Privacy-Preserving Deep Learning via Additively Homomorphic Encryption

Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai

***Abstract*— We present a privacy-preserving deep learning system in which many learning participants perform neural network-based deep learning over a combined dataset of all, without revealing the participants' local data to a central server. To that end, we revisit the previous work by Shokri and Shmatikov (ACM CCS 2015) and show that, with their method, local data information may be leaked to an honest-but-curious server. We then fix that problem by building an enhanced system with the following properties: 1) no information is leaked to the server and 2) accuracy is kept intact, compared with that of the ordinary deep learning system also over the combined dataset. Our system bridges deep learning and cryptography: we utilize asynchronous stochastic gradient descent as applied to neural networks, in combination with additively homomorphic encryption. We show that our usage of encryption adds tolerable overhead to the ordinary deep learning system.**

***Index Terms*— Privacy, deep learning, neural network, additively homomorphic encryption, LWE-based encryption, Paillier encryption.**

I. Introduction

A. Background

In recent years, *deep learning* (aka, *deep machine learning*) has produced exciting results in both acamedia and industry, where deep learning systems are approaching and even surpassing human-level accuracy. This is thanks to algorithmic breakthroughs and physical parallel hardware applied to *neural networks* for processing massive amount of data.

Massive data collection, while vital for deep learning, raises the issue of privacy. Individually, a collected photo can be permanently kept on a company server, outside the owner's control. Legally, privacy and confidentiality concerns may prevent hospitals and research centers from sharing their medical datasets, barring them from enjoying the advantage of large-scale deep learning over joint datasets.

Manuscript received August 1, 2017; revised October 20, 2017; accepted December 2, 2017. Date of publication December 29, 2017; date of current version January 30, 2018. This work was supported by the Japan Science and Technology Agency CREST under Grant JPMJCR168A. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Ivan Visconti. *(Corresponding author: Le Trieu Phong.)*

L. T. Phong, Y. Aono, L. Wang, and S. Moriai are with the National Institute of Information and Communications Technology, Tokyo 184-8795, Japan (e-mail: phong@nict.go.jp; aono@nict.go.jp; wlh@nict.go.jp; shiho.moriai@nict.go.jp).

T. Hayashi is with the National Institute of Information and Communications Technology, Tokyo 184-8795, Japan, and also with Kobe University, Kobe 657-8501, Japan (e-mail: t-hayashi@eedept.kobe-u.ac.jp).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2017.2787987

Shokri and Shmatikov [28] presented a system for privacy-preserving deep learning that allowed local datasets of several participants staying home while the learned model for the neural network over the joint dataset could be obtained by the participants. To achieve the result, the system in [28] needed the following: each learning participant, using local data, first computed gradients of a neural network; then a portion (e.g. $1\% \sim 100\%$) of those gradients must be sent to a parameter cloud server. The server is *honest-but-curious*. Namely, it is assumed to be *curious* in extracting the data of individuals; and yet, it is assumed to be *honest* in operations.

To protect privacy, the system of Shokri and Shmatikov retained an accuracy/privacy tradeoff (see Table I): sharing no local gradients leads to perfect privacy but not desirable accuracy; on the other hand, sharing all local gradients violates privacy but leads to good accuracy. To compromise, sharing a part of local gradients is the main solution in [28] for maintaining the best accuracy possible.

B. Our Contributions

We demonstrate that, in the system of Shokri and Shmatikov [28], even a small portion of the gradients stored over the cloud server can be exploited. Namely, local data can be surreptitiously extracted from those gradients. Illustratively, we show in Section III a few examples on how a small fraction of gradients leaks useful information on data.

We then propose a novel deep learning system to protect the gradients over the *honest-but-curious* cloud server, using additively homomorphic encryption. All gradients are encrypted and stored on the cloud server. The additive homomorphic property enables the computation across the gradients. Our system is described in Section IV, and depicted in Figure 5, enjoying the following properties on security and accuracy:

Security. *Our system leaks no information of participants to the honest-but-curious parameter (cloud) server.*

Accuracy. *Our system achieves identical accuracy to a corresponding deep learning system (i.e., asynchronous SGD (ASGD)) trained over the joint dataset of all participants.*

In short, our system enjoys the best of both worlds: cryptographic security and deep learning accuracy. See Theorem 1 and Theorem 2 in Section IV.

1) Our Tradeoff: Protecting the gradients against the cloud server comes with the cost of increased communication between the learning participants and the cloud server.

TABLE I
COMPARISON OF TECHNIQUES

System	Method to protect gradients (against the curious server)	Potential information leakage to server?
Shokri-Shmatikov [28]	Partial sharing [28][Sect. 5] (and Laplace noises [28][Sect. 7])	yes
Ours (Section IV)	Additively homomorphic encryption	no

TABLE II
INCREASED COMMUNICATION FACTOR

Our system	**Increased factor** (compared to ordinary Asynchronous SGD)
LWE-based	**2.47** (MNIST dataset [5]), **2.42** (SVHN dataset [24]), **2.40** (speech dataset [16])
Paillier-based	**2.93** (all datasets)

(Using parameters for 128-bit security in encryption)

We show in Table II that the increased factors are not too big: less than three for concrete mixed National Institute of Standards and Technology (MNIST) dataset [5] and Street View House Numbers (SVHN) dataset [24]. For example, in the case of MNIST, if each learning participant needs to communicate 0.56 MB[1] of plain gradients to the server at each upload or download, then with our system using Learning with Errors (LWE)-based encryption, the corresponding communication cost at each upload or download becomes

$$\mathbf{2.47}\ \text{(Table II's factor)} \times \mathbf{0.437}\ \text{(original MB)} \approx \mathbf{1}\ \text{MB}$$

which needs around 8 milliseconds to be transmitted over a 1 Gbps channel. Technical details are in Sections V and VI.

On the computational side, we estimate that our system employing a multilayer perceptron with 109386 gradients finishes in around 2.25 hours to obtain around 97% accuracy when training and testing over the MNIST dataset, which matches results for the same type of neural network given in [2]. Moreover, using the same convolutional neural network as in [28] for MNIST with 105506 gradients, we estimate that our system finishes in around 7.3 hours (with accuracy around 99% identical to that of [28] thanks to Theorem 2).

2) Discussion on the Tradeoffs: We show that the tradeoff between *accuracy/privacy* in [28] can be shifted to *efficiency/privacy* in our system. The accuracy/privacy tradeoff of [28] may make privacy-preserving deep learning less attractive than ordinary deep learning, as accuracy is the main appeal in the field. Our efficiency/privacy tradeoff, keeping ordinary deep learning accuracy intact, can be improved if more processing units and more dedicated programming codes are employed.

3) Threat Model: Throughout this paper, we will consider the server as an honest-but-curious entity, while the learning participants as honest entities. This is relevant to the scenario in which learning participants are considered as organizations such as financial institutions or hospitals[2] acting with responsibilities and by-laws. Below we will discuss other works, particularly [28], in that threat model.

[1]Size of 109386 gradients each of 32 bits; the size is computed via the formula $(109386 \times 32/8 \times 10^6) \approx 0.437$.

[2]For examples, our setting closely captures the one in medical scenario (http://pscanner.ucsd.edu) in which 14 data providers having data from totally 37 million patients.

C. Technical Overviews

A succinct comparison is found in Table I. Below we present the underlying technicalities.

1) Asynchronous SGD (ASGD) [16], [27], No Privacy Protection: Both our system and that of [28] rely on the fact that neural networks can be trained via a variant of SGD called asynchronous SGD [16], [27] with *data parallelism* and *model parallelism*. Specifically, first a global weight vector W_{global} for the neural network is initialized randomly. Then, at each iteration, replicas of the neural network are run over local datasets (i.e. data parallelism), and the corresponding local gradient vector G_{local} is sent to the cloud server. For each G_{local}, the cloud server then updates the global parameters as follows:

$$W_{\text{global}} := W_{\text{global}} - \alpha \cdot G_{\text{local}} \tag{1}$$

where α is a learning rate. The updated global parameters W_{global} are broadcast to all replicas, who then use them to replace their old weight parameters. The process of updating and broadcasting W_{global} is repeated until a desired minimum for a pre-defined cost function (based on cross-entropy or squared-error) is reached. For model parallelism, the update at (1) is computed in parallel via components of the vectors W_{global} and G_{local}.

2) Shokri-Shmatikov Systems: The system in [28, Sec. 5] can be called *gradients-selective* ASGD for the following reasons. In[28, Sec. 5], the update rule at (1) is modified as follows:

$$W_{\text{global}} := W_{\text{global}} - \alpha \cdot G_{\text{local}}^{\text{selective}} \tag{2}$$

in which vector $G_{\text{local}}^{\text{selective}}$ contains selective (say $1\% \sim 100\%$) gradients of G_{local}. The update using (2) allows each participant to choose which gradients to share globally, with the hope of reducing the risk of leaking sensitive information on the participant's local dataset to the cloud server. However, as showed in Section III, a small portion of gradients leaks information to the server.

In[28, Sec. 7], Shokri-Shmatikov showed an additional technique on using differential privacy to counteract indirect leakage from gradients. Their strategy was to add Laplace noises into $G_{\text{local}}^{\text{selective}}$ at (2). Due to noises, this method harms learning accuracy which is the main appeal of deep learning.

3) Our System: Our system can be called *gradients-encrypted* ASGD for the following reasons. In our system in Section IV, we make use of the following update formula

$$\mathbf{E}(W_{\text{global}}) := \mathbf{E}(W_{\text{global}}) + \mathbf{E}(-\alpha \cdot G_{\text{local}}) \tag{3}$$

in which $\mathbf{E}$ is homomorphic encryption supporting addition over ciphertexts. The decryption key is only known to the participants and not to the cloud server. Therefore, the honest-but-curious cloud server knows nothing about each G_{local}, and hence obtains no information on each local dataset of participants. Nonetheless, as

$$\mathbf{E}(W_{\text{global}}) + \mathbf{E}(-\alpha \cdot G_{\text{local}}) = \mathbf{E}(W_{\text{global}} - \alpha \cdot G_{\text{local}})$$

by the additively homomorphic property of $\mathbf{E}$, each participant will get the correctly updated W_{global} via decryption. Moreover, when the original update at (1) is parallelized via components of the vectors W_{global} and G_{local}, our system applies homomorphic encryption to each of the components accordingly.

In addition, to ensure the integrity of the homomorphic ciphertexts, each client will use a secure channel such as TLS/SSL (distinct from each other) to communicate the homomorphic ciphertexts with the server.

4) Extension of Our System: Our idea of using the encrypted update rule as in (3) can be extended to other SGD-based machine learning methods. For example, our system can readily be used with logistic regression with distributed learning participants who each hold a local dataset. In this case, the only change is that each participant will run the SGD-based logistic regression instead of the neural network of deep learning.

D. More Related Works

Gilad-Bachrach *et al.* [18] present a system called *CryptoNets*, which allows homomorphically encrypted data feedforwarding an already-trained neural network. Because *CryptoNets* assumes that the weights in the neural network have been trained beforehand, the system aims at making prediction for individual data items.

The goal of our paper and [28] differ from that of [18], as our system and Shokri-Shmatikov's exactly aim at training the weights via multiple data sources, whereas *CryptoNets* [18] does not.

We consider the cloud server as the adversary in this paper while learning participants are seen as honest entities. Our scenario and adversary model is different from that of Hitaj *et al.* [20] which examines dishonest learning participants.

Mohassel and Zhang [23] examine privacy-preserving methods for linear regression, logistic regression and neural network training over two servers which are assumed to have not colluded. Their model is different from ours.

In the same research line of only using additively homomorphic encryption, privacy-preserving linear/logistic regression systems have been proposed in[8] and [10].

Using secret sharing, Bonawitz *et al.* [14] proposes a secure aggregation method and applies it to deep neural networks to aggregate user-provided model updates. In particular, the work [14] tries to deal with many (e.g., $2^{14} = 16384$ in the experiments or perhaps more practically) mobile devices so that dropouts (failures in completing the protocol) can be frequent. In contrast, we would like to have multiple (e.g., 10) data providers as organizations such as hospitals or banks each of which holds a large dataset on patients or customers so that dropouts are not considered. For examples, our setting closely captures the situation in medical scenario mentioned in Footnote 2. Another application for our case is that multiple banks and/or credit card companies would like to train over a joint dataset of all (e.g., for the task of fraud detection) but they concern about the curiosity of the central server on their plain and sensitive data.

Due to the tool used (secret sharing in [14] vs. additively homomorphic encryption in ours), the condition for security changes, for example honest majority in [14] vs. computational assumption (LWE) in ours. We do not have any threshold for correctness while [14] does. This diversity is good so that choices are ready for specific applications.

The issue of information leakage from the trained model is also important, and yet it is orthogonal to this work. A common tool to address this issue is differential privacy (often with accuracy decline due to noise added), as used in [6] for deep learning.

E. Improvements on the Conference Version

A preliminary version of this paper was at [26]. This full version generalizes the main system, adding multiple processing units at the honest-but-curious server, and allowing the learning participants to upload and download parts of encrypted gradients. In addition, communication costs are revised and computational costs are given anew.

II. Preliminaries

A. On Additively Homomorphic Encryption

Definition 1 (Homomorphic Encryption): Public key additively homomorphic encryption (PHE) schemes consist of the following (possibly probabilistic) poly-time algorithms.

- $\mathsf{ParamGen}(1^\lambda) \rightarrow pp$: λ is the security parameter and the public parameter pp is implicitly fed in following algorithms.
- $\mathsf{KeyGen}(1^\lambda) \rightarrow (pk, sk)$: pk is the public key, while sk is the secret key.
- $\mathsf{Enc}(pk, m) \rightarrow c$: probabilistic encryption algorithm produces c, which is the ciphertext of message m.
- $\mathsf{Dec}(sk, c) \rightarrow m$: decryption algorithm returns message m encrypted in c.
- $\mathsf{Add}(c, c')$: for ciphertexts c and c', the output is the encryption of plaintext addition c_{add}.
- $\mathsf{DecA}(sk, c_{\text{add}})$: decrypting c_{add} to obtain an addition of plaintexts.

Ciphertext indistinguishability against chosen plaintext attacks [19] (or CPA security for short below) ensures that no bit of information is leaked from ciphertexts.

B. On Deep Machine Learning

1) Some Concepts and Notations: Deep machine learning can be seen as a set of techniques applied to neural networks. Figure 1 shows a neural network with 5 inputs, 2 hidden layers, and 2 outputs. The node with $+1$ represents the bias term.

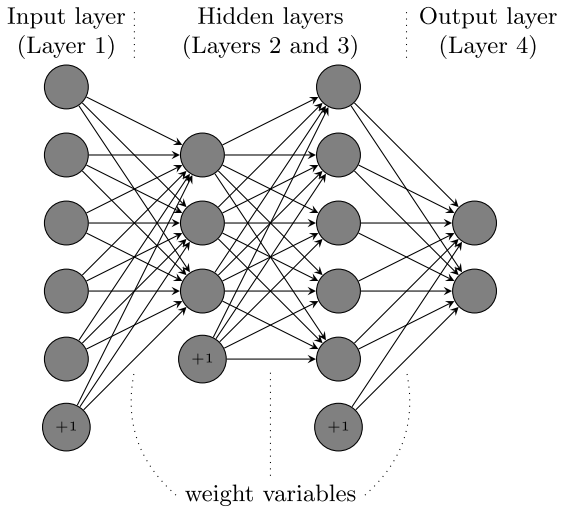


Fig. 1. A neural network with 5 inputs, 2 hidden layers, 2 outputs.

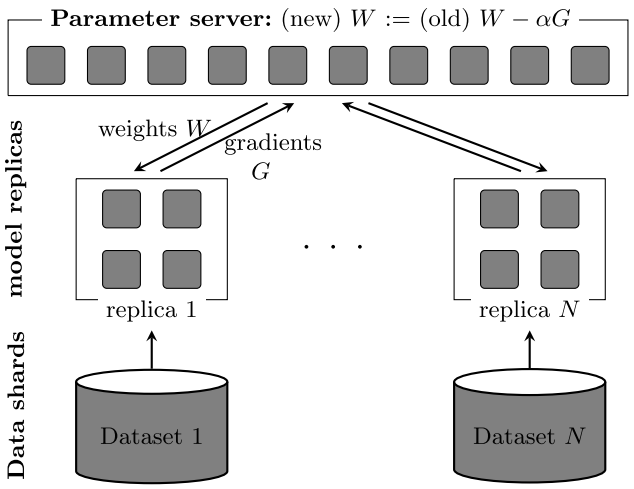


Fig. 2. Asynchronous SGD [16], [27].

The neuron nodes are connected via weight variables. In a deep learning structure of neural network, there can be multiple layers each with thousands of neurons.

Each neuron node (except the bias node) is associated with an *activation function* f. Examples of f in deep learning are $f(z) = \max\{0, z\}$ (rectified linear), $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ (hyperbolic tangent), and $f(z) = (1 + e^{-z})^{-1}$ (sigmoid). The output at layer $l + 1$, denoted as $a^{(l+1)}$, is computed as $a^{(l+1)} = f(W^{(l)}a^{(l)} + b^{(l)})$ in which $(W^{(l)}, b^{(l)})$ are the weights connecting layers l and $l + 1$, and $a^{(l)}$ is the output at layer l.

The learning task is, given a training dataset, to determine these weight variables to minimize a pre-defined cost function such as the cross-entropy or the squared-error cost function [3]. The cost function can be computed over all data items in the training dataset; or over a subset (called mini-batch) of t elements from the training dataset. Denote the cost function for the latter case as $J_{|\text{batch}|=t}$. In the extreme case of $t = 1$, corresponding to maximum stochasticity, $J_{|\text{batch}|=1}$ is the cost function defined over 1 single data item.

2) Stochastic Gradient Descent (SGD): Let W be the flattened vector consisting of all weight variables. Namely we take all weights in the neural network and arrange them consecutively to form the vector W. Denote $W = (W_1, \ldots, W_{n_{gd}}) \in \mathbb{R}^{n_{gd}}$. Let

$$G = \left(\frac{\delta J_{|\text{batch}|=t}}{\delta W_1}, \ldots, \frac{\delta J_{|\text{batch}|=t}}{\delta W_{n_{gd}}} \right) \tag{4}$$

be the gradients of the cost function $J_{|\text{batch}|=t}$ corresponding to variables $W_1, \ldots, W_{n_{gd}}$. The variable update rule in SGD is as follows, for a learning rate $\alpha \in \mathbb{R}$:

$$W := W - \alpha \cdot G \tag{5}$$

in which $\alpha \cdot G$ is component-wise multiplication, namely $\alpha \cdot G = (\alpha G_1, \ldots, \alpha G_{n_{gd}}) \in \mathbb{R}^{n_{gd}}$. The learning rate α can also be changed adaptively as described in[3] and [17].

3) Asynchronous (aka. Downpour) SGD [16], [27]: By (4) and (5), as long as the gradients G can be computed, the weights W can be updated. Therefore, the data used in computing G can be distributed (i.e., data parallelism). Moreover, the update process can be parallelized by considering separate components of the vectors (i.e., model parallelism).

Specifically, as in Figure 2, asynchronous SGD uses multiple replicas of a neural network. Before each execution, each replica downloads the newest weights from the parameter server; and each replica is run over a data shard, which is a subset of the training dataset. To use the power of parallel computation when the server has multiple processing units $\text{PU}_1, \ldots, \text{PU}_{n_{pu}}$, asynchronous SGD splits the weight vector W and gradient vector G into n_{pu} parts, namely $W = (W^{(1)}, \ldots, W^{(n_{pu})})$ and $G = (G^{(1)}, \ldots, G^{(n_{pu})})$, so that the update rule at (5) becomes as follows:

$$W^{(i)} := W^{(i)} - \alpha \cdot G^{(i)} \tag{6}$$

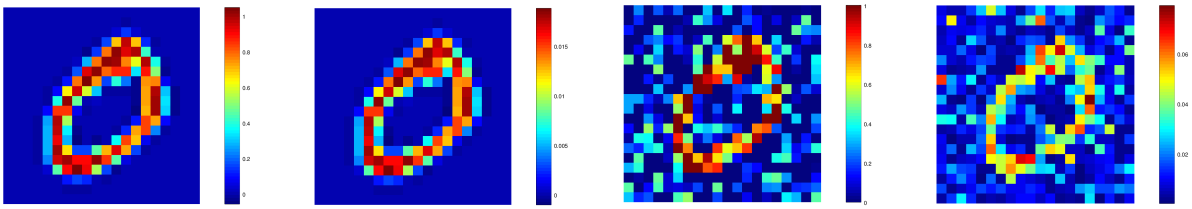
which is computed at processing unit PU_i. As the processing units $\text{PU}_1, \ldots, \text{PU}_{n_{pu}}$ can run in parallel, asynchronous SGD significantly increases the scale and speed of deep network training, as experimentally showed in [16].

III. Gradients Leak Information

This section shows that a small portion of gradients may reveal information on local data.

Example 1 (One Neuron): For illustration of how gradients leak information on data, we first use the neural network in Figure 4, with only one neuron. In the figure, real numbers x_i $(1 \le i \le d)$ are the input data, with a corresponding truth label y; real numbers W_i $(1 \le i \le d)$ are the weight parameters to be learned; and b is the bias. The function f is an activation function (either sigmoid, rectified linear, or hyperbolic tangent as described in Section II-B). The cost function is defined as the distance between the predicted value $h_{W,b}(x) \stackrel{\text{def}}{=} f(\sum_{i=1}^{d} W_i x_i + b)$ and the truth value y:

$$J(W, b, x, y) \stackrel{\text{def}}{=} (h_{W,b}(x) - y)^2$$



(**a**) (**b**) (**c**) (**d**)

Fig. 3. Original data (**a**) vs. leakage information (**b**), (**c**) from a small portion of gradients in a neural network. These images are best seen in color.

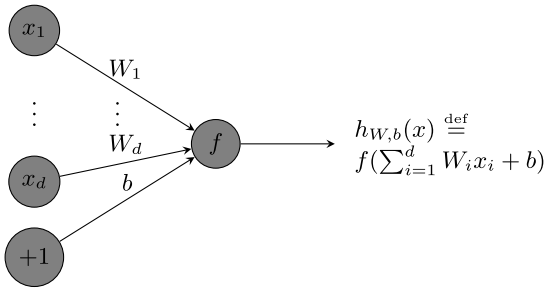


Fig. 4. A network with one neuron.

and hence the gradients are

$$\eta_k \stackrel{\text{def}}{=} \frac{\delta J(W,b,x,y)}{\delta W_k} = 2(h_{W,b}(x)-y)\frac{\delta h_{W,b}(x)}{\delta W_k}$$
$$= 2(h_{W,b}(x)-y)\frac{\delta f(\sum_{i=1}^{d} W_i x_i + b)}{\delta W_k}$$
$$= 2(h_{W,b}(x)-y) f'(\sum_{i=1}^{d} W_i x_i + b)\cdot x_k \tag{7}$$

and

$$\eta \stackrel{\text{def}}{=} \frac{\delta J(W,b,x,y)}{\delta b} = 2(h_{W,b}(x)-y)\frac{\delta h_{W,b}(x)}{\delta b}$$
$$= 2(h_{W,b}(x)-y)\frac{\delta f(\sum_{i=1}^{d} W_i x_i + b)}{\delta b}$$
$$= 2(h_{W,b}(x)-y) f'(\sum_{i=1}^{d} W_i x_i + b)\cdot 1. \tag{8}$$

The k-th component x_k of $x = (x_1,\ldots,x_d) \in \mathbb{R}^d$ or the truth label y can be inferred from the gradients by one of the following means:

(**O1**) Observe that $\eta_k/\eta = x_k$. Therefore, x_k is completely leaked if η_k and η are shared to the cloud server. For example, if 1% of local gradients, chosen randomly as suggested in [28], are shared to the server, then the probability that both η_k and η are shared is $(1/100)\times(1/100) = 1/10^4$, which is not negligible.

(**O2**) Observe that the gradient η_k is proportional to the input x_k for all $1 \le i \le d$. Therefore, when $x = (x_1,\ldots,x_d)$ is an image, one can use the gradients to produce a related "proportional" image, and then obtain the truth value y by guessing.

Example 2 (General Neural Networks, cf. Figure 3(b)): The above observations (**O1**) and (**O2**) similarly hold for general neural networks, with both cross-entropy and squared-error cost functions [3]. In particular, following [3],

$$\eta_{ik} \stackrel{\text{def}}{=} \frac{\delta J(W,b,x,y)}{\delta W_{ik}^{(1)}} = \xi_i \cdot x_k \tag{9}$$

where $W_{ik}^{(1)}$ is the weight parameter connecting layer 1's input x_k with hidden node i of layer 2; ξ_i is a real number.

In Figure 3(b), using a neural network on [1], we demonstrate that gradients at (9) are indeed *proportional* to the original data, as Figure 3(b) only differs from Figure 3(a) at the value bar. The original data is a 20x20 image, reshaped into a vector of $(x_1,\ldots,x_{400}) \in \mathbb{R}^{400}$. The vector is an input to a neural network of 1 hidden layer of 25 nodes; and the output layer contains 10 nodes. The total number of gradients in the neural network is

$$(400+1)\times 25 + (25+1)\times 10 = 10285.$$

At (9), we have $1 \le k \le 400$ and $1 \le i \le 25$. We then use a small part of the gradients at (9), namely $(\eta_{1k})_{1\le k\le 400}$, reshaped into a 20x20 image, to draw Figure 3(b). It is clear that the part (namely $400/10285 \approx 3.89\%$) of the gradients reveals the truth label 0 of the original data.

Example 3 (General Neural Networks, With Regularization, cf. Figure 3(c)): In a neural network with regularization, following [3] we have

$$\eta_{ik} \stackrel{\text{def}}{=} \frac{\delta J(W,b,x,y)}{\delta W_{ik}^{(1)}} = \xi_i \cdot x_k + \lambda W_{ik}^{(1)}$$
$$\eta_i \stackrel{\text{def}}{=} \frac{\delta J(W,b,x,y)}{\delta b_i^{(1)}} = \xi_i$$

where notations are as in Example 2 above; $b_i^{(1)}$ is the bias associated with node i of layer 2; and $\lambda \ge 0$ is a regularization term.

As in observation (**O1**), both η_{ik} and η_i are known to the server with a non-negligible probability. Additionally, in Figure 3(c), we use the following observation:

$$\frac{\eta_{ik}}{\eta_i} = x_k + \frac{\lambda W_{ik}^{(1)}}{\xi_i}$$

which is an approximation of the data x_k. In Figure 3(c), we take $\lambda = 0.1$, and other details identical to Example 2 above. Due to the term $\lambda W_{ik}^{(1)}/\xi_i$, there is noise in the figure, but the truth value (number 0) of the original data can still be seen.

Example 4 (Laplace Noises Added, cf. Figure 3(d)): As differential privacy and secrecy are orthogonal, adding Laplace

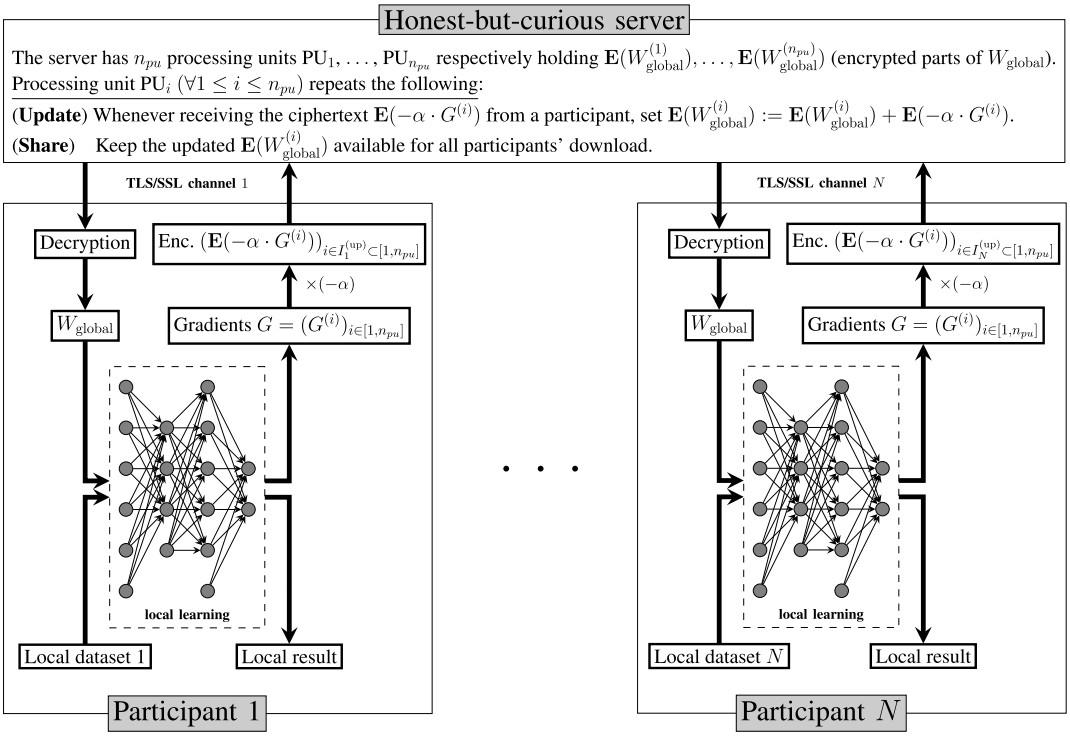


Fig. 5. Our system (*gradients-encrypted* asynchronous SGD) for privacy-preserving deep learning, with a cloud server and N participants.

noises to the gradients may not protect the secrecy of those gradients from the curious server. To illustrate, in Figure 3(d), we added Laplace noises of mean 0 and derivation $\sigma = 1/100$ to gradients η_{ik} at (9), namely

$$\eta_{ik} = \xi_i \cdot x_k + Laplace\left(0, \frac{1}{100}\right)$$

so that the image contains noise, and yet the truth value can still be seen. In addition, when the parameter σ is larger, η_{ik} may be dominated by noise so that the truth label is difficult to be seen, but this may not help in updating the weight variables.

IV. Our System: Privacy-Preserving Deep Learning Without Accuracy Decline

Our system is depicted in Figure 5, consisting of a common cloud server and N (e.g. $= 10 \sim 100$) learning participants.

A. Learning Participants

The participants jointly set up the public key pk and secret key sk for an additively homomorphic encryption scheme. The secret key sk is kept confidential from the cloud server, but is known to all learning participants. Each participant establishes a TLS/SSL secure channel, different from each other, to communicate and protect the integrity of the homomorphic ciphertexts.

Then, the participants locally hold their datasets and run replicas of a deep learning-based neural network. The initial (random) weight W_{global} to run the local neural network is initialized by Participant 1 who also sends $\mathbf{E}(W_{\text{global}}^{(1)}), \ldots, \mathbf{E}(W_{\text{global}}^{(n_{pu})})$ to the server initially, in which $W_{\text{global}}^{(i)}$ is also a vector constituting the i-th part of W_{global}. The gradient vector G obtained after each execution of the neural network is split into n_{pu} parts, namely $G = (G^{(1)}, \ldots, G^{(n_{pu})})$, multiplied by the learning rate α, and then encrypted using the public key pk. The resulting encryption $\mathbf{E}(-\alpha \cdot G^{(i)})$ $(\forall 1 \le i \le n_{pu})$ from each learning participant is sent to the processing unit PU_i of the server. It is also worth noting that the learning rate α can be adaptively changed locally at each learning participant as described in [16].

As seen in Figure 5, each participant $1 \le k \le N$ will perform the following steps:

1. Download the ciphertexts $\mathbf{E}(W_{\text{global}}^{(j)})$ stored at the processing units PU_j of the server for all $j \in I_k^{(\text{down})} \subset [1, n_{pu}]$. Usually $I_k^{(\text{down})} = [1, n_{pu}]$, namely the participant will download all encrypted parts of the global weight, but it is possible that $I_k^{(\text{down})} \subsetneq [1, n_{pu}]$ if the learning participant has restricted download bandwidth.
2. Decrypt the above ciphertexts using the secret key sk to obtain $W_{\text{global}}^{(j)}$ for all $j \in I_k^{(\text{down})}$, and replace those values into the corresponding places of $W_{\text{global}} = (W_{\text{global}}^{(1)}, \ldots, W_{\text{global}}^{(n_{pu})})$.
3. Get a mini-batch of data from its local dataset.
4. Using the values of W_{global} and data items at steps **2** and **3**, compute the gradients $G = (G^{(1)}, \ldots, G^{(n_{pu})})$ with respect to variable W_{global}.
5. Encrypt and send back the ciphertexts $\mathbf{E}(-\alpha \cdot G^{(i)})$ $\forall i \in I_k^{(\text{up})} \subset [1, n_{pu}]$ to the corresponding processing unit PU_i of the server. The subset for uploading $I_k^{(\text{up})} \subset [1, n_{pu}] = \{1, \ldots, n_{pu}\}$ depends on the choice of participant k. For a full upload, $I_k^{(\text{up})} = [1, n_{pu}]$, so that all encrypted gradients are uploaded to the server.

The downloads and uploads of the encrypted parts of W_{global} can be *asynchronous* in two aspects: the participants are independent with each other; and the processing units are also independent with each other.

B. Cloud Server

The cloud server is a common place to recursively update the encrypted weight parameters. In particular, each processing unit PU_i at the server, after receiving any encryption $\mathbf{E}(-\alpha \cdot G^{(i)})$, computes

$$\mathbf{E}(W_{\text{global}}^{(i)}) + \mathbf{E}(-\alpha \cdot G^{(i)}) \left(\text{which is} = \mathbf{E}(W_{\text{global}}^{(i)} - \alpha \cdot G^{(i)})\right)$$

where the equality is ensured by the additively homomorphic property of encryption. Therefore, the part $W_{\text{global}}^{(i)}$ at the processing unit PU_i is updated to $W_{\text{global}}^{(i)} - \alpha \cdot G^{(i)}$, or notationally $W_{\text{global}}^{(i)} := W_{\text{global}}^{(i)} - \alpha \cdot G^{(i)}$.

Theorem 1 (Security Against the Cloud Server): Our system in Figure 5 leaks no information on the datasets to the honest-but-curious cloud server, provided that the underlying homomorphic encryption scheme is CPA-secure.

Proof: The participants only send encrypted gradients to the cloud server. Therefore, if the encryption scheme is CPA-secure, no bit of information on the data of the participants can be leaked. ∎

Theorem 2 (Accuracy Equivalence to Asynchronous SGD): Our system in Figure 5, functions as asynchronous SGD described in Section II-B when all ciphertexts are decrypted and $I_k^{(\text{up})} = I_k^{(\text{down})} = [1, n_{pu}]\ \forall 1 \le k \le N$ (meaning all gradients are uploaded and downloaded). Therefore, our system can achieve the same accuracy as that of asynchronous SGD.

Proof: After decryption, the update rule of weight parameter becomes $W_{\text{global}}^{(i)} := W_{\text{global}}^{(i)} - \alpha \cdot G^{(i)}$ for any $1 \le i \le n_{pu}$ in which $G^{(i)}$ is the gradient vector computed from data samples held by participant k (and the downloaded W_{global}). Since the update rule is identical to (6) and each learning participant in our system functions as a replica (as in asynchronous SGD) when encryption is removed, the theorem follows. ∎

Remark 1: As learning participants connect to the server via distinct TLS/SSL channels, the learning participants will be only able to retrieve the neural network weights but not the gradients, assuming that the server and any leaning participant do not collude. This helps to control the information sharing among learning participants, because deriving any information on data from the model becomes a model inversion problem.

V. Instantiations of Our System

In this section we use additively homomorphic encryption schemes to instantiate our system in Section IV. We use the following schemes to show two instantiations of our system: LWE-based encryption (modern, potentially post-quantum), and Paillier encryption (classical, smaller key sizes, not post-quantum).

A. Using an LWE-Based Encryption

The mark $\xleftarrow{\text{g}}$ is for "sampling randomly from a discrete Gaussian distribution", so that $x \xleftarrow{\text{g}} \mathbb{Z}_{(0,s)}$ means x appears with a probability proportional to $\exp(-\pi x^2/s^2)$.

1) LWE-Based Encryption: We use an additively homomorphic variant [9], [11] of the public key encryption scheme in [21]. The CPA security of this scheme is recalled in Appendix A.

- $\mathsf{ParamGen}(1^\lambda)$: Fix $q = q(\lambda) \in \mathbb{Z}^+$ and $l \in \mathbb{Z}^+$. Fix $p \in \mathbb{Z}^+$ so that $\gcd(p, q) = 1$. Return $pp = (q, l, p)$.
- $\mathsf{KeyGen}(1^\lambda, pp)$: Take $s = s(\lambda, pp) \in \mathbb{R}^+$ and $n_{lwe} \in \mathbb{Z}^+$. Take $R, S \xleftarrow{\text{g}} \mathbb{Z}_{(0,s)}^{n_{lwe} \times l}$, $A \xleftarrow{\$} \mathbb{Z}_q^{n_{lwe} \times n_{lwe}}$. Compute $P = pR - AS \in \mathbb{Z}_q^{n_{lwe} \times l}$. Return the public key $pk = (A, P, n_{lwe}, s)$, and the secret key $sk = S$.
- $\mathsf{Enc}(pk, m \in \mathbb{Z}_p^{1\times l})$: Take $e_1, e_2 \xleftarrow{\text{g}} \mathbb{Z}_{(0,s)}^{1\times n_{lwe}}$, $e_3 \xleftarrow{\text{g}} \mathbb{Z}_{(0,s)}^{1\times l}$. Compute $c_1 = e_1\ A + pe_2 \in \mathbb{Z}_q^{1\times n_{lwe}}$, $c_2 = e_1\ P + pe_3 + m \in \mathbb{Z}_q^{1\times l}$. Return $c = (c_1, c_2)$.
- $\mathsf{Dec}(S, c = (c_1, c_2))$: Compute $\overline{m} = c_1\ S + c_2 \in \mathbb{Z}_q^{1\times l}$. Return $m = \overline{m} \bmod p$.
- $\mathsf{Add}(c, c')$: For addition, compute and return $c_{\text{add}} = c + c' \in \mathbb{Z}_q^{1\times(n_{lwe}+l)}$.

2) Data Encoding and Encryption: A real number $a \in \mathbb{R}$ can be represented, with prec bits of precision, by an integer $\lfloor a \cdot 2^{\mathsf{prec}} \rfloor \in \mathbb{Z}$. Let us realize the encryption $\mathbf{E}(\cdot)$ in Figure 5. Because both $W_{\text{global}}^{(i)}$ and $\alpha \cdot G^{(i)}$ are in the space $\mathbb{R}^{L_i}$ where L_i is the length of the partition so that $\sum_{i=1}^{n_{pu}} L_i = n_{gd}$, it suffices to describe an encryption of a real vector $r = (r^{(1)}, \ldots, r^{(L_i)}) \in \mathbb{R}^{L_i}$. The encryption is, for $l = L_i$,

$$\mathbf{E}(r) = \mathsf{lweEnc}_{pk}\Big(\overbrace{\underbrace{\lfloor r^{(1)} \cdot 2^{\mathsf{prec}} \rfloor}_{\in \mathbb{Z}_p} \cdots \underbrace{\lfloor r^{(L_i)} \cdot 2^{\mathsf{prec}} \rfloor}_{\in \mathbb{Z}_p}}^{\mathbb{Z}_p^{1\times L_i}}\Big). \quad (10)$$

For vectors $r, t \in \mathbb{R}^{L_i}$, the decryption of

$$\mathbf{E}(r) + \mathbf{E}(-t) \in \mathbb{Z}_q^{1\times(n_{lwe}+L_i)} \quad (11)$$

will yield, for all $1 \le j \le L_i$,

$$\lfloor r^{(j)} \cdot 2^{\mathsf{prec}} \rfloor - \lfloor t^{(j)} \cdot 2^{\mathsf{prec}} \rfloor \in \mathbb{Z}_p \subset (-p/2, p/2] \quad (12)$$

and hence

$$u^{(j)} = \lfloor r^{(j)} \cdot 2^{\mathsf{prec}} \rfloor - \lfloor t^{(j)} \cdot 2^{\mathsf{prec}} \rfloor \in \mathbb{Z} \quad (13)$$

if $p/2$ is large enough (see below). The subtraction $r^{(j)} - t^{(j)} \in \mathbb{R}$ is computed via $u^{(j)}/2^{\mathsf{prec}} \in \mathbb{R}$, so that finally $r - t \in \mathbb{R}^L$ is obtained after decryption as desired. To get (13) from (12), it suffices that $p/2 > 2 \cdot 2^{\mathsf{prec}}$, as via normalisation, we can assume $-1 < r^{(j)}, t^{(j)} < 1$. In general, to handle n_{gradupd} additive terms without overflow, it is necessary that $p/2 > n_{\text{gradupd}} \cdot 2^{\mathsf{prec}}$, or equivalently,

$$p > n_{\text{gradupd}} \cdot 2^{\mathsf{prec}+1}. \quad (14)$$

Lemma 1 (Choosing Parameters): When $n_{lwe} \ge 3000$, $s = 8$, it is possible to set

$$\log_2 q \approx \log_2 p + \log_2 n_{\text{gradupd}} + \log_2(167.9\sqrt{n_{lwe}} + 33.9) + 1$$

in which p satisfies (14), and n_{gradupd} is the number of gradient updates at each processing unit of the cloud server in Figure 5. For example, when $n_{lwe} = 3000$, $p = 2^{48} + 1$, $n_{\text{gradupd}} = 2^{15}$, it is possible to set $q = 2^{77}$.

It is worth noting that the parameters in Lemma 1 is very conservative, due to the consideration of unlikely large noise in the proof. Therefore, n_{gradupd} can be larger than stated. Indeed, we experimentally check with $q = 2^{77}$, $n_{lwe} = 3000$, $p = 2^{48} + 1$, $s = 8$ and confirm that n_{gradupd} can be twice (i.e. $n_{\text{gradupd}} = 2 \cdot 2^{15}$) of that stated in Lemma 1 without any decryption error.

Theorem 3 (Increased Communication Factor, LWE-Based): The communication between the server and participants of our system is

$$\frac{n_{pu} n_{lwe} \log_2 q}{n_{gd} \cdot \mathsf{prec}} + \frac{\log_2 q}{\mathsf{prec}}$$

time of the communication of the corresponding asynchronous SGD, in which (n_{lwe}, p, q) is parameters of the encryption scheme, and n_{gd} is the number of gradient variables represented by prec bits.

Proof: In asynchronous SGD (Section II-B), each replica sends $n_{gd} = \sum_{i=1}^{n_{pu}} L_i$ gradients (each of prec bits) to the parameter server at each iteration, so that the communication cost for one iteration in bits is

$$\mathsf{PlainBits} = n_{gd} \cdot \mathsf{prec}.$$

In our system, we compute the ciphertext length that each participant sends to the cloud parameter server at each iteration. By (10), the ciphertext sent to processing unit PU_i is in $\mathbb{Z}_q^{1 \times (n_{lwe} + L_i)}$ so that its length in bits is $(n_{lwe} + L_i) \log_2 q$. The length in bits of all ciphertexts sent to the parameter server from the corresponding learning participant at each interaction is at most

$$\begin{aligned} \mathsf{EncryptedBits} &= \sum_{i=1}^{n_{pu}} (n_{lwe} + L_i) \log_2 q \\ &= n_{pu} n_{lwe} \log_2 q + n_{gd} \log_2 q. \end{aligned}$$

Therefore, the increased factor is

$$\frac{\mathsf{EncryptedBits}}{\mathsf{PlainBits}} = \frac{n_{pu} n_{lwe} \log_2 q}{n_{gd} \cdot \mathsf{prec}} + \frac{\log_2 q}{\mathsf{prec}}$$

ending the proof. ∎

B. Using Paillier Encryption

1) Paillier Encryption: With public key $pk = n$ (a large positive integer), the encryption of an integer $m \in \{0, \ldots, n-1\}$ is $\mathsf{PaiEnc}_{pk}(m) = r^n (1+n)^m \mod n^2$ in which r is chosen randomly from $\{0, \ldots, n-1\}$. The encryption is additively homomorphic because the ciphertext product $\mathsf{PaiEnc}_{pk}(m_1)\mathsf{PaiEnc}_{pk}(m_2) \mod n^2$ becomes an encryption of $m_1 + m_2 \mod n$. For decryption and CPA security, see the paper [25].

2) Packing Data in Encryption: As the plaintext space has $\log_2 n \geq 2048$ bits, we can pack many non-negative integers $I_1, \ldots, I_t$ each of prec bits into each Paillier plaintext as follows.

$$\mathsf{PaiEnc}_{pk}\Big(\overbrace{\underbrace{[I_1 0_{\mathsf{pad}}]}_{\mathsf{prec}+\mathsf{pad}\text{ bits}} \cdots \underbrace{[I_t 0_{\mathsf{pad}}]}_{\mathsf{prec}+\mathsf{pad}\text{ bits}}}^{\lfloor \log_2 n \rfloor \text{ bits}} \Big)$$

in which 0_{pad} is the zero padding of pad bits, which helps to prevent overflows in ciphertext additions. Typically, $\mathsf{pad} \approx \log_2 n_{\text{gradupd}}$ as we need n_{gradupd} additions of ciphertexts. Moreover, as the number of plaintext bits must be less than $\log_2 n$, it is necessary that $t(\mathsf{prec}+\mathsf{pad}) \leq \log_2 n$. Therefore,

$$t = \left\lfloor \frac{\lfloor \log_2 n \rfloor}{\mathsf{prec} + \mathsf{pad}} \right\rfloor$$

which is the upper-bound of packing prec-bit integers into one Paillier plaintext. To handle both negative and positive integers, we can use the bijection $z \in [0, 2^{\mathsf{prec}} - 1] \mapsto z - \lfloor z / 2^{\mathsf{prec}} \rceil \cdot 2^{\mathsf{prec}}$.

As a real number $0 \leq r < 1$ can be represented as an integer of form $\lfloor r \cdot 2^{\mathsf{prec}} \rfloor$, the above packing method can be used to encrypt around $\lfloor \log_2 n \rfloor / (\mathsf{prec} + \mathsf{pad})$ real numbers in the range $[0, 1)$ with precision prec, tolerating around 2^{pad} ciphertext additions. To realise the encryption $\mathbf{E}(\cdot)$ in Figure 5, it suffices to describe an encryption of a real vector $r = (r^{(1)}, \ldots, r^{(L_i)}) \in \mathbb{R}^{L_i}$ because both $W_{\text{global}}^{(i)}$ and $\alpha \cdot G^{(i)} \in \mathbb{R}^{L_i}$ in which $\sum_{i=1}^{n_{pu}} L_i = n_{gd}$. The encryption $\mathbf{E}(r)$ consists of approximately $\lceil L_i / t \rceil$ Paillier ciphertexts:

$$\mathsf{PaiEnc}_{pk}\Big(\overbrace{\underbrace{\lfloor r^{(1)} \cdot 2^{\mathsf{prec}} \rfloor 0_{\mathsf{pad}}}_{\mathsf{prec}+\mathsf{pad}\text{ bits}} \cdots \underbrace{\lfloor r^{(t)} \cdot 2^{\mathsf{prec}} \rfloor 0_{\mathsf{pad}}}_{\mathsf{prec}+\mathsf{pad}\text{ bits}}}^{\lfloor \log_2 n \rfloor \text{ bits}} \Big), \ldots,$$

$$\mathsf{PaiEnc}_{pk}\Big(\overbrace{\underbrace{\lfloor r^{(L_i - t + 1)} \cdot 2^{\mathsf{prec}} \rfloor 0_{\mathsf{pad}}}_{\mathsf{prec}+\mathsf{pad}\text{ bits}} \cdots \underbrace{\lfloor r^{(L_i)} \cdot 2^{\mathsf{prec}} \rfloor 0_{\mathsf{pad}}}_{\mathsf{prec}+\mathsf{pad}\text{ bits}}}^{\lfloor \log_2 n \rfloor \text{ bits}} \Big).$$

Theorem 4 (Increased Communication Factor, Paillier-Based): The communication between the server and participants of our system is

$$2\left(1 + \frac{\mathsf{pad}}{\mathsf{prec}}\right)$$

times of the communication of the corresponding asynchronous SGD, in which pad is the number of 0 paddings to tolerate 2^{pad} additions (equal to the number of gradient updates on the server), and prec is the bit precision of numbers.

Proof: In our system in Figure 5, each participant needs to encrypt and send at most $\sum_{i=1}^{n_{pu}} L_i = n_{gd}$ gradients to the server, in which L_i is the length of $G^{(i)}$. Therefore, with the above packing method for Paillier encryption, the number of Paillier ciphertexts sent from each participant is around

$$\sum_{i=1}^{n_{pu}} \frac{L_i}{t} \approx \frac{n_{gd}}{t} \approx \frac{n_{gd}(\mathsf{prec} + \mathsf{pad})}{\lfloor \log_2 n \rfloor}.$$

Since each Paillier ciphertext is of $2\lfloor \log_2 n \rfloor$ bits, the number of bits for each communication is around

$$\mathsf{EncryptedBits} \approx 2\lfloor \log_2 n \rfloor \cdot \frac{n_{gd}}{t} \approx 2 n_{gd}(\mathsf{prec} + \mathsf{pad}).$$

TABLE III

LWE-BASED ENCRYPTION SCHEME ($n_{lwe} = 3000$, $s = 8$, $p = 2^{48} + 1$, AND $q = 2^{77}$) RUNNING TIMES WITH VARIOUS NUMBERS OF GRADIENTS

		Number of gradients n_{gd}							
		20000	27882	50000	52650	100000	200000	300000	402250
		Processing time using 1 thread of computation							
Enc. via (10)	msec	164.4	213.1	364.6	454.8	732.9	1395.1	2031.6	2797.4
Dec. of (10)	msec	143.6	194.5	354.6	412.6	666.1	1355.0	1996.5	2738.0
Add of ciphertexts via (11)	usec	51	72.3	137.7	91.2	211.5	441.7	481.0	698.4
		Processing time using 20 threads of computation							
Enc. via (10)	msec	30.2	50.8	93.2	70.3	188.9	415.8	587.8	875.4
Dec. of (10)	msec	31.9	44.7	85.8	78.5	196.0	460.5	644.9	820.9
Add of ciphertexts via (11)	usec	9.5	11.1	15.8	14.0	19.5	38.7	41.1	56.4

On the other hand, note that each replica in asynchronous SGD needs to send n_{gd} gradients each of prec bits to the server, the communication cost in bits is $\mathsf{PlainBits} = n_{gd} \cdot \mathsf{prec}$. Therefore, the increased factor is

$$\frac{\mathsf{EncryptedBits}}{\mathsf{PlainBits}} = \frac{2n_{gd}(\mathsf{prec} + \mathsf{pad})}{n_{gd} \cdot \mathsf{prec}} = 2\left(1 + \frac{\mathsf{pad}}{\mathsf{prec}}\right)$$

as claimed. ∎

For the case of Paillier encryption, we can take $\mathsf{pad} = 15$, so that the increased communication factor becomes, via Theorem 4,

$$\frac{\mathsf{EncryptedBits}}{\mathsf{PlainBits}} = 2\left(1 + \frac{\mathsf{pad}}{\mathsf{prec}}\right) = 2\left(1 + \frac{15}{32}\right) \approx 2.93$$

which is independent of n_{gd}. This increased communication factor is a little larger than that of the LWE-based one when n_{gd} is big as evaluated below.

VI. CONCRETE EVALUATIONS WITH AN LWE-BASED ENCRYPTION

We take $n_{lwe} = 3000$, $s = 8$, $p = 2^{48}+1$, $n_{\text{gradupd}} = 2^{15}$, and $q = 2^{77}$ following Lemma 1. These parameters for (n_{lwe}, s, q) conservatively ensure that the LWE assumption has at least 128-bit security according to recent attacks [7], [15], [21], [22]. We will take $n_{pu} \in \{1, 10\}$, depending on the number of gradients.

A. Increased Factors in Communication

Let us consider multiple number of gradient parameters n_{gd}:

- $n_{gd} = 109386$: this number of gradient parameters is used with the dataset MNIST [5]. Specifically, consider a multilayer perceptron with form 784 (input) – 128 (hidden) – 64 (hidden) – 10 (output). The number of gradients for this network is $(784+1)128+(128+1)64+(64+1)10 = 109386$. We will consider cases where real numbers are represented by 32 bits, so that $\mathsf{prec} = 32$. Theorem 3 tells us that the increased communication factor between our system and the related asynchronous SGD is
$$\frac{n_{pu}n_{lwe}\log_2 q}{n_{gd} \cdot \mathsf{prec}} + \frac{\log_2 q}{\mathsf{prec}} = \frac{1 \cdot 3000 \cdot 77}{109386 \cdot 32} + \frac{77}{32} \approx 2.47.$$
- $n_{gd} = 402250$: this is used in [28] with the dataset SVHN [24]. The increased communication factor becomes
$$\frac{n_{pu}n_{lwe}\log_2 q}{n_{gd} \cdot \mathsf{prec}} + \frac{\log_2 q}{\mathsf{prec}} = \frac{1 \cdot 3000 \cdot 77}{402250 \cdot 32} + \frac{77}{32} \approx 2.42.$$
- $n_{gd} = 42 \cdot 10^6$: this number of gradient parameters is used in [16] for speech data. As the number of gradients is large, consider $n_{pu} = 10$. The increased communication factor becomes
$$\frac{n_{pu}n_{lwe}\log_2 q}{n_{gd} \cdot \mathsf{prec}} + \frac{\log_2 q}{\mathsf{prec}} = \frac{10 \cdot 3000 \cdot 77}{42 \cdot 10^6 \cdot 32} + \frac{77}{32} \approx 2.4.$$

B. Estimating the Computational Costs

To estimate the running time of our system, we use the following formulas

$$\begin{aligned}\mathbf{T}^{(i)}_{\text{ours, one run}} = \mathbf{T}^{(i)}_{\text{original, one run}} + \mathbf{T}^{(i)}_{\text{enc}} + \mathbf{T}^{(i)}_{\text{dec}} \\ + \mathbf{T}^{(i)}_{\text{upload}} + \mathbf{T}^{(i)}_{\text{download}} + \mathbf{T}_{\text{add}},\end{aligned} \tag{15}$$

$$\mathbf{T}_{\text{our system}} = \sum_{i=1}^{N} n^{(i)}_{\text{upload/download}} \times \mathbf{T}^{(i)}_{\text{ours,one run}} \tag{16}$$

where, in (15), $\mathbf{T}^{(i)}_{\text{ours, one run}}$ is the running time of the participant i when doing the following: waiting for the addition of ciphertexts at the server ($\mathbf{T}_{\text{add}}$); downloading the added ciphertext from the server ($\mathbf{T}^{(i)}_{\text{download}}$); decrypting the downloaded ciphertext ($\mathbf{T}^{(i)}_{\text{dec}}$); training with the downloaded weight ($\mathbf{T}^{(i)}_{\text{original, one run}}$); encrypting the resulting gradients ($\mathbf{T}^{(i)}_{\text{enc}}$) and sending back that ciphertext to the server ($\mathbf{T}^{(i)}_{\text{upload}}$). The total running time of our system $\mathbf{T}_{\text{our system}}$ will be the sum of all N participants' running times, multiplied by the number n_{repeat} of repetitions, as expressed in (16).

1) Environment: Our codes for homomorphic encryption are in C++, and the benchmarks are over an Xeon CPU E5-2660 v3@ 2.60GHz server. To estimate the communication speed between each learning participants and the server, we assume a 1 Gbps network. To measure the running time of MLP, we use the Tensorflow 1.1.0 library [4] over Cuda-8.0 and GPU Tesla K40m.

2) Time of Encryption, Decryption, and Addition: Table III gives the time for encryption, decryption and addition depending on the number of gradients n_{gd}, using $n_{lwe} = 3000$, $s = 8$, $p = 2^{48} + 1$, and $q = 2^{77}$. Figure 6 depicts the time of

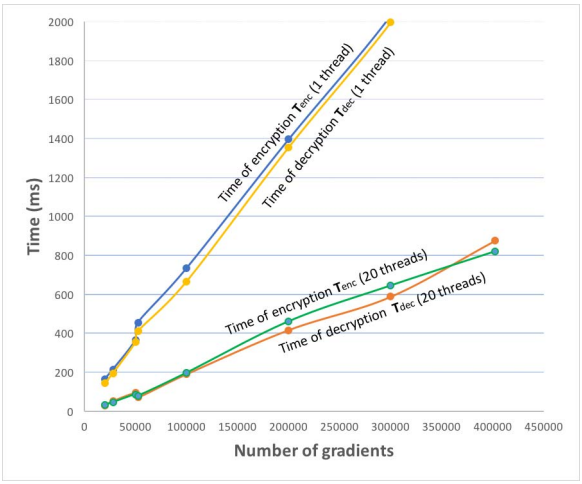


Fig. 6. Computational costs of the LWE-based encryption and decryption when $n_{lwe} = 3000, s = 8, q = 2^{77}$.

encryption and decryption using 1 thread and 20 threads of computation respectively.

3) Multilayer Perceptron (MLP): Consider an MLP with form 784 (input) – 128 (hidden) – 64 (hidden) – 10 (output). The number of gradients for this network is $(784+1)128+(128+1)64+(64+1)10 = 109386$. For 32-bit precision, these gradients are around $109386 \times 32/(8 \times 10^6) \approx 0.437$ MB in plain. The ciphertext of these gradients is of $0.437 \times 2.47 \approx 1.0$ MB as computed in Section VI-A which can be sent in around 0.008 seconds (= 8 ms) via an 1 Gbps communication channel. The original running time of this MLP with one data batch of size 50 (MNIST images) is $\mathbf{T}^{(i)}_{\text{MLP, one run}} \approx 4.6$ (ms). Therefore,

$$\begin{aligned}\mathbf{T}^{(i)}_{\text{ours, one run}} &= \mathbf{T}^{(i)}_{\text{MLP, one run}} + \mathbf{T}^{(i)}_{\text{enc}} + \mathbf{T}^{(i)}_{\text{dec}} \\ &\quad + \mathbf{T}^{(i)}_{\text{upload}} + \mathbf{T}^{(i)}_{\text{download}} + \mathbf{T}_{\text{add}} \\ &\approx 4.6 + 188.9 + 196.0 + 8 + 8 + 19.5/10^3 \\ &\approx 405.5 \text{ (ms)}. \end{aligned} \tag{17}$$

Suppose that there are totally 2×10^4 times of upload and download from all training participants to the server, namely $\sum_{i=1}^{N} n^{(i)}_{\text{upload/download}} = 2 \times 10^4$. For simplicity suppose $\mathbf{T}^{(i)}_{\text{ours, one run}}$ is the same for all $1 \le i \le N$. In such case, the running time of our system can be estimated as

$$\begin{aligned}\mathbf{T}_{\text{our system}} &= \mathbf{T}^{(i)}_{\text{ours, one run}} \times 2 \times 10^4 \\ &= 405.5 \times 2 \times 10^4 \text{ (ms)} \approx 2.25 \text{ (hours)}.\end{aligned}$$

As seen in Theorem 2, the accuracy of our system can be identical to that of asynchronous SGD. Therefore, it suffices to estimate the accuracy of asynchronous SGD over MNIST containing randomly shuffled 6×10^4 images for training and 10^4 images for testing. As above, the batch size is 50. The initial weights are randomly selected from a normal distribution of mean 0 and standard deviation 0.1 (via the function `random_normal` of Tensorflow). The activation function is the `relu` function in Tensorflow. The Adam optimizer (the `AdamOptimizer` in Tensorflow) is used for training with input learning rate 10^{-4}, with no drop out. After 2×10^4 iterations elapsed by 2 minutes, our Tensorflow code achieves around 97% accuracy over the testing set.

4) Convolutional Neural Networks (CNN): Thanks to Theorem 2, our system in Section IV has the same accuracy as asynchronous SGD, so that we can re-use the accuracy results in [28]. In particular, let us consider the CNN [28, Fig. 6] for MNIST with the number of gradients $n_{gd} = 105506$. The corresponding accuracy reported in [28, Table 4, Fig. 8] is around 99% when 10% or more (100%) of the gradients are sent to the server. For the discussion of efficiency, let the number of training epoch be 35; the mini-batch size be 32; and the number of learning participant $N = 30$ (all used in [28]). Note that the number of gradients n_{gd} in this case is almost the same as that in the MLP case, so the speed of encryption and decryption and communication will be identical (or a little lesser). Therefore, we can safely re-use (17) so that

$$\begin{aligned}\mathbf{T}^{(i)}_{\text{ours, one run}} &= \mathbf{T}^{(i)}_{\text{CNN, one run}} + \mathbf{T}^{(i)}_{\text{enc}} + \mathbf{T}^{(i)}_{\text{dec}} \\ &\quad + \mathbf{T}^{(i)}_{\text{upload}} + \mathbf{T}^{(i)}_{\text{download}} + \mathbf{T}_{\text{add}} \\ &\approx \mathbf{T}^{(i)}_{\text{CNN, one run}} + 188.9 + 196.0 \\ &\quad + 8 + 8 + 19.5/10^3 \\ &\approx \mathbf{T}^{(i)}_{\text{CNN, one run}} + 401 \text{ (ms)}\end{aligned}$$

Since the number of images in the training set is 60000 shared between $N = 30$ leaning participants, each participant will hold $60000/30 = 2000$ images. The number of mini-batch is 32, so that there will be about $2000/32$ uploads and downloads from each participant in each epoch. Therefore there will be $(2000/32) \times 30 \times 35$ uploads and downloads with 30 participants in 35 epochs. The estimated running time of our system will be

$$\begin{aligned}\mathbf{T}_{\text{our system}} &= \mathbf{T}^{(i)}_{\text{ours, one run}} \times (2000/32) \times 30 \times 35 \\ &\approx \left(\mathbf{T}^{(i)}_{\text{CNN, one run}} + 401\right) 65625 \text{ (ms)} \\ &\approx \left(\mathbf{T}^{(i)}_{\text{CNN, one run}} + 401\right) \frac{65625}{36 \times 10^5} \text{(hours)} \\ &\approx 0.018\mathbf{T}^{(i)}_{\text{CNN, one run}} + 7.3 \text{ (hours)}\end{aligned}$$

which is around 3.3 times that of the MLP case.

VII. Conclusion

In this paper, we have shown that sharing gradients even partially over a parameter cloud server as in [28] may leak information. We then proposed a new system that utilizes additively homomorphic encryption to protect the gradients against the curious server. In addition to privacy preservation, our system enjoys the property of not declining the accuracy of deep learning.

Appendix A
CPA Security of the LWE-Based Homomorphic Encryption Scheme

We first recall the definition of CPA security.

Definition 2 (CPA Security): With respect to a PHE scheme as in Definition 1, consider the following game between an adversary $\mathcal{A}$ and a challenger:

- **Setup.** The challenger creates pp and key pairs (pk, sk). Then pp and pk are given to $\mathcal{A}$.
- **Challenge.** $\mathcal{A}$ chooses two plaintexts m_0, m_1 of the same length, then submits them to the challenger, who in turn takes $b \in \{0,1\}$ randomly and computes $C^* = \mathsf{Enc}(pk, m_b)$. The challenge ciphertext C^* is returned to $\mathcal{A}$, who then produces a bit b'.

A PHE scheme is secure against chosen plaintext attacks (CPA-secure) if the advantage

$$\mathbf{Adv}_{\mathcal{A}}^{\text{cpa}}(\lambda) = \left|\Pr[b' = b] - \frac{1}{2}\right|$$

is negligible in λ.

Related to the decision LWE assumption LWE(n_{lwe}, s, q), where n_{lwe}, s, q is the parameter for security, consider matrix $A \xleftarrow{\$} \mathbb{Z}_q^{m \times n_{lwe}}$, vectors $r \xleftarrow{\$} \mathbb{Z}_q^{m \times 1}$, $x \xleftarrow{\text{g}} \mathbb{Z}_{(0,s)}^{n_{lwe} \times 1}$, $e \xleftarrow{\text{g}} \mathbb{Z}_{(0,s)}^{m \times 1}$. Then vector $Ax + e$ is computed over $\mathbb{Z}_q$. Define the following advantage of a poly-time probabilistic algorithm $\mathcal{D}$:

$$\mathbf{Adv}_{\mathcal{D}}^{\text{LWE}(n_{lwe},s,q)}(\lambda) = \Big|\Pr[\mathcal{D}(A, Ax + e) \to 1] - \Pr[\mathcal{D}(A, r) \to 1]\Big|.$$

The LWE assumption asserts that $\mathbf{Adv}_{\mathcal{D}}^{\text{LWE}(n_{lwe},s,q)}(\lambda)$ is negligible as a function of λ.

Claim 1: The additively homomorphic encryption scheme in Section V-A is CPA-secure under the LWE assumption. Specifically, for any poly-time adversary $\mathcal{A}$, there is an algorithm $\mathcal{D}$ of essentially the same running time such that

$$\mathbf{Adv}_{\mathcal{A}}^{\text{cpa}}(\lambda) \le (l+1) \cdot \mathbf{Adv}_{\mathcal{D}}^{\text{LWE}(n_{lwe},s,q)}(\lambda).$$

Proof: The proof has been in [9] and is given here for completeness. First, we modify the original game in Definition 2 by providing $\mathcal{A}$ with a random $P \xleftarrow{\$} \mathbb{Z}_q^{n_{lwe} \times l}$. Namely $P = pR - AS \in \mathbb{Z}_q^{n_{lwe} \times l}$ is turned to random. This is indistinguishable to $\mathcal{A}$ thanks to the LWE assumption with secret vectors as l columns of S. More precisely, we need the condition $\gcd(p, q) = 1$ to reduce $P = pR - AS \in \mathbb{Z}_q^{n_{lwe} \times l}$ to the LWE form. Indeed, $p^{-1}P = R + (-p^{-1}A)S \in \mathbb{Z}_q^{n_{lwe} \times l}$. As A is random, $A' = -p^{-1}A \in \mathbb{Z}_q^{n_{lwe} \times n_{lwe}}$ is also random. Therefore, $P' = p^{-1}P \in \mathbb{Z}_q^{n_{lwe} \times l}$ is random under the LWE assumption which in turn means P is random as claimed.

Second, the challenge ciphertext $c^* = e_1[A|P] + p[e_2|e_3] + [0|m_b]$ is turned to random. This relies on the LWE assumption with secret vector e_1, while also basing on the first modification so that $[A|P]$ is a random matrix. Here, the condition $\gcd(p, q) = 1$ is also necessary as above. Thus b is perfectly hidden after this change. The factor $l + 1$ is due to l uses of LWE in changing P and 1 use in changing c^*. ∎

Appendix B
Proof of Lemma 1

We will use the following lemmas [12], [13], [21]. Below $\langle \cdot, \cdot \rangle$ stands for inner product. Writing $\|\mathbb{Z}_{(0,s)}^n\|$ is a short hand for taking a vector from the discrete Gaussian distribution of deviation s and computing its Euclidean norm.

Lemma 2: Let $c \ge 1$ and $C = c \cdot \exp(\frac{1-c^2}{2})$. Then for any real $s > 0$ and any integer $n \ge 1$, we have

$$\Pr\left[\|\mathbb{Z}_{(0,s)}^n\| \ge \frac{c \cdot s\sqrt{n}}{\sqrt{2\pi}}\right] \le C^n.$$

Lemma 3: For any real $s > 0$ and $T > 0$, and any $x \in \mathbb{R}^n$, we have

$$\Pr\left[\|\langle x, \mathbb{Z}_{(0,s)}^n \rangle\| \ge Ts\|x\|\right] < 2\exp(-\pi T^2).$$

Proof: [Proof of Claim 1] Suppose $c_{\text{add}} = \sum_{i=1}^{n_{\text{add}}} c^{(i)} = \sum_{i=1}^{n_{\text{add}}} (c_1^{(i)}, c_2^{(i)})$ is the additions of n_{add} ciphertexts. In Claim 1, $n_{\text{add}} = n_{\text{gradupd}}$. Its decryption is

$$\begin{aligned}
\overline{m}_{\text{add}} &= \sum_{i=1}^{n_{\text{add}}} \left[c_1^{(i)} S + c_2^{(i)}\right] \\
&= \sum_{i=1}^{n_{\text{add}}} \left[(e_1^{(i)} A + p e_2^{(i)})S + e_1^{(i)} P + p e_3^{(i)} + m^{(i)}\right] \\
&= \sum_{i=1}^{n_{\text{add}}} \left[e_1^{(i)}(AS + P) + p(e_2^{(i)} S + e_3^{(i)}) + m^{(i)}\right] \\
&= \sum_{i=1}^{n_{\text{add}}} \left[p(e_1^{(i)} R + e_2^{(i)} S + e_3^{(i)}) + m^{(i)}\right] \in \mathbb{Z}_q^{1 \times l}
\end{aligned}$$

Therefore, the accumulative noise after n_{add} ciphertext additions is

$$\mathsf{Noise} = p \sum_{i=1}^{n_{\text{add}}} (e_1^{(i)} R + e_2^{(i)} S + e_3^{(i)}) \in \mathbb{Z}_q^{1 \times l}$$

The term $e_1^{(i)} R + e_2^{(i)} S + e_3^{(i)} \in \mathbb{Z}_q^{1 \times l}$ has l components, each of which can be written as the inner product of two vectors of form

$$\begin{aligned}
e &= (e_1^{(i)}, e_2^{(i)}, e_3^{(i)}) \\
x &= (r, \quad r', \mathbf{0}1\mathbf{0}_{1 \times l})
\end{aligned}$$

where,

- vectors $e_1^{(i)} \xleftarrow{\text{g}} \mathbb{Z}_{(0,s)}^{1 \times n_{lwe}}$, $e_2^{(i)} \xleftarrow{\text{g}} \mathbb{Z}_{(0,s)}^{1 \times n_{lwe}}$, and $e_3^{(i)} \xleftarrow{\text{g}} \mathbb{Z}_{(0,s)}^{1 \times l}$
- vectors $r, r' \xleftarrow{\text{g}} \mathbb{Z}_{(0,s)}^{1 \times n_{lwe}}$ represent corresponding columns in matrices R, S; and $\mathbf{0}1\mathbf{0}_{1 \times l}$ stands for a vector of length l with all 0's except one 1.

We have

$$\begin{aligned}
e &\in \mathbb{Z}_{(0,s)}^{1 \times (2n_{lwe} + l)} \\
\|x\| &\le \|(r, r')\| + 1.
\end{aligned}$$

Applying Lemma 2 for the above vector (r, r') of length $2n_{lwe}$, with overwhelming probability of

$$1 - C^{2n_{lwe}} (\ge 1 - 2^{-80} \text{ for our choices of parameters})$$

we have

$$\|x\| \le \frac{c \cdot s\sqrt{2n_{lwe}}}{\sqrt{2\pi}} + 1.$$

We now use Lemma 3 with vectors x and e. Let ρ be the error per message symbol in decryption, we set $2\exp(-\pi T^2) = \rho$,

so $T = \sqrt{\ln(2/\rho)}/\sqrt{\pi}$. The bound on the inner product $\langle x, e \rangle$ becomes $Ts\|x\|$, which is not greater than

$$U = \frac{s\sqrt{\ln(2/\rho)}}{\sqrt{\pi}} \left(\frac{c \cdot s\sqrt{2n_{lwe}}}{\sqrt{2\pi}} + 1 \right)$$

so that each component in $\mathsf{Noise} \in \mathbb{Z}_q^{1\times l}$ will be less than

$$B = pn_{\text{add}} \cdot U.$$

For correctness, it suffices to set $B = q/2$, namely $q = 2B$. This means

$$\log_2 \ q = \log_2 \ p + \log_2 n_{\text{add}} + \log_2(U) + 1.$$

We take $c = 1.1$ so that $C^{2n_{lwe}} < 2^{-80}$ when $n_{lwe} \geq 3000$. Also take $\rho = 2^{-80}, s = 8$ so that

$$U = 167.9106\sqrt{n_{lwe}} + 33.8197$$

and the proof follows. ■

ACKNOWLEGMENT

We are grateful to the anonymous reviewers for their comprehensive comments. This is the full version of a conference paper [26], with improvements described in Section I-E.

REFERENCES

[1] *Machine Learning Coursera's Course*. Accessed: Dec. 2, 2017. [Online]. Available: https://www.coursera.org/learn/machine-learning
[2] *MNIST Results*. Accessed: Dec. 2, 2017. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[3] *Stanford Deep Learning Tutorial*. Accessed: Dec. 2, 2017. [Online]. Available: http://deeplearning.stanford.edu
[4] *Tensorflow*. Accessed: Dec. 2, 2017. [Online]. Available: https://www.tensorflow.org
[5] *The MNIST Dataset*. Accessed: Dec. 2, 2017. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[6] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.
[7] Y. Aono, X. Boyen, L. T. Phong, and L. Wang, "Key-private proxy Re-encryption under LWE," in *Proc. Prog. Cryptol.-INDOCRYPT*, vol. 8250. 2013, pp. 1–18.
[8] Y. Aono, T. Hayashi, L. T. Phong, and L. Wang, "Privacy-preserving logistic regression with distributed data sources via homomorphic encryption," *IEICE Trans. Inf. Syst.*, vol. 99-D, no. 8, pp. 2079–2089, 2016.
[9] Y. Aono, T. Hayashi, L. T. Phong, and L. Wang, "Efficient key-rotatable and security-updatable homomorphic encryption," in *Proc. 5th ACM Int. Workshop Secur. Cloud Comput. (SCCAsiaCCS)*, 2017, pp. 35–42.
[10] Y. Aono, T. Hayashi, L. T. Phong, and L. Wang, "Input and output privacy-preserving linear regression," *IEICE Trans. Inf. Syst.*, vol. 100-D, no. 10, pp. 2339–2347, 2017.
[11] Y. Aono, T. Hayashi, L. T. Phong, and L. Wang, "Efficient homomorphic encryption with key rotation and security update," *IEICE Trans. Inf. Syst.*, vol. E101-A, no. 1, pp. 39–50, 2018.
[12] W. Banaszczyk, "New bounds in some transference theorems in the geometry of numbers," *Math. Annalen*, vol. 296, no. 1, pp. 625–635, 1993.
[13] W. Banaszczyk, "Inequalities for convex bodies and polar reciprocal lattices in R^n," *Discrete Comput. Geometry*, vol. 13, no. 1, pp. 217–231, 1995.
[14] K. Bonawitz *et al.*, "Practical secure aggregation for privacy preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2017, pp. 1175–1191.
[15] I. Chillotti, N. Gama, M. Georgieva, and M. E. Izabachène "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Proc. Adv. Cryptol.-ASIACRYPT*, 2016, pp. 3–33.
[16] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. 26th Annu. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1232–1240.
[17] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Feb. 2011.
[18] R. Gilad-Bachrach *et al.*, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. 33nd Int. Conf. Mach. Learn. (ICML)*, vol. 48. 2016, pp. 201–210.
[19] O. Goldreich, *Foundations of Cryptography: Basic Applications*. vol. 2. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[20] B. Hitaj, G. Ateniese, and F. Pérez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2017, pp. 603–618.
[21] R. Lindner and C. Peikert, "Better key sizes (and attacks) for LWE-based encryption," in *Topics in Cryptology-CT-RSA* (Lecture Notes in Computer Science), vol. 6558, A. Kiayias, Ed. Springer, 2011, pp. 319–339.
[22] M. Liu and P. Q. Nguyen, "Solving BDD by enumeration: An update," in *Topics in Cryptology-CT-RSA* (Lecture Notes in Computer Science), vol. 7779, E. Dawson, Ed. Springer, 2013, pp. 293–309.
[23] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy*, May 2017, pp. 19–38.
[24] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn*, 2011, pp. 1–9.
[25] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 1592, J. Stern, Ed. Springer, 1999, pp. 223–238.
[26] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning: Revisited and enhanced," in *Proc. 8th Int. Conf. Appl. Techn. Inf. Secur. (ATIS)*, 2017, pp. 100–110.
[27] B. Recht, C. Ré, S. J. Wright, and F. Niu,"Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, Eds. ACM, 2011, pp. 693–701.
[28] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1310–1321.

Le Trieu Phong received the B.S. degree from the University of Natural Sciences, Ho Chi Minh City, Vietnam, in 2002, and the Ph.D. degree from the Tokyo Institute of Technology in 2009. He is currently a Senior Researcher with the Cybersecurity Research Institute, National Institute of Information and Communications Technology, Japan. His current research interests are applied cryptography and privacy-preserving data mining.

Yoshinori Aono received the B.S. degree in engineering from the Musashi Institute of Technology in 2005, and the M.S. and Ph.D. degrees in mathematical and computing sciences from the Tokyo Institute of Technology, in 2007 and 2010, respectively. He is currently a Senior Researcher with the Cybersecurity Research Institute, National Institute of Information and Communications Technology, Japan, conducting researches on security analysis of cryptography. He was a recipient of the Encouragement award in GPU Challenge 2009 from IPSJ, the Young researcher's award 2011 from IEICE, and the SCIS 2013 paper award from IEICE.

Takuya Hayashi received the bachelor's degree in media architecture and the master's degree in systems information science from Future University Hakodate, in 2008 and 2010, respectively, and the Ph.D. degree in functional mathematics from Kyushu University in 2013. He is currently an Assistant Professor with Kobe University, and an Invited Advisor with the National Institute of Information and Communications Technology, Japan. His current research interests are in cryptanalysis and efficient implementation of public key cryptosystems. He was a recipient of the SCIS paper prize from IEICE in 2010 and the DOCOMO Mobile Science Award in 2013.

Lihua Wang received the B.S. degree in mathematics from Northeast Normal University, China, in 1988, the M.S. degree in mathematics from the Harbin Institute of Technology, China, in 1994, and the Ph.D. degree in engineering from the University of Tsukuba, Japan, in 2006. She is currently a Senior Researcher with the Cybersecurity Research Institute, National Institute of Information and Communications Technology, Japan. Her research interests include cryptography and information security.

Shiho Moriai received the B.E. degree from Kyoto University in 1993 and the Ph.D. degree from the University of Tokyo in 2003. She is currently the Director of Security Fundamentals Laboratory, Cybersecurity Research Institute, National Institute of Information and Communications Technology, Japan. She was a recipient of the IPSJ Industrial Achievement Award in 2006, the Minister's Award of The Ministry of Economy, Trade, and Industry, the Industrial Standardization Awards in 2011, and the Commendation for Science and Technology from the Minister of Education, Culture, Sports, Science, and Technology in 2014.