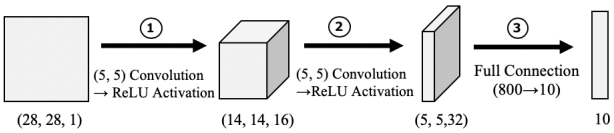


Fig. 2 The architecture used in our experiment for MNIST.

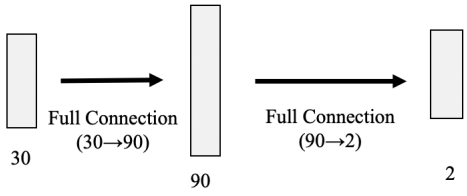


Fig. 3 The architecture used in our experiment for Cancer.

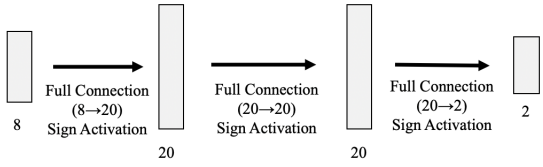


Fig. 4 The architecture used in our experiment for Diabetes.

Furthermore, the batch size for each participant is set to 1,000, that is, the entire batch size m = 3,000. A 3-layer network consisting of the following structure is utilized as shown in **Fig. 2**:

(1) a 5×5 convolution layer with 16 outputs per window, a $2-2$ stride, ReLU activation function, and padding;
(2) a $1 \times 5 \times 5$ convolution layer with 32 outputs per window, a $1-2-2$ stride, no weight sharing on the first axis, and ReLU activation function;
(3) a dense layer with 10 outputs.

Cancer: The Cancer dataset [*3] contains 569 data points with each point containing 30 real-valued features. The inference task is to infer whether a tumor is cancerous or benign. In this experiment, the dataset is divided into 390 training samples and 179 test samples, and then each participant owns 130 training samples for training. The batch size for each participant is set to 10, that is, m = 30. We utilized a single full connection layer from 90 to 2 neurons as shown in **Fig. 3**.

Diabetes: The Diabetes dataset [*4] contains 768 data points with each point containing 8 real-valued features. The inference task is to infer whether a tumor is diabetic or benign. In this experiment, the dataset is divided into 600 training samples and 168 test samples, and then each participant owns 200 training samples for training. The batch size for each participant is set to 10, that is, m = 30. An architecture for Diabetes as shown in **Fig. 4** is the following structure:

(1) a full connection layer with 20 neurons and a Sign activation function;
(2) a full connection layer with 20 neurons and a Sign activation function;
(3) a dense layer with 2 outputs.

[*3] Cancer: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data
[*4] Diabetes: https://www.kaggle.com/uciml/pima-indians-diabetes-database

5.3 Results

The inference accuracy and training time of SPGC with each dataset are described below.

5.3.1 Accuracy

The accuracy of inference of SPGC with the MNIST, Cancer, and Diabetes datasets is shown in **Fig. 5**, **Fig. 6**, and **Fig. 7**, respectively. Under the non-privacy setting as described in Section 5.3.2, the convergence for training is 30 epochs on the MNIST dataset, 30 epochs on the Cancer dataset, and ten epochs on the Diabetes dataset.

MNIST: The accuracy of inference is 97.4% under the non-privacy setting and 97.1% under the Only-MPC setting. In contrast, the accuracy of SPGC is 88.6%, 90.5%, and 92.8% for each noise as presented in Fig. 5 (a), Fig. 5 (b), and Fig. 5 (c), respectively. These results show that SPGC improved accuracy compared to the accuracy of the LDP setting, especially with an intense noise such as $\epsilon = 0.5$. Meanwhile, the SPGC and the LDP setting have almost the same accuracy at epoch 18 for $\epsilon = 8.0$ and $\epsilon = 2.0$. This means that, in proportion to the amount of noise, e.g., in Fig. 5 (a) and Fig. 5 (b), the difference in the accuracy between LDP and SPGC becomes smaller.

Cancer: Both the non-privacy setting and the Only-MPC setting have 98.3% accuracy. Meanwhile, as shown in Fig. 6 (a) and Fig. 6 (b), the accuracy of SPGC is 60.3% and 63.1% for $\epsilon = 0.5$ and $\epsilon = 2.0$, respectively. In contrast, the accuracy for $\epsilon = 8.0$ is 92.1%, as shown in Fig. 6 (c). Besides, in a comparison between SPGC and the LDP setting, as shown in Fig. 6 (a), there is no difference between those accuracies at epoch 30, which is the convergence of the training on the Non-privacy setting, when the amount of noise is large at $\epsilon = 0.5$. Nevertheless, the accuracy of SPGC is 92.1%, and the LDP setting is 86.6% when the amount of noise is small at $\epsilon = 8.0$. In other words, SPGC has higher accuracy of inference than the LDP setting.

Diabetes: Both the non-privacy setting and the Only-MPC setting have 67.2% accuracy. On the other hand, as shown in Fig. 7 (a), the accuracy by SPGC is 33.9%, which is lower than the LDP setting. In contrast, in Fig. 7 (b), the accuracy of SPGC at the convergence of training is 51.8% while that of the LDP setting is 44.0%. Furthermore, in Fig. 7 (c), the accuracy of SPGC is 64.5% while that of the LDP setting is 61.9%.

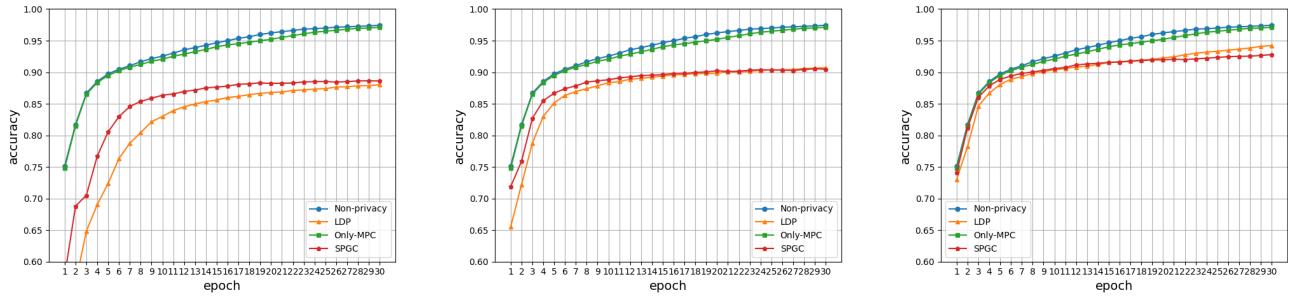
5.3.2 Training Time

Training time of SPGC with MNIST, Cancer and Diabetes datasets are shown in **Fig. 11** (a), Fig. 11 (b) and Fig. 11 (c), respectively.

MNIST: SPGC required 115.7 hours while the Only-MPC setting required only 70.1 hours, a difference of 45.6 hours. Meanwhile, there was a 3-hour difference in the training time between $\epsilon = 8.0$ and $\epsilon = 0.01$. The total training time for SPGC is 72 times longer than that of LDP.

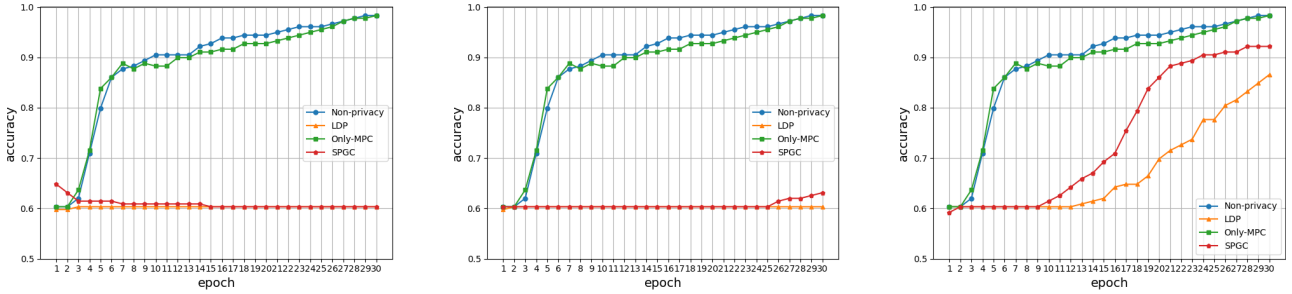
Cancer: SPGC required 10.8 hours for the training while the Only-MPC setting required 6.5 hours, a difference of 4.3 hours. Meanwhile, there was a 0.4 hour difference in the training time between $\epsilon = 8.0$ and $\epsilon = 0.01$. The total training time for SPGC is 98 times longer than that of LDP.

Diabetes: SPGC required 1.23 hours for the training while the Only-MPC setting required 0.80 hours, a difference of 0.44 hours.



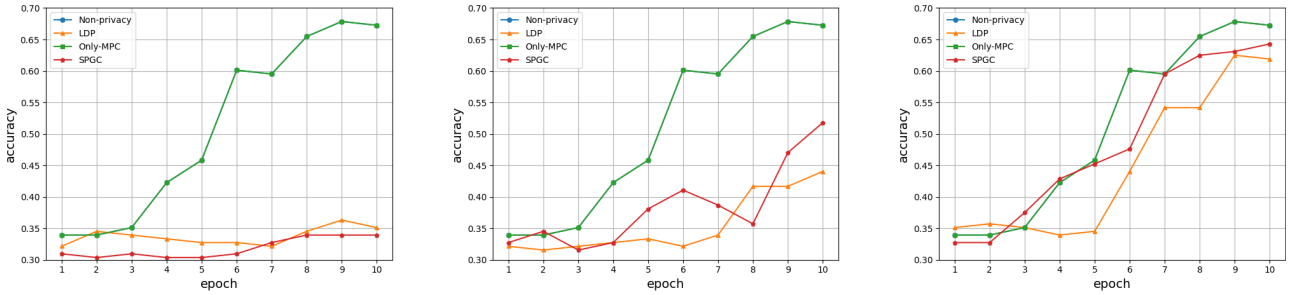
(a) ϵ=0.5 (b) ϵ=2.0 (c) ϵ=8.0

Fig. 5 Inference accuracy of SPGC with MNIST.



(a) ϵ=0.5 (b) ϵ=2.0 (c) ϵ=8.0

Fig. 6 Inference accuracy of SPGC with Cancer.



(a) ϵ=0.5 (b) ϵ=2.0 (c) ϵ=8.0

Fig. 7 Inference accuracy of SPGC with Diabetes.

Meanwhile, there was a 0.05 hours difference in the training time between $\epsilon = 8.0$ and $\epsilon = 0.01$. The total training time for SPGC is 21 times longer than that of LDP.

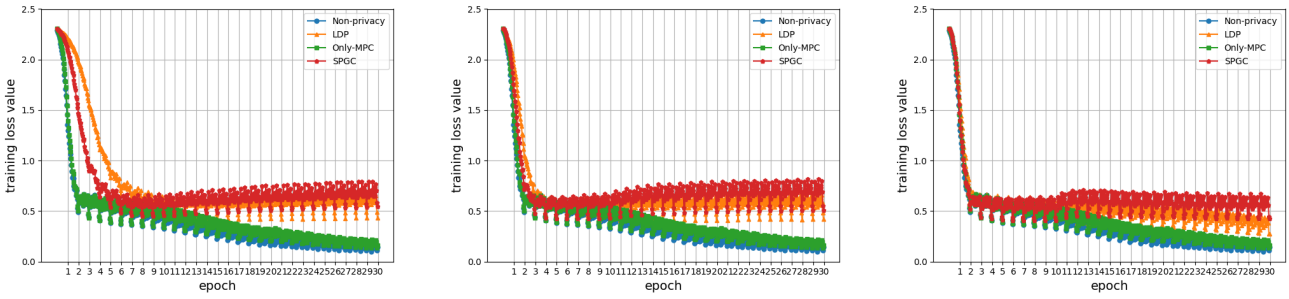
6. Discussion

In this section, we discuss considerations on the accuracy and training time of SPGC and the effect on the number of participants.

6.1 Accuracy

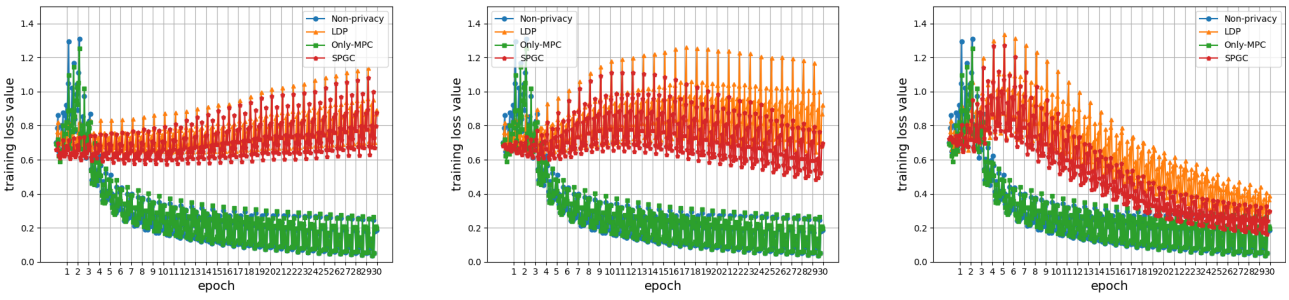
Based on the results shown in Section 5.3.1, we discuss the performance of SPGC in terms of two standpoints, more specifically the difference in accuracy between SPGC and LDP for each dataset and those between the Non-privacy setting and the Only-MPC setting. To do this, we also measure training loss values as shown in **Fig. 8** to **Fig. 10**. Training loss values represent the convergence of the training process. The figures show that the training on each dataset is converged. We note that a periodic change in the training loss values is often caused by typical training even in the Non-privacy setting.

Comparing SPGC with the LDP setting, the accuracy of SPGC on the MNIST dataset is almost the same as the LDP setting for $\epsilon = 8.0$. Specifically, a sufficient number of participants is needed to improve the accuracy of the LDP setting. Since noise is generated from the Gaussian distribution with a mean of zero, the amount of noise becomes close to zero when many participants join the protocols. The number of participants is three in the experimental setting, and the number of participants is insufficient to make the amount of noise zero in the LDP setting. Therefore, LDP achieved the same accuracy for $\epsilon = 8.0$ as SPGC. For $\epsilon = 0.5$, SPGC can provide higher accuracy than LDP because much noise is provided in the LDP setting. In contrast, for $\epsilon = 8.0$, the accuracy of LDP became higher than SPGC after epoch 23. The result is thought to stem from the bit truncation caused by the use of secure multiparty computation, more specifically the use of modulo operations. In this setting, the bit truncation by the secure multiparty computation affects accuracy significantly more than the noise itself.



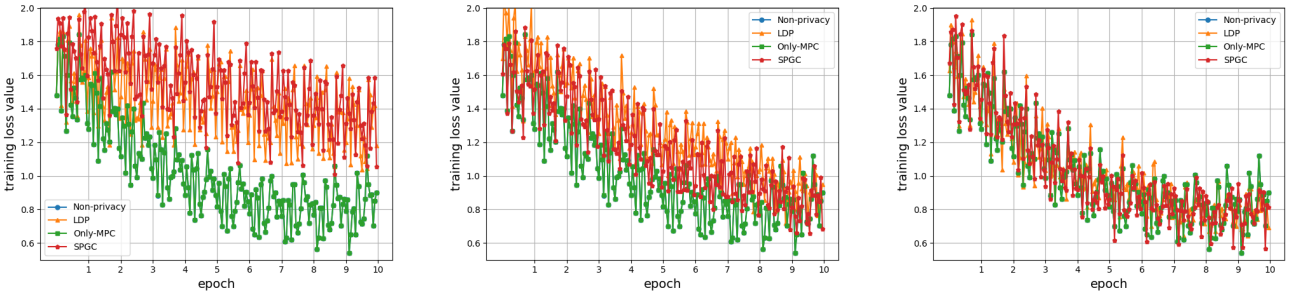
(a) ϵ=0.5 (b) ϵ=2.0 (c) ϵ=8.0

Fig. 8 Training loss value of SPGC with MNIST.



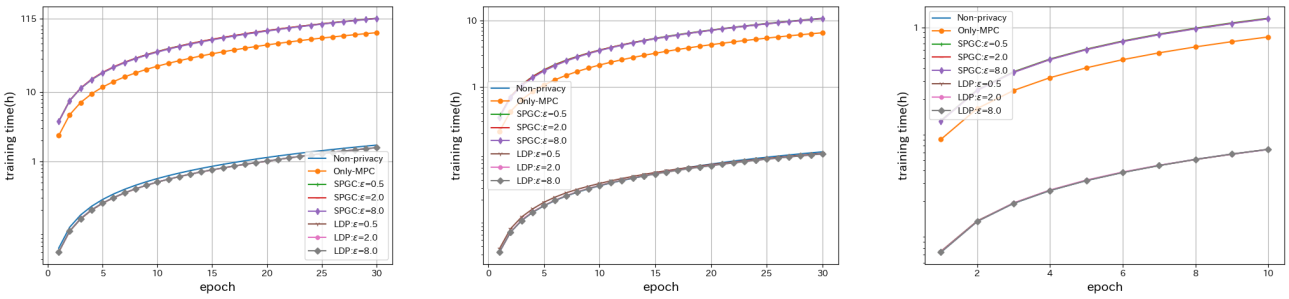
(a) ϵ=0.5 (b) ϵ=2.0 (c) ϵ=8.0

Fig. 9 Training loss value of SPGC with Cancer.



(a) ϵ=0.5 (b) ϵ=2.0 (c) ϵ=8.0

Fig. 10 Training loss value of SPGC with Diabetes.



(a) MNIST (b) Cancer (c) Diabetes

Fig. 11 Training time of SPGC with each dataset.

Likewise, on evaluation of Cancer, SPGC can provide higher accuracy for $\epsilon = 8.0$. Meanwhile, for $\epsilon = 0.5$ and $\epsilon = 2.0$, the accuracy of SPGC is decreased in proportion to epochs. The decrease was caused because noise for $\epsilon = 0.5$ and $\epsilon = 2.0$ are too strong for the Cancer dataset, as shown in Fig. 6. Samples near the decision boundary are often misclassified due to these strong noises. Namely, only the noise for $\epsilon = 8.0$ is suitable for the Cancer dataset, and then the relation between SPGC and the LDP setting is familiar as seen in Fig. 5 (a). Consequently, the figures show that SPGC can control noise better than LDP.

Nevertheless, for $\epsilon = 0.5$ in Diabetes, the accuracy worsens in proportion to the number of epochs on both SPGC and the LDP setting. Because this noise is too large relative to the original data features, inference became entirely random. Meanwhile, for $\epsilon = 2.0$ in the evaluation of Diabetes, SPGC improved the accuracy after epoch 26 compared to the LDP setting. This improve-

Table 1 Details on training time: for each dataset, let Training(h) be the training time until convergence, Offline(h) be gradient computation time as shown in lines 1–5 of Algorithm 1, Online(h) be the remaining time of Algorithm 1 and Algorithm 2. The values enclosed in ( ) are under the LDP setting.

	MNIST			Cancer			Diabetes		
	Offline	Online	Training	Offline	Online	Training	Offline	Online	Training
$\epsilon = 0.01$	1.69(1.53)	115.77(0.04)	117.45(1.57)	0.11(0.11)	10.93(0.01)	11.03(0.12)	0.07(0.07)	1.19(0.00)	1.26(0.07)
$\epsilon = 0.1$	1.66(1.53)	114.47(0.04)	116.13(1.57)	0.11(0.10)	10.50(0.01)	10.61(0.11)	0.07(0.06)	1.20(0.00)	1.27(0.06)
$\epsilon = 0.5$	1.67(1.55)	114.05(0.04)	115.7(1.59)	0.11(0.10)	10.65(0.01)	10.76(0.11)	0.07(0.06)	1.16(0.00)	1.23(0.06)
$\epsilon = 2.0$	1.67(1.55)	113.45(0.04)	115.1(1.59)	0.11(0.09)	10.52(0.00)	10.63(0.10)	0.07(0.06)	1.15(0.00)	1.22(0.06)
$\epsilon = 8.0$	1.67(1.55)	112.78(0.04)	114.5(1.59)	0.11(0.09)	10.53(0.00)	10.63(0.10)	0.07(0.06)	1.14(0.00)	1.21(0.06)
Only-MPC	1.66	70.16	71.82	0.11	6.40	6.51	0.07	0.73	0.80
Non-privacy	1.68	0.05	1.73	0.10	0.01	0.11	0.06	0.00	0.06

ment is thought to stem from greater control of noise by SPGC than in the LDP setting.

Next, the difference in accuracy between the Only-MPC setting and Non-privacy setting is caused by the bit truncation from the conversion to the fixed points for modulo operations. More specifically, in the evaluation of MNIST, the accuracy of the Only-MPC setting decreased by a few percent because of the bit truncation since MNIST contains more features than the other two datasets. Likewise, in the evaluation of Cancer, the bit truncation affected the accuracy for lower epochs than 27. However, the accuracy of the Only-MPC setting is identical to Non-privacy at the convergence of the training, which is epoch 30. In contrast, on the evaluation of Diabetes, the accuracy of the Only-MPC setting is identical to that of Non-privacy because the number of data samples is small, and there is no sample whose evaluation result is changed by the bit truncation. This means that the Only-MPC setting does not affect accuracy for medical diagnosis datasets.

Finally, we compare the accuracy of SPGC with CGC [9]. An implementation of CGC has not yet been released so we refer here to the accuracy of the MNIST evaluation described in the original CGC paper [9]. For instance, the accuracies of CGC are 82.3% for $\epsilon = 0.5$, 87.8% for $\epsilon = 2.0$, and 92.1% for $\epsilon = 8.0$, respectively. On the other hand, those of SPGC are 88.6% for $\epsilon = 0.5$, 90.5% for $\epsilon = 2.0$, and 92.8% for $\epsilon = 8.0$, respectively. We believe that the above advantage of SPGC over CGC is by virtue of our conversion from a floating-point gradient into a fixed-point gradient, more specifically revising the incorrectness of CGC. The above advantage becomes more obvious in proportion to the noise parameter ϵ.

6.2 Training Time

We discuss the effect on the training time for SPGC. According to Figs. 8–10, the convergence of the training process in SPGC and the LDP setting is slower than the Non-privacy setting. This is considered an adverse effect caused by the noise for differential privacy. Namely, gradients may draw away the convergence due to the noise generated from the Gaussian distribution. Hence, the periodic change of the training loss values becomes larger, and consequently, the convergence becomes slower.

Also, training time increases when an amount of noise increases as shown in Section 5.3.2 in Fig. 11 (a). This increase is caused by two computations, the gradient computations by participant in lines 1–4 on Algorithm 1 and the communication overhead between two servers in line 5 on Algorithm 2. To provide further evidence from the standpoints described above, we conduct more experiments with additional parameters $\epsilon = 0.01$, $\epsilon = 0.1$ for each dataset, and then discuss the results (including those in the previous section) in detail. More specifically, we measure the computational time for gradients in lines 1–6 on Algorithm 1 and the communication overhead in Algorithm 2. We show the results in **Table 1**.

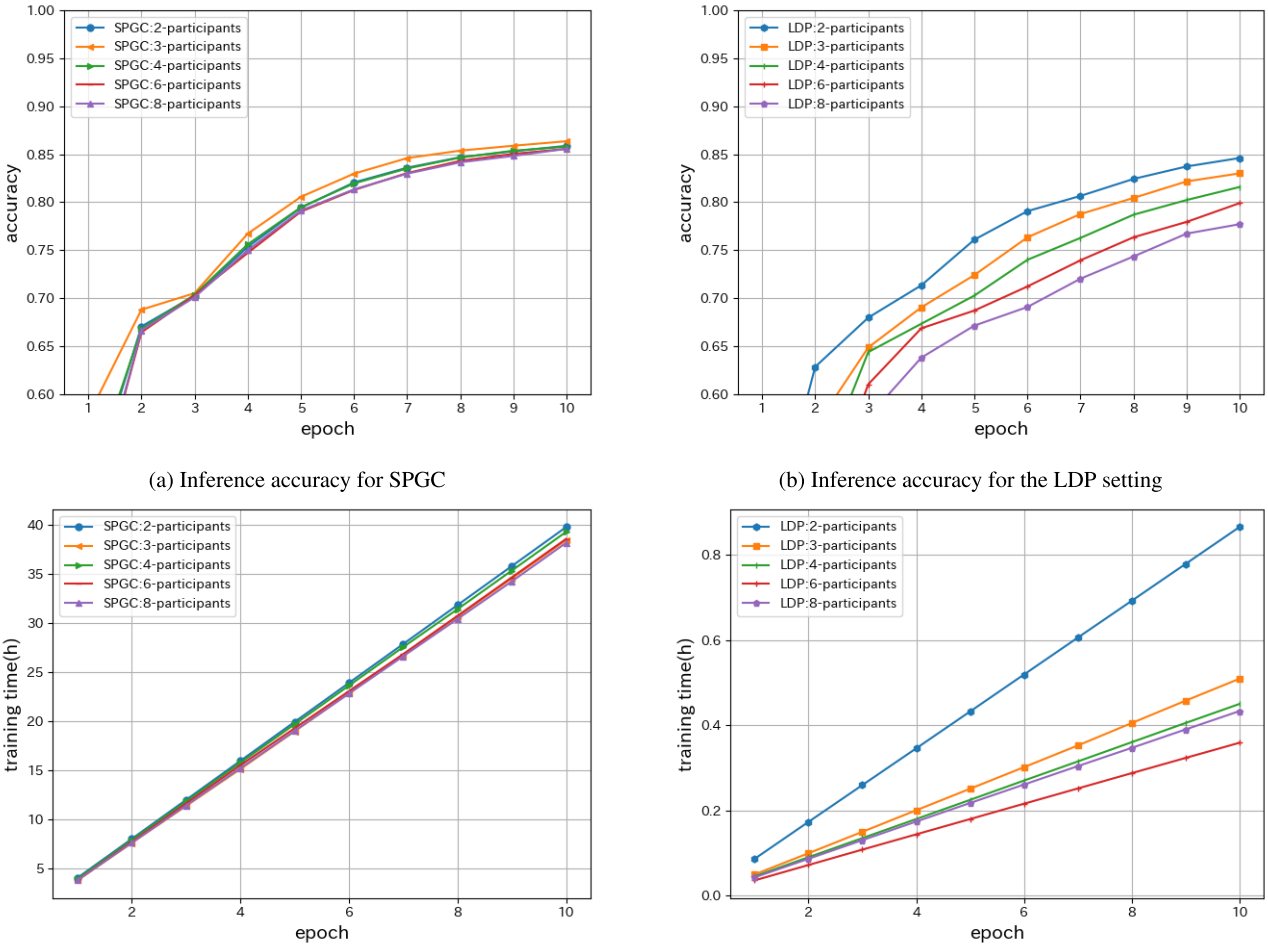
Table 2 Communication overhead (GB): for each dataset, these values are measured between H_1 and H_2.

	MNIST	Cancer	Diabetes
$\epsilon = 0.01$	7,758.0	736.9	81.6
$\epsilon = 0.1$	7,746.9	707.8	78.4
$\epsilon = 0.5$	7,641.9	711.7	78.9
$\epsilon = 2.0$	7,641.9	699.8	77.5
$\epsilon = 8.0$	7,528.2	698.8	77.5
Only-MPC	433.8	39.5	4.4

According to Table 1, for all datasets, the computational time for Algorithm 2 increases in proportion to ϵ, more specifically in the amount of noise. In contrast, the computational time for gradients is almost stable. This means that the training time depends on the operations of the server-side. Indeed, by evaluating the computational time for the algorithms in detail, we discovered that the server-side operations consumed about 98% of the total computational time. Although SPGC needs significantly heavier online results than the LDP setting, we recall that the accuracy of the LDP setting deteriorates, as described in Section 6.1. For instance, for $\epsilon = 0.1$ on the MNIST evaluation, the accuracy of SPGC is 83.6% while the LDP setting is 78%. Likewise, for $\epsilon = 8.0$ on the Cancer evaluation, the accuracy of SPGC is five points higher than the LDP setting. Thus, it is considered that SPGC still has an advantage because its accuracy is higher in spite of worse online results.

More specifically, a large majority of the computational time is caused by communication for garbled circuits between servers. **Table 2** shows results on the communication overhead between two servers for each dataset and parameter. According to the table, the computational cost of SPGC significantly differs from the Only-MPC setting for all datasets. The difference was caused by the noise generated within the garbled circuits. Notably, the communication overhead of SPGC becomes more prominent in proportion to the amount of noise, e.g., 433.8 GB at the Only-MPC setting, 7,528.2 GB at $\epsilon = 8.0$ and 7,758.0 GB at $\epsilon = 0.01$.

The reason is that the size of garbled circuits becomes larger to generate noise for the differential privacy in proportion to the noise itself. Let σ' be $\frac{2^N-1}{m}\sigma$. To generate noise with $\mathcal{N}(0, \sigma'^2)$



(a) Inference accuracy for SPGC

(b) Inference accuracy for the LDP setting

(c) Training time for SPGC

(d) Training time for the LDP setting

Fig. 12 Effect of the number of participants on accuracy and training time. We measured the accuracy and the training time for a noise parameter $\epsilon = 0.5$ on the evaluation of the MNIST dataset by changing the number of participants.

within garbled circuits, the scaled standard deviation σ' is multiplied by the noise generated from the Gaussian distribution $\mathcal{N}(0, 1)$. The size of the circuits increases with respect to a value of σ'.

Based on the observation mentioned above, we conclude that the training time increases significantly due to increasing the communication overhead relative to noise generation. Dwork et al. [18] showed that the communication overhead of secret sharing increases due to the amount of noise for differential privacy. Our observation about the relationship between garbled circuits and differential privacy is identical to the result by Dwork et al., except that we use garbled circuits in collaborative learning.

6.3 Number of Participants

To understand the effect from the number of participants on accuracy and training time, we evaluate SPGC with various participants. In our additional experiment, following the experiments presented in Ref. [67], the accuracy and the training time are measured on the MNIST dataset for 3, 6, and 8 participants. Furthermore, we additionally measure for 2 and 4 participants. In doing so, the training data is divided equally among the participants, e.g., 10,000 data per participant for the 6-participant setting, and $\epsilon = 0.5$ is utilized as the noise parameter. The experiment results are measured until ten epochs and are then compared with the accuracy and the training time on LDP.

6.3.1 Accuracy

The results on the accuracy are shown in **Fig. 12** (a) and Fig. 12 (b). The accuracy of SPGC is almost stable relative to the number of participants at epoch 10, while the accuracy on the LDP setting is worsening. In particular, the accuracy of SPGC is stable for any number of participants while the LDP setting deteriorates by 1.5 points per participant. We thus confirm that the accuracy of SPGC is independent of the number of participants.

The above result also indicates an advantage of SPGC for the privacy guarantee. Generally speaking, there is a trade-off between accuracy and privacy guarantee, and hence the privacy guarantee may be degraded for the purpose of improving accuracy or in other words reducing the noise. This means that reducing the noise is unnecessary for SPGC because when maintaining accuracy for any number of participants the privacy guarantee is also stable. Consequently, the privacy guarantee by SPGC is independent of the number of participants.

6.3.2 Training time

The results on the training time are shown in Fig. 12 (c) and Fig. 12 (d). The training time of SPGC is stable relative to the number of participants because of the following reasons: (1) the

total size of the dataset is the same for all settings, and thus the computational complexity is constant, and (2) all experiments were conducted on virtual machines via a physical computer, and thus the process synchronization is not affected. The results on the training time of SPGC may be different if multiple physical machines and the WAN environment are utilized. In contrast, the training time of LDP becomes shorter in proportion to the number of participants because the training data each participant owns becomes small. Meanwhile, the training time with eight participants was longer than that for six participants, but this was caused by the `vectorized_map` function. More specifically, in doing parallel computations, the race condition of a computational resource occurred in the current environment. This race condition is therefore thought to make the training time longer.

6.4 Variation of Noise Generation

We discuss further extensions of SPGC from the standpoint of noise generation below. In this work, we utilized the standard arguments [20] for the noise generation in the construction and analyzed the privacy via the composition theorem by Kairouz et al. [32]. Although we avoided the use of the moment accountant by Abadi et al. [1] due to prototype implementation, the moment accountant is potentially available for SPGC. By introducing the moment accountant, we will be able to find a better trade-off between accuracy and privacy guarantee. Meanwhile, a typical way [61] to provide LDP-based collaborative learning is to adopt the Laplace mechanism for the noise generation, whereas SPGC has adopted the Gaussian mechanism, whose privacy guarantee is stronger than the use of the Laplace mechanism. The use of such an intense noise is an advantage by virtue of controlling noise well for SPGC.

6.5 Effect by Clipping Parameter

Clipping parameter C has an enormous impact on the accuracy because C determines the amount of noise and the clipped gradients. Specifically, when the clipping parameter C is small, the amount of noise given to gradients is also small. In other words, the noise becomes large in proportion to the clipping parameter C while its resultant gradients are close to the proper gradients. Although a good way to choose C is the use of the median of the unclipped gradients [1], computing the median between all the participants' gradients may be difficult. Consequently, C should be decided as a common parameter in advance.

To evaluate the effect on the clipping parameter C, we conducted an additional experiment on MNIST for $C = 0.5, 1.0, 2.0$ at $\epsilon = 0.5$. The above parameters for C have also been utilized as a part of the experiment in the existing work [1]. As a result, the accuracies are 89.6% for $C = 0.5$, 88.6% for $C = 1.0$, and 86.8% for $C = 2.0$. According to the result, for $C = 1.0$, the accuracy deteriorates by 1.8 points compared to $C = 2.0$. We leave finding an optimized value of C as an open problem to resolve.

6.6 Limitation

The current construction of SPGC contains several restrictions on collaborative learning.

First, SPGC is secure only against an honest-but-curious adversary, and a malicious adversary who ignores the protocol is outside the scope of our work.

Second, SPGC needs the synchronized process to wait for receiving gradients from all the participants. Related to the synchronized process, SPGC has two underlying restrictions in the performance. First, the entire training time will become longer from the standpoint of participants who have already finished the training. Furthermore, the dynamic participation [67] which allows participants to leave/join a protocol anytime, e.g., even during the training, is not supported in SPGC.

Third, additional experimental evaluations are necessary. For instance, the current experiments were conducted on the local host environment, and an experiment on the WAN setting is a subject for future work. Likewise, evaluations with a large dataset such as CIFAR-10 should be conducted to measure the performance of SPGC. Noise parameters suitable for each data should be discussed since we have not optimized the amount of noise in the current experiments. We plan to conduct experiments on these settings in the future.

7. Related Works

In this section, we describe related works on integration of secure multiparty computation and differential privacy, privacy-preserving machine learning, differential privacy, and secure multiparty computation.

7.1 Integration of Secure Multiparty Computation and Differential Privacy

Dwork et al. [18] proposed a noisy summation protocol based on secret sharing and discussed its communication complexity and circuit depth for secret sharing. Many subsequent works have focused on basic computations such as median [4], [51], [63] and summation [26], [53], [60] with respect to the integration of secure multiparty computation and differential privacy. Several applications, such as a generic architecture for computing statistics [21] and a password protection scheme [45], were also proposed.

In recent years, applications of privacy-preserving methods to deep learning [9], [65], [67] have been presented. We describe details of existing protocols below. SPGC was inspired by CGC [9], which is the closest work to ours. We found a pitfall in CGC where the training fails as described in Section 4 and revised it, and thus SPGC is significantly better. Furthermore, the throughput was improved potentially by introducing fixed-point operations in the algorithms. Ryffel et al. [56] developed a PyTorch-based library. Their main motivation is to serve as a public library that leads researchers to investigate an integration protocol of differential privacy and secure multiparty computation. However, they did not provide experiments on datasets with heavy features or deep and rigorous discussion. Finally, HybridAlpha [67] and the work by Truex et al. [65] aim to optimize noise for differential privacy and mitigate the risks of the privacy leakage. However, the privacy guarantee of these works weakens as the number of participants increases, and thus the noise is uncontrollable. Also, they did not give explicit consideration to the differences in the performance of protocols with and without differential privacy.

Nonetheless, these works are based on attractive and elegant approaches in the sense of the use of highly advanced cryptography. For instance, HybridAlpha is based on functional encryption [5], and the work by Truex et al. is based on threshold encryption [14]. Readers who are interested in training that leverages advanced cryptography are advised to read these papers.

7.2 Other Privacy-preserving Machine Learning

There are two kinds of major approaches for privacy-preserving machine learning, namely, differential privacy-based learning [1], [47], [61], [63], [72] and secure multiparty computation-based learning [2], [13], [42], [43], [58], [66]. Differential privacy can minimize the computational overhead for training in contrast to the use of secure multiparty computation. However, the security during the training itself is not directly supported. More precisely, in collaborative learning, a large amount of noise is necessary to guarantee privacy during the training and hence its resultant accuracy is significantly downgraded. On the other hand, the secure multiparty computation can protect the training, but training examples themselves might be leaked from outputs of the model by attacks on machine learning [23], [62]. Therefore, the use of secure multiparty computation cannot rigorously guarantee the privacy of training examples. Consequently, both the differential privacy and the secure multiparty computation should be served jointly. Based on the technical backgrounds above, the integration protocols [9], [56], [65], [67] described in the previous section have been proposed.

Meanwhile, several works in literature [6], [7], [11], [16], [31], [34], [37], [41], [50], [54], [55], [57] provide only the privacy of inference instead of training. Although the privacy guarantee on training is outside the scope of these works, some of them [11], [31], [50], [55] have improved the computational performance by combining multiple secure multiparty computation protocols, such as the combination of secret sharing and garbled circuits. The latest works [34], [54] provide cryptographic compilers generating code to ensure the computation privacy from TensorFlow code. Readers interested in the combination of secure multiparty computations for machine learning are advised to read these papers.

7.3 Differential Privacy

Differential privacy proposed by Dwork et al. [19] has been extended in many forms so [36], [40], [68]. Local differential privacy [17] is an extension model of differential privacy whereby each data owner generates and applies noise to its data, and then a model on a central server is trained by receiving the data with noise from each data owner. Local differential privacy is a practical approach to support training among multiple data owners. A recent work [44] has found some optimized results for the utility on local differential privacy. These state-of-the-art results on differential privacy can be used to improve the accuracy of inference on SPGC in several use cases. The accuracy of inference on SPGC in several use cases can be improved by utilizing these state-of-the-art results on differential privacy.

According to the composition theorem [32], the overall privacy level degrades under the composition of interactive queries where each query meets a specific differential privacy guarantee. Although protecting a model from the attacks described in the previous section via the differential privacy is guaranteed in a formal way [70], Jayaraman and Evans [29] have experimentally proven that noise can be small enough to prevent such attacks. A vital issue for future research is finding a practical way to mitigate accuracy degradation and the privacy guarantee.

7.4 Secure Multiparty Computation

There are three kinds of secure multiparty computation: secret sharing [59], homomorphic encryption [24], and garbled circuits [69]. These computations can handle different operations. Secret sharing-based protocols [3], [15], [42] and garbled circuit-based protocols [15], [42], [64] for example, provide comparison operations necessary for machine learning. Notably, the garbled circuits can deal with any circuit and provide support for a wide range of operations. TinyGarble [64], which is a library of garbled circuits, is utilized in our implementation. In recent years, developments in secure multiparty computation aim to support the design of privacy-preserving machine learning [8], [10], [49].

In contrast, homomorphic encryption-based protocols do not require communication for their computations, but their range of operations is smaller than the other two protocols. Accordingly, several works [12], [30], [33] in the past years have focused on the use of homomorphic encryption. HEAAN [12] is a library optimized for data analysis that can outperform all previous protocols. The design and development of a privacy-preserving machine learning protocol based on these homomorphic encryption-based protocols will be a challenge for future years.

8. Conclusion

In this paper, we present SPGC, a collaborative learning framework integrating secure multiparty computation and differential privacy. While existing protocols [65], [67] remove noise for differential privacy via secure multiparty computation, SPGC creates noise within the secure multiparty computation. Moreover, SPGC has formally overcome the pitfall of CGC by Chase et al. [9], in which the training fails. We also conducted extensive experiments with academic and medical diagnosis datasets. For instance, the training time of SPGC with the Cancer dataset was carried out in about 10.8 hours until convergence of training, and SPGC achieved an accuracy of 92.1%, which is 5.6% higher than that of the naive LDP-based approach. The above higher accuracy of SPGC also indicates that SPGC can provide a privacy guarantee independent of the number of participants. Our results also reveal that the training time becomes longer in proportion to the amount of noise. We are confident that our results can be used to improve the security and privacy of collaborative learning. In the future, we plan to conduct additional experiments with more complicated and more profound architectures, e.g., CIFAR-10.

Acknowledgments This work is supported by the Cabinet Office (CAO), Cross-ministerial Strategic Innovation Promotion Program (SIP), Cyber Physical Security for IoT Society (funding agency: NEDO). We would also like to appreciate anonymous reviewers for their valuable comments.

References

[1] Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K. and Zhang, L.: Deep Learning with Differential Privacy, *Proc. CCS*, pp.308–318, ACM (2016).

[2] Agrawal, N., Shahin Shamsabadi, A., Kusner, M.J. and Gascón, A.: QUOTIENT: Two-Party Secure Neural Network Training and Prediction, *Proc. CCS*, pp.1231–1247, ACM (2019).

[3] Bogdanov, D., Laur, S. and Willemson, J.: Sharemind: A Framework for Fast Privacy-Preserving Computations, *Proc. ESORICS*, LNCS, Vol.5283, pp.192–206, Springer (2008).

[4] Bohler, J. and Kerschbaum, F.: Secure Sublinear Time Differentially Private Median Computation, *Proc. NDSS*, Internet Society (2020).

[5] Boneh, D., Sahai, A. and Waters, B.: Functional Encryption: Definitions and Challenges, *Proc. TCC*, LNCS, Vol.6597, pp.253–273, Springer (2011).

[6] Bourse, F., Minelli, M., Minihold, M. and Paillier, P.: Fast Homomorphic Evaluation of Deep Discretized Neural Networks, *Proc. CRYPTO*, LNCS, Vol.10993, pp.483–512, Springer (2018).

[7] Byali, M., Chaudhari, H., Patra, A. and Suresh, A.: FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning, *Proc. Privacy Enhancing Technologies*, Vol.2, pp.459–480 (2020).

[8] Chandran, N., Gupta, D., Rastogi, A., Sharma, R. and Tripathi, S.: EzPC: Programmable and Efficient Secure Two-Party Computation for Machine Learning, *Proc. IEEE EuroS&P*, pp.496–511, IEEE (2019).

[9] Chase, M., Gilad-Bachrach, R., Laine, K., Lauter, K. and Rindal, P.: Private Collaborative Neural Network Learning (2017) (online), available from ⟨https://eprint.iacr.org/2017/762⟩.

[10] Chaudhari, H., Choudhury, A., Patra, A. and Suresh, A.: ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction, *Proc. CCSW*, pp.81–92, ACM (2019).

[11] Chaudhari, H., Rachuri, R. and Suresh, A.: Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning, *Proc. NDSS*, The Internet Society (2020).

[12] Cheon, J.H., Kim, A., Kim, M. and Song, Y.: Homomorphic Encryption for Arithmetic of Approximate Numbers, *Proc. ASIACRYPT*, LNCS, Vol.10624, pp.409–437, Springer (2017).

[13] Dalskov, A., Escudero, D. and Keller, M.: Secure Evaluation of Quantized Neural Networks, *Proc. Privacy Enhancing Technologies*, Vol.2020, No.4, pp.355–375 (2020).

[14] Damgård, I. and Jurik, M.: A Generalisation, a Simpli.cation and Some Applications of Paillier's Probabilistic Public-Key System, *Proc. PKC*, LNCS, Vol.1992, pp.119–136, Springer (2001).

[15] Demmler, D., Schneider, T. and Zohner, M.: ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation, *Proc. NDSS*, Internet Society (2015).

[16] Dowlin, N., Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M. and Wernsing, J.: CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy, *Proc. ICML*, pp.201–210 (2016).

[17] Duchi, J.C., Jordan, M.I. and Wainwright, M.J.: Local Privacy and Statistical Minimax Rates, *Proc. FOCS*, pp.429–438, IEEE (2013).

[18] Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I. and Naor, M.: Our Data, Ourselves: Privacy Via Distributed Noise Generation, *Proc. EUROCRYPT*, LNCS, Vol.4004, pp.486–503, Springer (2006).

[19] Dwork, C., McSherry, F., Nissim, K. and Smith, A.: Calibrating Noise to Sensitivity in Private Data Analysis, *Proc. TCC*, LNCS, Vol.3876, pp.265–284, Springer (2006).

[20] Dwork, C. and Roth, A.: The Algorithmic Foundations of Differential Privacy, *Foundations and Trends in Theoretical Computer Science*, Vol.9, No.3–4, pp.211–407 (2014).

[21] Eigner, F., Kate, A., Maffei, M., Pampaloni, F. and Pryvalov, I.: Differentially Private Data Aggregation with Optimal Utility, *Proc. ACSAC*, pp.316–325, ACM (2014).

[22] Fredrikson, M., Jha, S. and Ristenpart, T.: Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures, *Proc. CCS*, pp.1322–1333, ACM (2015).

[23] Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D. and Ristenpart, T.: Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing, *Proc. USENIX Security*, pp.17–32, USENIX Association (2014).

[24] Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattice, *Proc. STOC*, pp.169–178, ACM (2009).

[25] Goodfellow, I.: Efficient Per-Example Gradient Computations (2015).

[26] Goryczka, S. and Xiong, L.: A Comprehensive Comparison of Multiparty Secure Additions with Differential Privacy, *IEEE Trans. Dependable and Secure Computing*, Vol.14, No.5, pp.463–477 (2017).

[27] Hsu, J.: Composition, Verification, and Differential Privacy (2018) (online), available from ⟨https://justinh.su/files/slides/tpdp18-invited.pdf⟩.

[28] Iwahana, K., Yanai, N., Cruz, J.P. and Fujiwara., T.: SPGC: An Integrated Framework of Secure Computation and Differential Privacy for Collaborative Learning, *Proc. DPM*, Lecture Notes in Computer Science, Springer (2021).

[29] Jayaraman, B. and Evans, D.: Evaluating Differentially Private Machine Learning in Practice, *Proc. USENIX Security*, pp.1895–1912, USENIX Association (2019).

[30] Jiang, X., Kim, M., Lauter, K. and Song, Y.: Secure Outsourced Matrix Computation and Application to Neural Networks, *Proc. CCS*, pp.1209–1222, ACM (2018).

[31] Juvekar, C., Vaikuntanathan, V. and Chandrakasan, A.: GAZELLE: A Low Latency Framework for Secure Neural Network Inference, *Proc. USENIX Security*, pp.1651–1668, USENIX Association (2018).

[32] Kairouz, P., Oh, S. and Viswanath, P.: The Composition Theorem for Differential Privacy, *IEEE Trans. Information Theory*, Vol.63, No.6, pp.4037–4049 (2017).

[33] Khedr, A., Gulak, G. and Vaikuntanathan, V.: SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers, *IEEE Trans. Computers*, Vol.65, No.9, pp.2848–2858 (2016).

[34] Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A. and Sharma, R.: CrypTFlow: Secure TensorFlow Inference, *Proc. IEEE S&P*, pp.1646–1663, IEEE (2020).

[35] Kushilevitz, E., Lindell, Y. and Rabin, T.: Information-theoretically secure protocols and security under composition, *SIAM Journal on Computing*, Vol.39, pp.2090–2112 (2010).

[36] Liu, C., Chakraborty, S. and Mittal, P.: Dependence Makes You Vulnberable: Differential Privacy Under Dependent Tuples, *Proc. NDSS*, pp.21–24, Internet Society (2016).

[37] Liu, J., Juuti, M., Lu, Y. and Asokan, N.: Oblivious Neural Network Predictions via MiniONN transformations, *Proc. CCS*, pp.619–631, ACM (2017).

[38] McSherry, F.D.: Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis, *Proc. SIGMOD*, pp.19–30, ACM (2009).

[39] Melis, L., Song, C., De Cristofaro, E. and Shmatikov, V.: Exploiting Unintended Feature Leakage in Collaborative Learning, *Proc. IEEE S&P*, pp.691–706, IEEE (2019).

[40] Mironov, I.: Rényi Differential Privacy, *Proc. CSF*, pp.263–275, IEEE (2017).

[41] Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W. and Popa, R.A.: Delphi: A Cryptographic Inference Service for Neural Networks, *Proc. USENIX Security*, pp.2505–2522, USENIX Association (2020).

[42] Mohassel, P. and Rindal, P.: ABY3: A Mixed Protocol Framework for Machine Learning, *Proc. CCS*, pp.35–52, ACM (2018).

[43] Mohassel, P. and Zhang, Y.: SecureML: A System for Scalable Privacy-Preserving Machine Learning, *Proc. IEEE S&P*, pp.19–38, IEEE (2017).

[44] Murakami, T. and Kawamoto, Y.: Utility-Optimized Local Differential Privacy Mechanisms for Distribution Estimation, *Proc. USENIX Security*, pp.1877–1894, USENIX Association (2019).

[45] Naor, M., Pinkas, B. and Ronen, E.: How to (Not) Share a Password: Privacy Preserving Protocols for Finding Heavy Hitters with Adversarial Behavior, *Proc. CCS*, pp.1369–1386, ACM (2019).

[46] Nasr, M., Shokri, R. and Houmansadr, A.: Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning, *Proc. IEEE S&P*, pp.739–753, IEEE (2019).

[47] Papernot, N., Song, S., Mironov, I., Raghunathan, A., Talwar, K. and Erlingsson, Ú.: Scalable Private Learning with PATE, *Proc. ICLR* (2018) (online), available from ⟨https://openreview.net/forum?id=rkZB1XbRZ⟩.

[48] Park, C., Hong, D. and Seo, C.: An Attack-Based Evaluation Method for Differentially Private Learning Against Model Inversion Attack, *IEEE Access*, Vol.7, pp.124988–124999 (2019).

[49] Patra, A., Schneider, T., Suresh, A. and Yalame, H.: ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation, *Proc. USENIX Security*, USENIX Association (2021).

[50] Patra, A. and Suresh, A.: BLAZE: Blazing Fast Privacy-Preserving Machine Learning, *Proc. NDSS*, The Internet Society (2020).

[51] Pettai, M. and Laud, P.: Combining Differential Privacy and Secure Multiparty Computation (2015) (online), available from ⟨https://eprint.iacr.org/2015/598⟩.

[52] Rahman, M.A., Rahman, T., Laganière, R., Mohammed, N. and Wang, Y.: Membership Inference Attack against Differentially Private Deep Learning Model, *Transactions on Data Privacy*, Vol.11, No.1, pp.61–79 (2018).

[53] Rastogi, V. and Nath, S.: Differentially Private Aggregation of Distributed Time-Series with Transformation and Encryption, *Proc. SIGMOD*, pp.735–746, ACM (2010).

[54] Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A. and Sharma, R.: CrypTFlow2: Practical 2-Party Secure Inference,

Proc. CCS, pp.325–342, ACM (2020).
[55] Riazi, M.S., Samragh, M., Chen, H., Laine, K., Lauter, K.E. and Koushanfar, F.: XONN: XNOR-based Oblivious Deep Neural Network Inference, *Proc. USENIX Security*, pp.1501–1518, USENIX Association (2019).
[56] Ryffel, T., Trask, A., Dahl, M., Wagner, B., Mancuso, J., Rueckert, D. and Passerat-Palmbach, J.: A generic framework for privacy preserving deep learning, *Proc. PPML with NeurIPS* (2018).
[57] Sanyal, A., Kusner, M., Gascon, A. and Kanade, V.: TAPAS: Tricks to Accelerate (encrypted) Prediction As a Service, *Proc. ICML*, pp.4497–4506 (2018).
[58] Sav, S., Pyrgelis, A., Troncoso-Pastoriza, J.R., Froelicher, D., Bossuat, J., Sousa, J.S. and Hubaux, J.: POSEIDON: Privacy-Preserving Federated Neural Network Learning, *Proc. NDSS*, Internet Society (2021).
[59] Shamir, A.: How to Share a Secret, *Comm. ACM*, Vol.22, No.11, pp.612–613 (1979).
[60] Shi, E., Chan, T.H., Rieffel, E., Chow, R. and Song, D.: Privacy-preserving aggregation of time-series data, *Proc. NDSS*, pp.1–17, Citeseer (2011).
[61] Shokri, R. and Shmatikov, V.: Privacy-Preserving Deep Learning, *Proc. CCS*, pp.1310–1321, ACM (2015).
[62] Shokri, R., Stronati, M., Song, C. and Shmatikov, V.: Membership Inference Attacks Against Machine Learning Models, *Proc. IEEE S&P*, pp.3–18, IEEE (2017).
[63] Smith, A., Thakurta, A. and Upadhyay, J.: Is Interaction Necessary for Distributed Private Learning?, *Proc. IEEE S&P*, pp.58–77, IEEE (2017).
[64] Songhori, E.M., Hussain, S.U., Sadeghi, A.-R., Schneider, T. and Koushanfar, F.: TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits, *Proc. IEEE S&P*, pp.411–428, IEEE (2015).
[65] Truex, S., Baracaldo, N., Anwar, A., Steinke, T., Ludwig, H., Zhang, R. and Zhou, Y.: A Hybrid Approach to Privacy-Preserving Federated Learning, *Proc. AISec*, pp.1–11, ACM (2019).
[66] Wagh, S., Gupta, D. and Chandran, N.: Securenn: 3-party secure computation for neural network training, *Proc. Privacy Enhancing Technologies*, Vol.2019, No.3, pp.26–49 (2019).
[67] Xu, R., Baracaldo, N., Zhou, Y., Anwar, A. and Ludwig, H.: HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning, *Proc. AISec*, pp.13–23, ACM (2019).
[68] Yang, B., Sato, I. and Nakagawa, H.: Bayesian Differential Privacy on Correlated Data, *Proc. SIGMOD*, pp.747–762, ACM (2015).
[69] Yao, A.C.: Protocols for secure computations, *Proc. FOCS*, pp.160–164, IEEE (1982).
[70] Yeom, S., Giacomelli, I., Fredrikson, M. and Jha, S.: Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting, *Proc. CSF*, pp.268–282, IEEE (2018).
[71] Yu, L., Liu, L., Pu, C., Gursoy, M.E. and Truex, S.: Differentially Private Model Publishing for Deep Learning, *IEEE S&P 2019*, pp.332–349, IEEE (2019).
[72] Yu, L., Liu, L., Pu, C., Gursoy, M.E. and Truex, S.: Differentially Private Model Publishing for Deep Learning, *Proc. IEEE S&P*, pp.332–349, IEEE (2019).

Appendix

A.1 Lack of Correctness on CGC

We show a lack of correctness on CGC by Chase et al. [9] below. Loosely speaking, there are cases in which gradients of CGC are not accurately recovered. In particular, for summation of gradients provided by all participants, the summation may be equivalent to $-mC$ or mC, that is, $-mC \leq \sum_i g_i \leq mC$. In doing so, for the use of secret sharing whose modulus is mC, values of $-mC$ and mC are indistinguishable from each other. In other words, the training will become incorrect because gradients that are identical to mC and $-mC$ are dealt with by the same value on the smod operation. The lack of the correctness described above is formally shown through Lemma 2 and Theorem 4 as described below. Here, assume that the L2 norm is utilized as a clipping norm.

At first, in CGC, let $v = \{v_1, v_2, \cdots, v_n\}$ be data, the clip function $Clip(C, v)$ be computed as $min(1, \frac{C}{\|v\|_2})v$ and the gradient computation is defined as $F'(w_t, x_i)$, where w_t is the current weight and x_i is a training example, respectively. The aforementioned gradient computation of CGC is identical to $\tilde{g}(x_i) = \nabla_{w_t} L(w_t, x_i)$ of SPGC in this paper.

Lemma 2. *For any vector $v = \{v_1, v_2, \cdots, v_n\}$ and $C > 0$, an element y_i computed as $\boldsymbol{y} = Clip(C, v)$ is $|y_i| \leq C$ for any $i \in [1, n]$.*

Proof. We prove the lemma by discussing each case with respect to $\|v\|$ and C.

- For $\|v\| \leq C$, the following equation holds:

  $$Clip(C, v) = v = (v_1, v_2, \cdots, v_n).$$

  That is, for any $v_i \in \{v_1, v_2, \cdots, v_n\}$, $|y_i| = |v_i|$ holds. Then, the following inequation holds:

  $$|y_i| = |v_i| \ \leq \sqrt{|v_1|^2 + |v_2|^2 + \cdots + |v_n|^2} \ \leq C,$$

  because of $|v_i| \geq 0$. Furthermore, when $v_j = 0$ holds for any $j \in [1, n]\backslash\{i\}$, $|y_i| = |v_i| = C$ holds. That is, $|y_i| = |v_i| \leq C$ holds for any $v_i \in \{v_1, v_2, \cdots, v_n\}$. Therefore, for any $y_i \in \{y_1, y_2, \cdots, y_n\}$, $|y_i| \leq C$ holds.
- For any $\|v\| > C$, the following equation holds:

  $$Clip(C, v) \ = \frac{C}{\|v\|} v \ = \frac{C}{\sqrt{|v_1|^2 + |v_2|^2 + \cdots + |v_n|^2}} v.$$

  That is, for any $v_i \in \{v_1, v_2, \cdots, v_n\}$, $|y_i| = \frac{C}{\sqrt{|v_1|^2+|v_2|^2+\cdots+|v_n|^2}}|v_i|$ holds. Then, the following equation holds:

  $$|y_i| \ = \frac{C}{\sqrt{|v_1|^2 + |v_2|^2 + \cdots + |v_n|^2}} |v_i| \ \leq \frac{C}{\sqrt{|v_i|^2}} |v_i| \ = C,$$

  because of $|v_i| \geq 0$ holds. Furthermore, when $v_j = 0$ holds for any $j \in [1, n]\backslash\{i\}$, $|y_i| = \frac{C}{\sqrt{|v_1|^2+|v_2|^2+\cdots+|v_n|^2}}|v_i| = C$ holds. That is, $|y_i| = \frac{C}{\sqrt{|v_1|^2+|v_2|^2+\cdots+|v_n|^2}}|v_i| \leq C$ holds for any $v_i \in \{v_1, v_2, \cdots, v_n\}$. Therefore, for any $y_i \in \{y_1, y_2, \cdots, y_n\}$, $|y_i| \leq C$ holds. □

Next, we show that CGC lacks correctness by presenting a counterexample where gradients are not accurately recovered to an original value. Intuitively, since a modulus on CGC is mC, gradients whose values are mC and $-mC$ are indistinguishable from each other when gradients are equivalent to mC. This fact is presented as the following theorem.

Theorem 4. *Let a per-example gradient for a sample $x_i \in \{x_1, x_2, \cdots, x_m\}$ be $\tilde{g}(x_i) = Clip(C, F'(w_t, x_i))$. There is then a case where a summation of all the gradients is mC and corresponds to $-mC$ in smod operation.*

Proof. First, $|\tilde{g}(x_i)| \leq C$ holds for per-example gradients computed by each participant in accordance with Lemma 2. Then, a summation of gradients given from all the participants is computed as

$$g = \left| \sum_{i=1}^{m} \tilde{g}(x_i) \right| \leq mC. \tag{A.1}$$

Therefore, if $\tilde{g}(x_i) = C$ holds for any $x_i \in \{x_1, x_2, \cdots, x_m\}$, $g = mC$ holds from Equation (A.1). Then, from Theorem 4, for gradients mC,

$$\sum_{i=1}^{m} \tilde{g}(x_i) \; smod \; mC \; = (mC + mC) \bmod 2mC - mC \; = -mC$$

holds from the definition of the smod operation. Gradients with mC then become $-mC$ by the recovering phase of secure multi-party computation and therefore the original value mC cannot be recovered. □

For instance, in case of $F'(w_t, x_i) = \{0, \cdots, j, 0, \cdots, 0\}$ for any $j \in \mathbb{R}$ such that $j > C$ holds, $\tilde{g}(x_i) = C$ holds.

A.2 Proof of Lemma 1

In this appendix, we prove Lemma 1 below. From the assumption where $\alpha > mC > 1$ holds, $\frac{\alpha-1}{mC} > 0$ holds. Then, because $|\tilde{g}(x_i)| \leq C$ holds from Theorem 4, the following relationship holds for $\tilde{g}(x_i)$:

$$\begin{aligned} &-C \leq \tilde{g}(x_i) \leq C \\ &\Leftrightarrow -\frac{(\alpha-1)}{m} \leq \frac{\alpha-1}{mC}\tilde{g}(x_i) \leq \frac{(\alpha-1)}{m}. \end{aligned}$$

Then, on a summation of gradients, the following relationship holds from the relationship described above:

$$\begin{aligned} &-\alpha + 1 \leq \sum_{i=1}^{m} \frac{\alpha-1}{mC}\tilde{g}(x_i) \leq \alpha - 1 \\ &\Leftrightarrow -\alpha < \sum_{i=1}^{m} \frac{\alpha-1}{mC}\tilde{g}(x_i) < \alpha. \end{aligned}$$

Therefore, for any $\alpha \in \mathbb{N}$ such that $\alpha > mC > 1$ holds, $|\sum_i^m \frac{\alpha-1}{mC}\tilde{g}(x_i)| < \alpha$ holds. □

Kazuki Iwahana received B.Eng. degree in Engineering Science from Osaka University, Japan, in 2020. He has recently joined an M.S. course in the Graduate School of Information Science and Technology at Osaka University, Japan. His research interests include information security.

Naoto Yanai received his B. Eng. degree from The National Institution of Academic Degrees and University Evaluation, Japan, in 2009, his M.S. Eng. from the Graduate School of Systems and Information Engineering, the University of Tsukuba, Japan, in 2011, and his Dr.E. degree from the Graduate School of Systems and Information Engineering, the University of Tsukuba, Japan, in 2014. He had been an assistant professor at Osaka University, Japan, from 2014 to 2021 and is an associate professor at Osaka University. His research area is cryptography and information security.

Jason Paul Cruz received his B.S. degree in Electronics and Communications Engineering and M.S. degree in Electronics Engineering from the Ateneo de Manila University, Quezon City, Philippines, in 2009 and 2011, respectively, and his Ph.D. degree in Engineering from the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan in 2017. He is currently a Specially Appointed Assistant Professor at Osaka University, Osaka, Japan. His current research interests include role-based access control, blockchain technology, hash functions and algorithms, privacy-preserving cryptography, and Android programming.

Toru Fujiwara received his B.E., M.E., and Ph.D. degrees in Information and Computer Sciences from Osaka University in 1981, 1983, and 1986. In 1986, he joined the faculty of Osaka University. Since 1997, he has been a Professor at Osaka University. He is currently with the Graduate School of Information Science and Technology. His current research interests include coding theory and information security.