Lightweight Crypto-Assisted Distributed Differential Privacy for Privacy-Preserving Distributed Learning

Lingjuan Lyu
The Department of Computer Science
National University of Singapore
Singapore
lyulj@comp.nus.edu.sg

***Abstract*—The appearance of distributed learning allows multiple participants to collaboratively train a global model, where instead of directly releasing their private training data with the server, participants iteratively share their local model updates (parameters) with the server. However, recent attacks demonstrate that sharing local model updates is not sufficient to provide reasonable privacy guarantees, as local model updates may result in significant privacy leakage about local training data of participants. To address this issue, in this paper, we present an alternative approach that combines distributed differential privacy (DDP) with a three-layer encryption protocol to achieve a better privacy-utility tradeoff than the existing DP-based approaches. An unbiased encoding algorithm is proposed to cope with floating-point values, while largely reducing mean squared error due to rounding. Our approach dispenses with the need for any trusted server, and enables each party to add less noise to achieve the same privacy and similar utility guarantees as that of the centralized differential privacy. Preliminary analysis and performance evaluation confirm the effectiveness of our approach, which achieves significantly higher accuracy than that of local differential privacy approach, and comparable accuracy to the centralized differential privacy approach.**

***Index Terms*—Privacy-preserving, distributed learning, distributed differential privacy, encryption.**

I. Introduction

Collaborative learning is motivated by the need to train a more accurate global model on massive data collected from multiple parties, as insufficient local training data may end up with worse models with poor generalizability. In traditional centralized deep learning, participants pool their data into a trusted server to train a global model. The assumption of a trusted server that stores data in the clear is ill-suited for many applications as it constitutes a single point of failure for data breaches, and saddles the trusted curator with legal and ethical obligations to keep the user data secure [1]. Hence, parties should be prevented from sharing their private data in the clear for legal reasons. To alleviate this problem, distributed deep learning appears. Decomposing and parallelizing computation among different parties who share the common goals in building global model could help reduce the demand for resources on the centralized server. Another more important reason is the growing privacy and confidentiality challenges, as data collected by participants commonly contain sensitive information, which could be used for unintended purposes, posing immediate or potential privacy risks. Dean *et al.* [2] firstly introduced the concept of distributed deep learning, where instead of explicitly sharing training data, parties collaboratively train a global model by sharing local model updates with a *parameter server* (PS), while updating their local models by downloading the most-updated parameters from the PS. In particular, a special case is federated learning, which is tailored to deal with non-independent and identically distributed (non-IID), unbalanced and massively distributed data in mobile application [5].

Considering the ever increasing privacy issues, multiple trusted server based mechanisms have been developed to ensure privacy [6], [7]. For instance, McMahan *et al.* [7] adopted differential privacy by enabling the trusted server to add tailored noise to the weighted-average of user updates to guarantee user-level privacy. However, the default trusted server is entitled to see all users' update clearly, which is not preferred when the server is untrusted. This is because directly releasing local model updates may result in significant privacy leakage about local training data [8], either to a eavesdropper, or to a central server [7]. For instance, as demonstrated by Aono *et al.* [9], even a small portion of original gradients may reveal information of local data, and thus the server can extract individual data with non-negligible probability in the case of only one neuron, and even for general neural networks with regularization, the gradients can still reveal label information of the original data. As a result, when the server is untrusted, we cannot rely on the server with the task of noise generation. Neither can we entrust any single participant with this task, since the designated participant knows the noise and hence can deduce from the output the true aggregate as well.

Without an untrusted server, Shokri *et al.* [4] proposed to blur local model gradients by adding noise using differential privacy. However, their privacy bounds are given per-parameter, the large number of parameters prevents their technique from providing a meaningful privacy guarantee. An alternative secure aggregation protocol was proposed by Bonawitz *et al.* [10], which is proved to be secure in the honest-but-curious and active adversary settings, and the security is maintained even if an arbitrarily chosen subset of users dropout at any time. In particular, secure multiparty computation (SMC) is leveraged to compute the sum of local model updates from individual users' devices in a secure manner, which comes at the cost of extra computation and communication overheads. Aono *et al.* [9] used the additively homomorphic encryption to

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

preserve the privacy of gradients and enhance the security of the system. However, their protocol not only brings a large communication and computational overhead, but also is not resistant to the collusion between the server and multiple users. More recently, Truex *et al.* [11] recommended a hybrid approach to privacy-preserving federated learning. However, it is unclear how they feed the floating-point model updates into the Paillier cryptosystem while maintaining utility, and their adopted Paillier cryptosystem is much more expensive than the symmetric key based encryption. We also remark that most randomization-based schemes did not give theoretic privacy bounds [12], [13], thus not preferred.

To address all these issues, we propose a new privacy-preserving distributed learning approach using lightweight crypto-assisted distributed differential privacy. To summarize, the main contributions of this paper include:

- We propose a novel approach for privacy-preserving distributed learning, it combines distributed differential privacy with encryption to remove the reliance on any trusted party, and offer desirable utility and privacy.
- For the encryption part, we provide an efficient three-layer encryption protocol to ensure both individual privacy and server obliviousness. To cope with floating-point value, we propose an unbiased encoding algorithm, which largely reduces mean squared error due to rounding.
- Experimental results demonstrate that our approach delivers comparable performance to the centralized DP model, and yields significantly better results than the local DP model, thus confirming the effectiveness of our approach.

II. Preliminaries

A. Differential Privacy

As defined in Definition 1, differential privacy (DP) bounds the effect of the presence or the absence of a record on the output likelihood within a small factor ϵ. The additive term δ allows that the unlikely responses do not need to satisfy the pure ϵ-DP criterion.

Definition 1. *For scalars $\epsilon > 0$ and $0 \leq \delta < 1$, mechanism $\mathcal{M}$ is said to preserve (approximate) (ϵ, δ)-differential privacy if for all neighbouring pairs $D, D' \in \mathcal{D}^n$ and measurable $S \in \mathcal{R}$,*

$$\Pr\{\mathcal{M}(D) \in S\} \quad \leq \quad \exp(\epsilon) \cdot \Pr\{\mathcal{M}(D') \in S\} + \delta \ .$$

Local Differential Privacy (LDP). Compared to the centralized setting, the local version of DP offers a stronger level of protection, because each user individually applies a randomized algorithm $\mathcal{M}$ to obtain a perturbed response before submitting to an untrusted server. Crucially, the randomized algorithm $\mathcal{M}$ guarantees differential privacy independently of the server and the other users, even if they collude [14].

However, LDP comes at the cost of utility, since every user independently adds noise to ensure LDP, the effect of noise adds up when aggregating their results. While noise of scale (standard deviation) $\Theta(1)$ suffices for CDP, Duchi *et al.* [15] proved that the error of LDP must blowup by at least $\Theta(\sqrt{n})$ (n is the number of users), and often by an additional factor growing with the data dimension. This gap is essential for eliminating the trust in the centralized server, and cannot be removed by algorithmic improvement [16]. The most recent works introduced amplification by shuffling to lower the privacy cost of LDP algorithm when viewed in the central model of DP [17], [14]. LDP with shuffling yields a trust model which sits in between the central and local models for DP.

Distributed Differential Privacy (DDP). The notion of DDP reflects the fact that the noise in the target statistic is sourced from multiple parties. Given that the server evaluates a function $h : \mathcal{D}^n \to \mathcal{R}$ on randomized statistics of n parties , and a set of compromised parties K, we let $\boldsymbol{r}_K := \{\boldsymbol{r_i} : i \in K\}$ and denote $\bar{K}$ be the complement of K, *i.e.*, uncompromised set $\bar{K} = \{1, 2, ..., n\} \setminus K$. A formal DDP definition adapted from [18] is given as follows.

Definition 2 ($(\epsilon, \delta, \gamma)$-Distributed DP)**.** *Let $\epsilon > 0$, $0 \leq \delta < 1$ and $0 < \gamma < 1$, We say that the data randomization procedure $\mathcal{M}$ with randomness over the joint distribution of $\boldsymbol{r} := (\boldsymbol{r_1}, \cdots, \boldsymbol{r_n})$ preserves $(\epsilon, \delta, \gamma)$-distributed differential privacy (DDP), with respect to the function h and under γ fraction of uncompromised parties if the following conditions hold: For any neighbouring databases $D, D' \in \mathcal{D}^n$ that differ in one record, for any measurable subset $S \subseteq \mathcal{R}$, and for any subset $\bar{K}$ of at least γn honest parties,*

$$\Pr(\mathcal{M}(D) \in S|\boldsymbol{r}_K) \quad \leq \quad \exp(\epsilon) \cdot \Pr(\mathcal{M}(D') \in S|\boldsymbol{r}_K) + \delta.$$

In the above definition, DDP is parameterized by a probability γ, and is conditioned on the randomness $\boldsymbol{r}_K$ from the compromised parties, *i.e.*, it ensures that if at least $t = \gamma n$ participants are honest and uncompromised, we can still accumulate noise of a similar magnitude as that of the CDP.

DDP achieves a middle ground between CDP and LDP, in terms of both privacy and utility. Approaches to DDP that implement an overall additive noise mechanism by summing the same mechanism run at each party (typically with less noise) necessitates mechanisms with stable distributions (like Gaussian distribution, Binomial distribution)—to guarantee proper calibration of known end-to-end response distribution—and cryptography for hiding all but the final result from participants, as evidenced by [18], [19], [20], [21], [11].DDP utilizes this nice stability to permit each party to randomise its local statistic to a lesser degree ($\frac{\sigma}{\sqrt{n}}$) than would LDP (σ). In summary, the goal of DDP is to both avoid the trust on any third party (trusted server in CDP and trusted shuffler in LDP with shuffling), and achieve better utility than LDP. On the other hand, if the server colludes with all the parties except the victim, the privacy guarantee would downgrade to LDP.

In particular, crypto-assisted DDP differs from LDP with shuffling in two major ways: (1) shuffler model results in an approximate DP guarantee $(\epsilon\sqrt{\frac{\log\frac{1}{\delta}}{n}}, \delta)$ which incurs an expected error of $O(\epsilon\sqrt{log\frac{1}{\delta}})$. In practice, δ has to be smaller than $\frac{1}{n}$ in order to offer a meaningful privacy guarantee. In contrast, our crypto-assisted DDP can achieve the same order

of accuracy guarantees as that of the CDP model; (2) the line of work in LDP with shuffling all require an additional trusted shuffler that receives user messages and permutes them before they are handled to an untrusted server, limiting its practicality as trusted shuffler is notoriously difficult to achieve in practice. Moreover, it is still unclear whether LDP with shuffling can be adapted to the iterative deep learning process.

Privacy Accountant. In terms of privacy leakage, as deep learning needs to iterate over the training data and apply gradient computation multiple times during the training process, each access to the training data incurs some privacy loss from the overall privacy budget ϵ. The repeated applications of additive noise mechanisms follow from the composition theorems and their refinements [22]. The task of keeping track of the accumulated privacy loss in the course of execution of a composite mechanism, and enforcing the applicable privacy policy, can be performed by the privacy accountant.

Abadi *et al.* [6] first proposed moments accountant to provide a tighter bound on the privacy loss compared to the generic strong composition theorem. It keeps track of a bound on the moments of the privacy loss random variable to compute the spent privacy over the course of training. Another more state-of-the-art privacy accountant is Rényi Differential Privacy (RDP), which generalizes pure differential privacy and is closely related to the moments accountant. As defined in Definition 3, RDP is stated in terms of the Rényi divergence. RDP supports exact numerical computations for a finite number of orders, avoiding the need to specify a discrete list of moments ahead of time as required in the moments accountant approach of [6], and privacy bounds are tighter than those provided by Abadi *et al.* [6]. Hence, we choose to use RDP as a more natural analysis framework.

Definition 3 ((α, ϵ)-RDP [23])**.** *A randomized mechanism $f : \mathcal{D} \to \mathcal{R}$ is said to guarantee (α, ϵ)-RDP, if for any adjacent $D, D' \in \mathcal{D}$ it holds that*

$$D_\alpha(f(D)||f(D')) \leq \epsilon$$

B. Homomorphic Encryption

Additive homomorphic encryption allows the calculation of the encrypted sum of plaintexts from their corresponding ciphertexts. It ensures that the server can only decrypt the sum of all the received ciphertexts, but cannot access any of them. Specifically, Vernam cipher or one-time pad (OTP) has been mathematically proved to be completely secure, which cannot be broken given enough ciphertext and time. Therefore, we use simple but provably secure OTP for additively homomorphic encryption that allows efficient aggregation of ciphertexts [24], [25], the ciphertexts refer to the encrypted gradients in the context of sharing gradients in distributed learning. The main idea of forming the ciphertext is to combine the keystream with the plaintext digits. Meanwhile, rather than XOR operation typically found in stream ciphers, which is unsecured under the frequency analysis attacks, our encryption scheme uses modular addition (+), and is hence very efficient [24]. The security relies on two important features: (1) the keystream changes from one message to another; and (2) all the operations are performed modulo a large integer M [24].

III. Proposed Approach

Threat Model. In distributed learning system, insider attacks represent those launched by the data owners (participants) as well as the server, while outsider attacks include those launched by the eavesdroppers to the communication channel or the users of the final model when deployed as a service.

- *Insiders*. For insider in the distributed system, we consider the commonly used semi-honest or honest-but-curious adversarial model, where the parties or the server may attempt to learn or infer sensitive information from the honest parties, without deviating from the protocol. Moreover, a subset of parties may collude with the server to infer private information of other parties.
- *Outsiders*. We also consider potential attacks from adversaries outside of the system. Our work ensures that any adversary monitoring communications during training process cannot infer any private information about the participants. We also consider users of the final model as potential adversaries. The final model can be deployed as a service, while resisting to these adversaries.

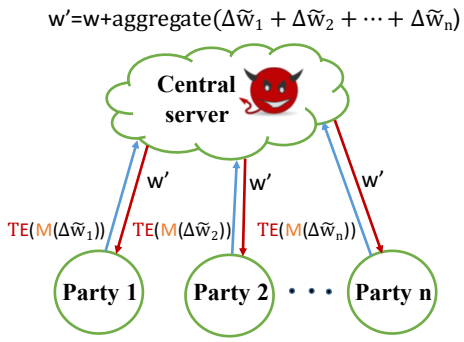

Fig. 1: DDP mechanism ($\mathcal{M}$) combined with three-layer encryption (TE): distributed learning without a trusted server. ($\Delta\tilde{\boldsymbol{w}}_j$: noisy local model updates).

A. System Setup

We consider the scenario where multiple parties each owns different groups of individuals with similar features, *i.e.*, horizontal distributed learning. For example, different hospitals, each holding the same kind of information for different patients, can collaboratively train a global model on the union of their patients, while ensuring privacy for each patient. Consequently, hereafter, instead of training a centralized model to solve the task associated with the whole database $\mathbf{D} \in \mathbb{R}^{r \times d}, \boldsymbol{Y} \in \mathbb{R}^r$, the whole database $\mathbf{D}$ is partitioned into n disjoint subsets, $\{\mathbf{D}_1, \mathbf{D}_2, ...\mathbf{D}_n\}$ that are held by n parties who are unwilling to make their training data, model parameters or model predictions public or share them with others. Here r refers to the total number of training records in $\mathbf{D}$, $\mathbf{D}_i$ and $\boldsymbol{Y}_i$ represent the training data and labels of party i respectively. Individual models are trained separately on each subset $\{\mathbf{D}_i, \boldsymbol{Y}_i\}$. The

key component in our system is to protect individual privacy by combining DDP with an efficient three-layer encryption protocol (DDP+TE). Figure 1 shows an outline of an epoch of training in the distributed system without a trusted server. During each epoch of local training, each participant adds an appropriate amount of noise according to the privacy budget allocated to that step, the sensitivity of the gradients, and the trust level γ in the system. The noisy response is finally encrypted using our three-layer encryption protocol and sent to the server. Homomorphic property allows the server to correctly aggregate individual responses, and expose the updated global model to all participants. The detailed process of privacy-preserving distributed learning between the server and parties is given in Algorithm 1, where we use Enc, $Senc$, and $Aenc$ to refer to homomorphic encryption, symmetric key encryption, and asymmetric/public key encryption, respectively. For privacy accountant, the server uses RDP to keep track of the spent privacy budget during distributed training process.

Algorithm 1 Privacy-Preserving Distributed Learning

Input: Set of parties P; number of honest, non-colluding parties $t = \gamma n$; noise parameter σ; learning rate η; sampling probability q; loss function $\mathcal{L}$; clipping value c; number of epochs E; public key pk.

Role: Party j

Downloads $\boldsymbol{w}$ from server and replaces local model $\boldsymbol{w}_j$ with $\boldsymbol{w}$;

for $t \in [1/q]$ **do**

Takes a random sample L_t with probability $q = L_t/|\mathbf{D}_i|$;

for $x \in L_t$ **do**

Computes gradient: $\Delta \boldsymbol{w}_j \leftarrow \nabla_{\boldsymbol{w}_j} \mathcal{L}(\boldsymbol{w}_j, x)$;

Clips gradient: $\Delta \overline{\boldsymbol{w}}_j \leftarrow \Delta \boldsymbol{w}_j / max(1, \|\Delta \boldsymbol{w}_j\|/c)$;

end for

Adds noise: $\Delta \hat{\boldsymbol{w}}_j \leftarrow \frac{1}{|L_t|} \sum_{x \in L_t} \Delta \overline{\boldsymbol{w}}_j + \mathcal{N}(0, c^2 * \frac{\sigma^2}{t} \boldsymbol{I})$.

$\boldsymbol{w}_j \leftarrow \boldsymbol{w}_j - \eta * \Delta \hat{\boldsymbol{w}}_j$

end for

$\Delta \tilde{\boldsymbol{w}}_j \leftarrow \boldsymbol{w}_j - \boldsymbol{w}$

Encrypts $\Delta \tilde{\boldsymbol{w}}_j$ with its keystream k_j as $c_j = Enc(\Delta \tilde{\boldsymbol{w}}_{\boldsymbol{j}}, k_j)$, and re-encrypts the encrypted gradients c_j with a fresh symmetric encryption key fsk as $Senc(c_j, fsk)$, the symmetric encryption key of the second layer is encrypted in the third layer by the server's public key pk as $Aenc(fsk, pk)$. Finally, the two-layer encrypted gradients $Senc(c_j, fsk)$ and the encrypted fresh symmetric encryption key $Aenc(fsk, pk)$ are sent to the server.

Role: Server

Initializes global parameters $\boldsymbol{w}$;

for $e \in [E]$ **do**

Server uses the paired secret key sk to decrypt the received $Aenc(fsk, pk)$ as fsk, then uses fsk to decrypt the received $Senc(c_j, fsk)$ as $c_j = Enc(\Delta \tilde{\boldsymbol{w}}_{\boldsymbol{j}}, k_j)$, finally decrypts the sum of all c_j using homomorphic property and updates global parameters as: $\boldsymbol{w}' = \boldsymbol{w} + Dec(\sum_{j \in P} Enc(\Delta \tilde{\boldsymbol{w}}_{\boldsymbol{j}}, k_j), -k_0) = \boldsymbol{w} + \sum_{j \in P} \Delta \tilde{\boldsymbol{w}}_{\boldsymbol{j}}$.

Sends $\boldsymbol{w}'$ to all parties for the next epoch of training.

Computes the overall privacy cost (ϵ, δ) given σ, q, e.

end for

B. Three-layer Encryption Protocol

Sharing gradients can prevent direct exposure of the local data, but may indirectly disclose local data privacy. To further avoid privacy leakage from gradients and facilitate gradients aggregation, we develop an efficient three-layer encryption protocol, which can be combined with DDP to ensure individual privacy and maintain utility. As the released gradient vector is high-dimensional, directly encrypting gradient vector using public key cryptography (asymmetric encryption) like Paillier is both computation and communication expensive. Therefore, we instead use the a more simple but provably secure technique called OTP for the purpose of homomorphic encryption. In more details, in our three-layer encryption protocol, the first layer protects local model gradients by using an efficient additive homomorphic encryption scheme, and the second layer and the third layer constitute a hybrid cryptosystem.

The First Layer: Additive Homomorphic Encryption. The detailed procedure for homomorphic encryption is presented in Algorithm 2. In practice, if p = max(x_i), M is derived as $M = 2^{\lceil log_2(p \times n) \rceil}$. All computations in the remainder of this paper are modulo M unless otherwise noted. A pseudorandom keystream k can be generated by a secure pseudo random function (PRF) by implementing a secure stream cipher, such as Trivium [26], keyed with each party's keystream k_i with a unique message ID and the server's keystream k_0. For encryption purpose, the secret keys are pre-computed through a trusted setup, which can be performed by a trusted dealer or through a standard SMC protocol, but the generated keystreams cannot be used more than once. The trusted setup generates non-zero random shares of 0, such that each participant $j \in P$ obtains a keystream k_j, and the server obtains the capability k_0 needed for decryption of the aggregate in each epoch, *i.e.*, $\sum_{j \in P} k_j + k_0 = 0$. Homomorphic encryption ensures server obliviousness by capturing the following security notions:

- The server knows nothing but the sum of all the received gradients in each epoch.
- If a set of compromised participants and the server form a coalition against the remaining participants, *i.e.*, the compromised participants can reveal their gradients, noise values and keystreams to the server as a form of auxiliary information, then the server can inevitably learn the sum of gradients of the remaining participants. In this case, the server learns no additional information.

The Second and Third Layers: A Hybrid Cryptosystem. In our three-layer encryption, the second layer is a data encapsulation scheme, which is a symmetric-key cryptosystem; and the third layer is a key encapsulation scheme, which is a public-key cryptosystem. For the second layer, a fresh symmetric encryption key fsk will be generated and used to re-encrypt the ciphertext of the first layer, and then the fresh symmetric key is encrypted by using the server's public key pk in the third layer. In this way, the encryption of high-dimensional data becomes very effective, and the server could be authenticated as well: only the server who has the corresponding secret key sk paired with the public key pk can decrypt the two-layer encrypted gradients, preventing the success of eavesdropping attack.

C. Unbiased Encoding

Like most cryptographic techniques, our encryption protocol requires all the original floating-point values to fall into a

Algorithm 2 Homomorphic Encryption Scheme

Setup

1: A trusted dealer randomly generates $n+1$ keystreams: $k_0, k_1, \ldots, k_n \in [0, M-1]$, such that $k_0 + \sum_{j \in P} k_j \pmod M = 0$, where M is a large integer, and $n = |P|$.
2: Party j obtains keystream k_j, and the server obtains the capability k_0.

Enc(m, k)

1: Represent message m as integer $m \in [0, M-1]$.
2: Let k be a randomly generated keystream, where $k \in [0, M-1]$.
3: Compute $c = Enc(m, k) = m + k$.

Dec(c, k)

1: $Dec(c, k) = c - k$.

AggrDec(k_0)

1: Let $c_j = Enc(m_j, k_j)$, where $j \in P$.
2: Server uses $-k_0 = \sum_{j \in P} k_j$ to decrypt the aggregation of other parties as follows: $Dec(\sum_{j \in P} c_j, -k_0) = \sum_{j \in P} c_j - \sum_{j \in P} k_j = \sum_{j \in P} m_j$.

integer-valued discrete group. For this purpose, we first need to map floating-point values to the integer domain through scaling and rounding, then convert to floating-point values through unscaling after decryption. In more detail, scaling and rounding operations are firstly applied to the privatized local model updates by each party, while unscaling is applied by the server after the decryption of the sum of the privatized local model updates is derived, hence we explicitly define scaling and rounding operations as *SR* and unscaling operation as *U*.

However, rounding may affect the precision of the float value, thus affecting utility, especially aggregating multiple values with reduced precision. The most common practice to encode a float value x with precision s is called Scaling, Rounding, Unscaling (SRU) algorithm [25], where $x \mapsto \frac{\lfloor sx \rfloor}{s}$. However, in SRU, the error of a scalar is upper-bounded by $\frac{1}{2s}$, then the error of the aggregate is upper-bounded by $\frac{n\sqrt{d}}{2s}$, where n and d represent the number of parties and the size of the locally released vector [27]. It may result in a large error when both n and d are large. To alleviate this issue, we propose an unbiased encoding algorithm, where the operation of rounding is replaced with randomized rounding. As shown in Algorithm 3, each scalar $x \in \mathbb{R}$ is first scaled and rounded as $\bar{x} = \lfloor sx \rfloor + \text{Ber}(sx - \lfloor sx \rfloor)$, where $\text{Ber}(sx - \lfloor sx \rfloor)$ refers to a Bernoulli random variable which takes the value 1 with probability $p = sx - \lfloor sx \rfloor$. Finally, $\bar{x}$ can be unscaled by dividing by the same scaling factor s. Theorem 1 states that this algorithm is an unbiased encoding, followed by the detailed proof. This ensures that $\mathbb{E}[\bar{x}/s] = \mathbb{E}[x]$ and that the mean squared error (MSE) due to rounding (which equals the variance) is at most $\frac{1}{4s^2}$. From Theorem 1, it then follows that $\mathbb{E}[\sum_{j \in P} \bar{x}_j / s] = \mathbb{E}[\sum_{j \in P} x_j]$, *i.e.*, the sum of real scalars is also unbiased in expectation and MSE of the sum over n scalars due to rounding is at most $\frac{n}{4s^2}$, and MSE of the sum over n vectors (suppose each vector is composed of d scalars) due to rounding is at most $\frac{n\sqrt{d}}{4s^2}$. In our case, each x is an element of the locally computed floating-point model updates, d is the size of local model updates, and $n = |P|$ is the number of parties. After the server decrypts the sum of the privatized local model updates, it unscales the sum back to the unbiased floating-point vector with precision s.

Algorithm 3 Unbiased Encoding

Input: scaling factor s**, scalar** x
Scaling and Randomized Rounding: $x \mapsto \bar{x} = \lfloor sx \rfloor + \text{Ber}(sx - \lfloor sx \rfloor)$

Unscaling: $\bar{x} \mapsto \frac{\bar{x}}{s}$.

Theorem 1. *(Unbiased encoding for scalar x). For a specific scaling factor s and any scalar $x \in \mathbb{R}$, the encoding in Algorithm 3 is unbiased in expectation, and the mean squared error (MSE) due to rounding is at most $\frac{1}{4s^2}$.*

Proof. For any scalar $x \in \mathbb{R}$ and scaling factor $s \in \mathbb{R}^+$, the expectation of the difference between the encoded scalar $\mathbb{E}[\bar{x}/s]$ and the scalar x can be expressed as:

$$\begin{aligned}\mathbb{E}[\bar{x}/s - x] &= \mathbb{E}[\{\lfloor sx \rfloor + Ber(sx - \lfloor sx \rfloor)\}/s - x] \\ &= \mathbb{E}[\tfrac{\lfloor sx \rfloor + Ber(sx - \lfloor sx \rfloor) - sx}{s}] = \tfrac{\lfloor sx \rfloor + sx - \lfloor sx \rfloor - sx}{s} = 0\end{aligned}$$

where the second last equality follows from the property of Bernoulli distribution: $\mathbb{E}[\text{Ber}(p)] = p$. For MSE,

$$\begin{aligned}\mathbb{E}[(\bar{x}/s - x)^2] &= \mathbb{E}[(\{\lfloor sx \rfloor + Ber(sx - \lfloor sx \rfloor)\}/s - x)^2] \\ &= \tfrac{\mathbb{E}[(\lfloor sx \rfloor + Ber(sx - \lfloor sx \rfloor) - sx)^2]}{s^2}\end{aligned}$$

let $p = sx - \lfloor sx \rfloor$, then above expression reduces to:

$$\begin{aligned}\frac{\mathbb{E}[(Ber(p) - p)^2]}{s^2} &= \frac{\mathbb{E}[(Ber(p))^2 - 2pBer(p) + p^2]}{s^2} \\ &= \frac{p - p^2}{s^2} \leq \frac{1}{4s^2}\end{aligned}$$

where the last equality follows from $\mathbb{E}[(Ber(p))^2] = p$ and $\mathbb{E}[\text{Ber}(p)] = p$, and the last inequality follows from the maximum value of $f = p - p^2$ equals $1/4$ when $p = 1/2$. □

In terms of utility analysis, it is apparent that MSE is inversely proportional to the scaling factor s, where $s = 10^N$ considers N digits to the right of the decimal point. Therefore, the higher s, more precision is kept. To avoid the utility degradation due to rounding, we choose a large scaling factor as $s = 10^5$ in all experiments, the upper bound of the MSE of the sum of the privatized local model updates $\frac{n\sqrt{d}}{4s^2}$ becomes negligible even when $n = 1000$, and $d = 10^6$. In contrast, normal SRU without unbiased encoding [27] results in a relatively large MSE, as $\frac{n\sqrt{d}}{2s} >> \frac{n\sqrt{d}}{4s^2}$. In terms of privacy analysis, applying scaling and rounding to the private data will not diminish its privacy guarantee according to the post-processing property of DP [22]. Moreover, rounding provides the covertness property by making the perturbed floating-point values as integers.

D. System Robustness

In distributed system, it might happen that, one or more participants might fail to respond or dropout the participation at some point before the completion of the given training epoch [10]. This would introduce two effects: first, differential privacy may not be guaranteed any more, since the sum of the noise will not meet the required level σ. Second, the server will not be able to decrypt the correct aggregate statistics, since $k_0 + \sum_{j \in P} k_j = 0$ will not hold any more. Fault tolerance requires the server to be resilient to party failures, *i.e.*, server can still be able to estimate the correct aggregate of the remaining parties even when an arbitrary subset of parties (unknown in advance) fail. Notice that all the existing fault tolerance schemes come with a trade-off [21], [18], [28]. In this section, we further extend our scheme to resist party failures by introducing setup extension and sanitization extension.

Setup Extension. The failure or dropout of any party can be detected as the server will not receive its message ID. Then the failed ID will be reported to the trusted third party, who initializes the trusted setup and issues certificates to all the parties including the server in the system. To deal with party failure and ensure the correctness of decryption, the trusted third party will update the keystream kept by the server as the negative sum of the remaining parties. Therefore, the corresponding keys of the non-responding parties will not be included during decryption at the server. Hence, our solution requires setup only once if there are no party failures or dropout, otherwise, one extra setup is needed only between the trusted third party and the server.

Sanitization Extension. In order to resist the failure or dropout of $n - t$ parties, each party should follow the strategy below to inject noise into individual training process: If at least $t = \gamma n$ honest participants out of the total n participants participate in the protocol, then we can distribute the noise generation task amongst these t participants. Our construction guarantees that, with high probability, the noise introduced by the honest participants can account for party failures, dropout, and collusion. Each honest party samples r_i from a Gaussian distribution of standard deviation $\sigma_i = \frac{\sigma}{\sqrt{t}}$ instead of $\frac{\sigma}{\sqrt{n}}$, then the total noise in the aggregated gradients still satisfies the expected standard deviation σ, *i.e.*, r_i is chosen from a Gaussian distribution that is sufficient to guarantee differential privacy, hence the aggregation remains differentially private. Note that in this case each party may add extra noise to the aggregation in order to ensure differential privacy even if less than $n-t$ parties fail, dropout, or collude. Clearly, this extra noise increases the error if all n parties operate correctly and add their noise shares faithfully. In the worst case, if every participant believes that the other $n - 1$ participants are compromised, each participant would need to add sufficient noise to ensure LDP of its own gradients, resulting in a large error in the aggregated gradients.

Our three-layer encryption protocol also allows the server to verify the authenticity of the received message (authentication) and check that the received message is intact (integrity), thus preventing an outsider from injecting fake packets.

IV. Experiments

For local model training, we follow the same privacy strategy used in the centralized training approach [6]. Each party is first initialized with the same model structure with same parameters. Each party will then conduct one full epoch of learning locally. At the end of each lot (several batches), Gaussian noise is introduced according to the norm clipping bound c and the noise parameter σ. Norm clipping bounds the influence of each individual example on the gradient, *i.e.*, sensitivity is bounded. After each party completes one epoch of local training with $\frac{1}{q}$ lots, where $q = L/N$ is the sampling probability per lot, each party sends its encrypted local gradients to the server. The server then decrypts the sum of all the received gradients, and updates the global model by integrating the average of local updates for the next epoch of local learning at each party.

Dataset and Model. We evaluate our approach on two standard image-classification datasets, MNIST and CIFAR-10 for fair comparison with [6]. MNIST dataset is for handwritten digit recognition, which consists of 60,000 training examples and 10,000 testing examples. Each example is a 28x28 grey-scale image, with 10 possible digits ["0"-"9"] locating at the center of the image[1]. We use a model structure similar to that in [6], *i.e.*, a feedforward neural network with 2 hidden layers of ReLU units and a softmax layer of 10 classes with cross-entropy loss. The first layer (PCA layer) contains 60 units and the second layer contains 1000 units. We set the norm clipping to 4.0, learning rate to 0.1 and sampling probability q to 0.01.

CIFAR-10 consists of color images classified into 10 classes and partitioned into 50,000 training examples and 10,000 test examples[2]. Each example is a 32x32 RGB image. We use the same model structure as that in [6], which consists of two convolutional layers followed by two fully connected layers. The convolutional layers use 5x5 convolutions with stride 1, followed by a ReLU and 2x2 max pools, with 64 channels each. The output of the last convolutional layer is fed into a fully connected layer with 384 units, and another one of the same size. As is standard for image datasets, we use data augmentation during training. For each training image, we generate a new distorted image by randomly picking a 24x24 patch from the image, randomly flipping the image along the left-right direction, and randomly distorting the brightness and the contrast of the image. For all datasets, we randomly and equally split all training data among all parties.

Baseline Comparison. We compare our approach (DDP+TE) with the following baselines: (1) Centralized non-private (CNP): in this approach, all the data is centrally held by the server and no privacy is considered in the learning process. (2) Centralized DP (CDP): while all the data is still centrally held by the server, the server now conducts deep learning with differential privacy. This is representative of the scenario in [6]. (3) Local DP (LDP): this approach excludes cryptosystem but requires each party to add enough noise of

[1]http://yann.lecun.com/exdb/mnist/

[2]https://www.cs.toronto.edu/~kriz/cifar.html

level σ to individually ensure local differential privacy before sharing their local model updates with the server.

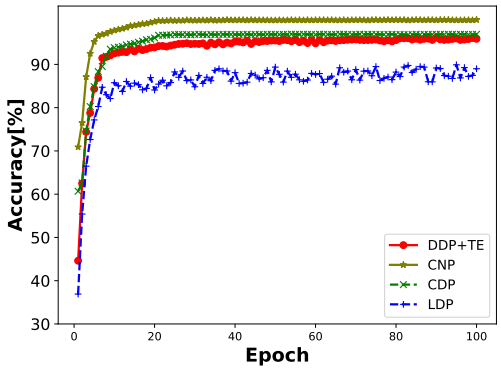

Fig. 2: Model convergence on MNIST dataset (DDP+TE adopts 10 parties and σ=4, all DP models adopt $(\epsilon, \delta) = (2, 10^{-5})$).

Note that the distributed non-private model achieves similar or even better results than the centralized non-private model [4], so we omit the comparison with the distributed non-private model.Figure 2 shows the training process of all approaches on MNIST dataset across 100 epochs, for DDP+TE, we set 10 parties with the total noise level $\sigma = 4$ (the medium noise setting in [6]). It can be observed that our approach achieves an accuracy of 0.95 in this setting. While this is lower than the CNP approach where an accuracy of approximately 0.98 is achieved, but our approach approximates CDP approach and significantly outperforms the LDP approach which only reaches 0.86. Additionally, we observe a large variation in the accuracy curve of the LDP approach as updates are overwhelmed by noise. We additionally experiment with $\sigma = 10$ and $\sigma = 2$ as was done in [6]. When $\sigma = 10, (\epsilon, \delta) = (0.5, 10^{-5})$, accuracies of CDP, LDP and our DDP+TE correspond to 0.90, 0.72, and 0.89. When $\sigma = 2, (\epsilon, \delta) = (8, 10^{-5})$, accuracies of CDP, LDP and our DDP+TE become 0.97, 0.93 and 0.97 respectively. We can thus conclude that our approach yields the most gain with larger σ values which translates to tighter privacy guarantees.

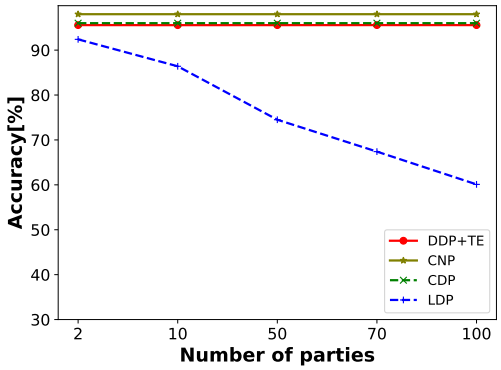

Fig. 3: MNIST accuracy for different number of parties and different approaches (all DP models adopt $(\epsilon, \delta) = (2, 10^{-5})$).

Figure 3 demonstrates the viability of our approach in highly distributed environments while highlighting the shortcomings of the LDP approach. As the number of parties n increases, the noise in the LDP approach increases proportionally while our approach maintains consistent accuracy. We find that with as few as 10 parties, the accuracy of LDP begins to drop significantly. This result can be easily explained by the fact that LDP results in the noise in the aggregated gradients increasing linearly with the number of parties. In contrast, our DDP+TE is nearly independent with the number of parties. We also evaluated our approach in a more severe setting with only half of the parties as honest, while the other half semi-honest and may collude with the server. In this case, $t = n/2$, and each party has to add noise with standard deviation $\sigma_i = \frac{\sigma}{\sqrt{t}}$. Our approach again consistently outperforms LDP. Specifically, after 100 epochs, our approach reached an accuracy of 0.91, while the LDP achieves only 0.77.

Similarly, for CIFAR-10, we follow the same setting in [6], *i.e.*, $\delta = 10^{-5}$, $c = 3$ and $\sigma = 6$, and deploy 10 parties for distributed learning. As shown in Table I, when $\epsilon = 2$, accuracies of CDP, our DDP+TE and LDP become 0.67, 0.66 and 0.51 respectively; when $\epsilon = 4$, accuracies of CDP, our DDP+TE and LDP become 0.70, 0.69 and 0.53 respectively; when $\epsilon = 8$, accuracies of CDP, our DDP+TE and LDP become 0.73, 0.72 and 0.58 respectively. Overall, we have demonstrated that our approach provides significant accuracy gains compared with LDP, and also scales well.

TABLE I: Accuracy on MNIST and CIFAR-10 under varying privacy budget ϵ and different privacy models.

Accuracy	MNIST			CIFAR-10		
	ϵ =0.5	ϵ =2	ϵ =8	ϵ =2	ϵ =4	ϵ =8
CNP	0.98	0.98	0.98	0.86	0.86	0.86
CDP	0.90	0.95	0.97	0.67	0.70	0.73
DDP+TE	0.89	0.95	0.97	0.66	0.69	0.72
LDP	0.72	0.86	0.93	0.51	0.53	0.58

Complexity Analysis. The main communication cost occurs when each party sends its encrypted gradients to the server, resulting in d ciphertexts, where d is the size of the released gradients (the encrypted symmetric key size is negligible compared with the encrypted gradients). Therefore, our encryption scheme using stream ciphers and hybrid encryption is relatively efficient compared with the asymmetric encryption, because encrypting a short plaintext (*i.e.*, the symmetric key fsk) requires only one asymmetric operation, while encrypting a longer message (millions of gradients) would in theory require many asymmetric operations.

For example, for training MLP models (with 117,040 parameters each represented by 4 bytes) on MNIST dataset [4], each party needs to send and download 117,040 $\times$ 4 bytes = 0.468 MB of message in each epoch. For encryption part, each party needs to send an encrypted message consisting of two parts: the gradients encrypted with k_j and fsk, and the encrypted fsk. As symmetric key encryption, such as AES-256, does not increase the size of the message (apart from potentially a few extra bytes for padding), the encrypted gradients in our scheme is of the same size (i.e., 0.468 MB). If we

choose RSA-2048 for the asymmetric key encryption, we only introduce an additional 256-byte message (encrypted symmetric key), which is negligible. In terms of the computation cost spent on encryption, we tested the performance in a local machine with 2.7 GHz Intel Core i7 processor and 16 GB 2133 MHz LPDDR3 memory. The time spent on the first layer of encryption using stream cipher is negligible, as modular addition (+) is very efficient [24]. The second layer of encrypting a 0.468 MB message with AES-256 takes $\sim$7.5 ms, and the third layer of encrypting a 256-bit message using RSA (of key size 2048 bits) takes $\sim$4 ms, hence each party takes total $\sim$11.5 ms on the encryption of local gradients in the system, which is pretty fast.

V. Conclusion and Future Work

In this paper, we present a novel approach that combines DDP with an efficient three-layer encryption protocol to achieve a better privacy-utility tradeoff than the existing approaches. An unbiased encoding algorithm is proposed to cope with floating-point values, while largely reducing mean squared error due to rounding. Using our approach, each user can add less noise while preserving individual privacy and maintaining utility without the need for a trusted server. Preliminary analysis and performance evaluation confirm the effectiveness of our approach, which achieves higher accuracy than that of the local DP model, and yields similar accuracy as that of the centralized DP model. Complexity analysis also validates the applicability of our approach. We expect to extend our approach to various machine learning models and applications. Applying more advanced techniques to further improve performance is another interesting direction. We also expect to adapt LDP with shuffling to the iterative distributed learning process, and conduct deeper comparison.

References

[1] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha, "Outis: Crypto-assisted differential privacy on untrusted servers," *arXiv preprint arXiv:1902.07756*, 2019.

[2] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al., "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[3] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, 2010, pp. 2595–2603.

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

[5] H Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Aguera y Arcas, "Federated learning of deep networks using model averaging," *arXiv preprint arXiv:1602.05629*, 2016.

[6] Martín Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 308–318.

[7] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang, "Learning differentially private recurrent language models," in *Proceedings of the 5th International Conference on Learning Representations*, 2018.

[8] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov, "Exploiting unintended feature leakage in collaborative learning," *arXiv preprint arXiv:1805.04049*, 2018.

[9] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al., "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

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

[11] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 1–11.

[12] Lingjuan Lyu, Yee Wei Law, Sarah M Erfani, Christopher Leckie, and Marimuthu Palaniswami, "An improved scheme for privacy-preserving collaborative anomaly detection," in *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE, 2016, pp. 1–6.

[13] Lingjuan Lyu, James C Bezdek, Yee Wei Law, Xuanli He, and Marimuthu Palaniswami, "Privacy-preserving collaborative fuzzy clustering," *Data & Knowledge Engineering*, vol. 116, pp. 21–41, 2018.

[14] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim, "The privacy blanket of the shuffle model," in *Annual International Cryptology Conference*. Springer, 2019, pp. 638–667.

[15] John C Duchi, Michael I Jordan, and Martin J Wainwright, "Local privacy and statistical minimax rates," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 2013, pp. 429–438.

[16] TH Chan, Kai-Min Chung, Bruce M Maggs, and Elaine Shi, "Foundations of differentially oblivious algorithms," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2019, pp. 2448–2467.

[17] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta, "Amplification by shuffling: From local to central differential privacy via anonymity," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2019, pp. 2468–2479.

[18] Elaine Shi, HTH Chan, Eleanor Rieffel, Richard Chow, and Dawn Song, "Privacy-preserving aggregation of time-series data," in *Annual Network & Distributed System Security Symposium (NDSS)*. Internet Society., 2011.

[19] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2006, pp. 486–503.

[20] Gergely Ács and Claude Castelluccia, "I have a dream!(differentially private smart metering)," in *Information hiding*. Springer, 2011, vol. 6958, pp. 118–132.

[21] Vibhor Rastogi and Suman Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 735–746.

[22] Cynthia Dwork and Aaron Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[23] Ilya Mironov, "Renyi differential privacy," in *Computer Security Foundations Symposium (CSF), 2017 IEEE 30th*. IEEE, 2017, pp. 263–275.

[24] Claude Castelluccia, Einar Mykletun, and Gene Tsudik, "Efficient aggregation of encrypted data in wireless sensor networks," in *Mobile and Ubiquitous Systems: Networking and Services, 2005. MobiQuitous 2005. The Second Annual International Conference on*. IEEE, 2005, pp. 109–117.

[25] Lingjuan Lyu, Karthik Nandakumar, Benjamin Rubinstein, Jiong Jin, Justin Bedo, and Marimuthu Palaniswami, "PPFA: Privacy preserving fog-enabled aggregation in smart grid," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3733–3744, 2018.

[26] Christophe De Canniere and Bart Preneel, "Trivium," in *New Stream Cipher Designs*, pp. 244–266. Springer, 2008.

[27] Lingjuan Lyu, *Privacy-preserving machine learning and data aggregation for Internet of Things*, Ph.D. thesis, The University of Melbourne, 2018.

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