Chapter 14
Private Parameter Aggregation for Federated Learning

K. R. Jayaram and Ashish Verma

Abstract Federated learning enables multiple distributed participants (potentially on different datacenters or clouds) to collaborate and train machine/deep learning models by sharing parameters or gradients. However, sharing gradients, instead of centralizing data, may not be as private as one would expect. Reverse engineering attacks on plain text gradients have been demonstrated to be practically feasible. This problem has been made more insidious by the fact that participants or aggregators may reverse engineer model parameters while participating honestly in the protocol (the so-called *honest, but curious* trust model). Existing solutions for differentially private federated learning, while promising, lead to less accurate models and require nontrivial hyperparameter tuning. In this chapter, we (1) describe various trust models in federated learning and their challenges, (2) explore the use of secure multi-party computation techniques in federated learning, (3) explore how additive homomorphic encryption can be used efficiently for federated learning, (4) compare these techniques with others like the addition of differentially private noise and the use of specialized hardware, and (5) illustrate these techniques through real-world examples.

14.1 Introduction

Some of the early success of distributed machine and deep learning (ML/DL) in several application domains [29, 31] has been in the context of massive centralized data collection, either at a single datacenter or at a cloud service. However, centralized data collection at a (third-party) cloud service can be incredibly privacy-invasive and can expose organizations (customers of the cloud service) to large legal liability when there is a data breach. This is especially true in the case of healthcare data, voice transcripts, home cameras, financial transactions, etc.

K. R. Jayaram (✉) · A. Verma
IBM Research, Yorktown Heights, NY, USA
e-mail: jayaramkr@us.ibm.com; Ashish.Verma1@ibm.com

H. Ludwig, N. Baracaldo (eds.), *Federated Learning*,
https://doi.org/10.1007/978-3-030-96896-0_14

Centralized data collection often results in “loss of control” over data once it is uploaded. A frequently asked question to which users often do not get a satisfactory answer is “Is the cloud service using my data as promised? Is it actually deleting my data when it claims to do so?”. Organizations that have not been convinced by privacy violations and loss of control have been forced by governmental regulations (like HIPAA and GDPR [34]) to restrict data sharing with third-party services.

Federated learning (FL) aims to mitigate these aforementioned issues while maintaining accuracy of ML/DL models. An entity in an FL job can be as small as a smart phone/watch or as large as an organization with multiple data centers. An FL algorithm aims to train an ML/DL model, e.g., a specific neural network model or an XGBoost model, on multiple entities, each with its own “local” dataset, without exchanging any data. This results in multiple “local models,” which are then combined (aggregated) by exchanging only parameters (e.g., the weights of a neural network model). An FL algorithm may use a central coordinator to collect parameters of all local models for aggregation, or it may be a peer-to-peer algorithm (broadcast, overlay multicast, etc.)

Initially, it was believed that the exchanged model updates in Federated Learning (FL) communications would contain far less, if any, information about the raw training data. Thus, sharing model updates was considered to be “privacy-preserving.” However, even if not discernible immediately, training data information is still embedded in the model updates. Recent research [14, 15, 17, 33, 42, 48, 50, 51] has demonstrated the feasibility and ease of inferring private attributes and reconstructing large fractions of training data by exploiting model updates, thereby challenging the privacy promises of FL in the presence of honest-but-curious aggregation servers.

14.2 Focus, Trust Model, and Assumptions

FL is *typically* deployed in two scenarios: *cross-device* and *cross-silo* [23]. The *cross-device* scenario involves a large number of parties (>1000), but each party has a small number of data items, constrained compute capability, and limited energy reserve (e.g., mobile phones or IoT devices). They are highly unreliable and are expected to drop and join frequently. Examples include a large organization learning from data stored on employees’ devices and a device manufacturer training a model from private data located on millions of its devices (e.g., Google Gboard [4]). A *trusted authority*, which performs aggregation and orchestrates training, is typically present in a *cross-device* scenario. Contrarily, in the *cross-silo* scenario, the number of parties is small, but each party has extensive compute capabilities (with stable access to electric power or equipped with hardware machine learning (ML) accelerators) and large amounts of data. The parties have reliable participation throughout the entire FL training lifecycle but are more susceptible to sensitive data leakage. Examples include multiple hospitals collaborating to train a tumor detection model on radiographs, multiple banks collaborating to train a credit card fraud detection model, etc. In *cross-silo* scenarios, there exists *no presumed central trusted authority*. All parties involved in the training are *equal* collaborators. The deployments often involve hosting aggregation in public clouds, or alternatively one of the parties acting as, and providing infrastructure for aggregation. In this chapter, we focus on private parameter aggregation in *cross-silo* scenarios.

We assume that each participant is convinced of the benefits (improvements in accuracy, robustness, etc.) of federated learning. We note that convincing participants to collaborate by projecting potential gains in accuracy due to federated learning is an open research problem. We focus on the so-called *honest, but curious* trust model. Here, each participant is convinced enough that it follows the steps of the federation protocol and does not collude with the coordinator to break the protocol. But the participant may be curious about the data of others, and it may be in their interest to reverse engineer the model parameters to try and discover other participants' data. We also assume that the coordinator is honest but curious with respect to individual participants' data. The participants want to reduce the required amount of trust in the coordinator as much as possible. We also assume that each participant does not attempt to poison or skew the global model by maliciously generating weights. This trust model also includes the simpler case where participants are forbidden from sharing data or the model parameters derived from the data due to regulatory reasons (e.g., FedRAMP, EU data protection guidelines [34]).

Examples and Use Cases This trust model is predominantly found in enterprise federated learning; for example, a multinational bank having branches in multiple countries and so regulated locally (BankA, BankA US, BankA UK, BankA India, etc.), where the bank wants to learn a fraud detection model across data of all its subsidiaries, but the data cannot be transferred to a central location due to governmental data sovereignty and jurisdiction laws [34]. Each participant here is a subsidiary with its national data center(s) and the coordinator might be located in a cloud platform or a global datacenter. Another example is a set of hospitals that want to collaborate to train a tumor detection model; each hospital is unable to trust the others and unwilling to trust and transfer data to a central service. Another example is a cloud-hosted machine learning service (e.g., Azure ML) that has multiple (competing) corporate clients which do not trust each other but have some level of trust in the cloud service to facilitate and secure federated learning.

14.3 Differentially Private Federated Learning

Differential privacy [12, 30] is a framework which deals with publishing the results of computations or queries made on a dataset in such a way that limits the disclosure of private information. In simple words, a computation on a dataset is called differentially private if an observer seeing its output cannot tell if a specific individual's information was used in the computation. ϵ-differential privacy, the typically used notion of differential privacy, is a mathematical definition for the privacy loss associated with any release of derivative information from a dataset.

Differential privacy has been recently used by a number of researchers during model training in a federated setting [1, 2, 32, 43]. A typical way to incorporate differential privacy into a federated learning setup is to add random noise in each of the model/data derivatives shared externally by a participant during the training process. The amount of random noise to be added depends on the level of privacy required by the participant. This noise addition adversely impacts the accuracy of the trained model. While there have been some studies that demonstrate that impact on accuracy can be minimized by adding a small amount of noise and carefully choosing the hyperparameters of the training [1, 43], there is no systematic study on how exactly the noise level impacts model convergence. This may require a long empirical process to determine achievable level of privacy without sacrificing too much accuracy. In this section, we describe this problem in more detail, with empirical evidence.

14.3.1 Background: Differential Privacy (DP)

Differential privacy literature has gained significant attention in the computer science field in the recent past. Here we briefly cover the differential privacy (DP) framework in the interest of keeping this chapter self-contained. For a detailed description of the differential privacy, we refer the reader to [12]. Differential privacy literature states that a (randomized) function $f(\cdot)$ is ϵ-differentially private if for all datasets X, X' that differ by only a single data item and all values of t

$$\left|\ln \frac{P(f(X)=t)}{P(f(X')=t}\right| \leq \epsilon \tag{14.1}$$

The parameter ϵ quantifies the privacy risk; lower ϵ means higher privacy. We work with a practical variant of the original differential privacy definition described in [12], called (ϵ, δ) differential privacy defined for the function $f(\cdot)$:

$$P(f(X)=t) \leq e^{\epsilon} P(f(X')=t) + \delta \tag{14.2}$$

This definition is interpreted as saying that $f(\cdot)$ is ϵ-differentially private with probability $1-\delta$. In order to achieve differential privacy, a noise term is added to the output of $f(\cdot)$ whose variance is dependent on the parameters ϵ and δ. Furthermore, it has been demonstrated that using an additive Gaussian noise term with 0 mean and variance

$$\sigma^2 = \Delta f \cdot \frac{2 \ln \frac{1.25}{\delta}}{\epsilon} \tag{14.3}$$

ensures (ϵ, δ)-differential privacy [12], where Δf is the sensitivity of $f(\cdot)$. Sensitivity is a measure of how much the computation can change when a single element of the underlying dataset changes. More formally, the sensitivity Δf is given by

$$\Delta f = \sup_{(X,X')} (\| f(X) - f(X') \|_l) \tag{14.4}$$

The above definitions deal with differential privacy of a single query, and however training a neural network requires many iterations, that is, many queries of the data. This means we cannot restrict ourselves to merely choosing a single value of ϵ but must consider the total privacy loss over the course of training. According to the composition theorem 3.16 in [12], using multiple (ϵ, δ)-DP will still be differentially private, but with a total privacy loss, referred to as the privacy budget (β), equal to the sum over all used ϵ. This means if we train for T iterations, using a constant value ϵ_0 at each iteration will take a total privacy budget of $\beta = \sum_{i=1}^{T} \epsilon_0 = T\epsilon_0$.

Of course, one's privacy budget does not have to be spent in this naive manner. Several authors have considered different strategies for spending privacy budgets with various improvements in mind. Shokri and Shmatikov [41] propose a way to limit the total number of queries, thus reducing overall privacy loss, and the work [35] uses a generalization of differential privacy to achieve tighter bounds on privacy loss per query.

14.3.2 Incorporating DP into SGD

Stochastic gradient descent (SGD) is one of the most popular optimization techniques used to train deep learning and various other machine learning models [5]. In distributed SGD, a mini-batch of samples is distributed over multiple learners who compute gradients of the model on their local share of the mini-batch and then share the gradients with a centralized parameter server. The parameter server computes the average gradient across the learners, updates the model parameters, and distributes the updated model parameters back to the learners.

In a typical federated learning setting, gradients of the model parameters are computed by the participants on their private dataset and shared with the centralized aggregator. Furthermore, concepts from differential privacy literature are utilized to modify SGD algorithm in order to prevent any leakage of information about the data through the shared gradients. This modification results in the so-called differentially private SGD [1], which is described below.

Let us now discuss how to use (ϵ, δ)-differential privacy in the context of federated learning using SGD. Here, $f(\cdot)$ corresponds to computing the gradient of the model parameters on the local dataset.

The update rule for mini-batch SGD with batch size S and learning rate η is

$$\theta_{k+1} = \theta_k - \eta \frac{1}{S} \sum_{i=1}^{S} g(x_i) \tag{14.5}$$

where $g(x_i)$ is the gradient of the loss function evaluated on data point $x_i \in X$. The quantity that is shared during gradient exchange is $f(x) = \eta \frac{1}{S} \sum_{i=1}^{S} g(x_i)$. Note, for the above formulation, the sensitivity would depend upon the gradient of only one data point which is different in the two datasets X and X', i.e.,

$$\begin{aligned}\Delta f &= \|f(X) - f(X')\|_l \\ &= \frac{\eta}{S} \left\| \sum_{i=1}^{S} (g(x_i) - g(x_i')) \right\|_l \\ &= \frac{\eta}{S} \|g(x_j) - g(x_j')\|_l \\ &\le \frac{\eta}{S} \cdot 2C\end{aligned}$$

where the constant C is an upper bound on the gradient. Hence, following the result from [12] above, the variance of the Gaussian noise term to be added to the gradients to make them (ϵ, δ)-differentially private is $2C \frac{\eta}{S} \frac{\ln \frac{1.25}{\delta}}{\epsilon}$. We describe the high-level algorithm for differentially private SGD [1] below.

Algorithm 14.1 (ϵ, δ)-differentially private SGD

```
Inputs: learning rate: $\eta$, batch size: S, clipping length: C, privacy budget: T$\epsilon_0$, initial weights: $\theta_0$
  for t = 0, 1, 2, 3, ..., T do
  Sample S data points uniformly at random

  compute $f(X) = \sum_{i=1}^{S} g(x_i)$

  clip gradient: $f(X) \leftarrow f(X)/\max(1, \frac{\|f(X)\|}{C})$

  sample $Z_k$ from the distribution $N(0, 2C \frac{\eta}{S} \frac{\ln \frac{1.25}{\delta}}{\epsilon})$
  $\theta_{t+1} \leftarrow \theta_t - \eta f(X) + Z_k$
  end for
```

14.3.3 Experiments and Discussion

Let us now discuss some experiments to understand the challenges in applying DP to federated learning in detail. Consider training two models on two different datasets—Resnet-18 model [19] on the CIFAR-10 [27] dataset and the Resnet-50 [19] model on the SVHN [37] dataset. All models are trained for 200 epochs using plain and DP versions of SGD. The learning rate schedule was to use 0.1 for the first 80 epochs, 0.01 for the next 40, and 0.001 for the remaining 80 epochs.

14.3.3.1 Accuracy vs ϵ

Figures 14.1 and 14.2 illustrate the convergence plots of different training runs for various values of batch size and ϵ. The very first observation is that the models converge to a lower accuracy as we decrease the value of ϵ. From Fig. 14.1, for a given batch size, say 1024, we observe that training schemes with lower noise added (corresponding to higher values of ϵ like 10, 1, and 0.1) have accuracy closer to the accuracy of non-private training. Once enough noise is added (corresponding to ϵ of 0.05 and lower), accuracy drops and does so precipitously for 0.01 and lower values

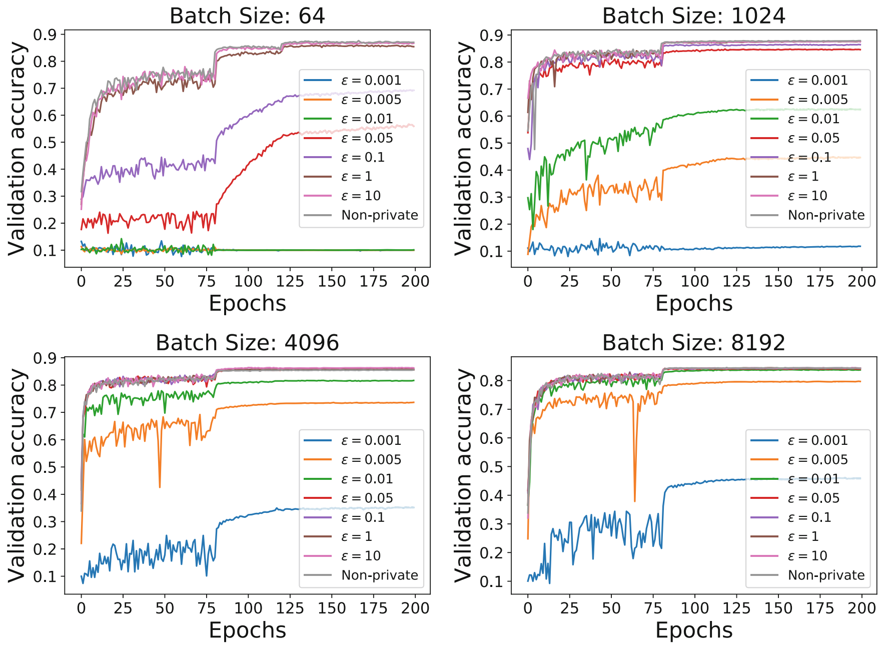

Fig. 14.1 Validation accuracy of Resnet18 on CIFAR10 per epoch for different batch sizes (64, 1K, 4K, and 8K) and ϵ (from 10 down to 0.001). Lower ϵ implies more noise added and hence more privacy

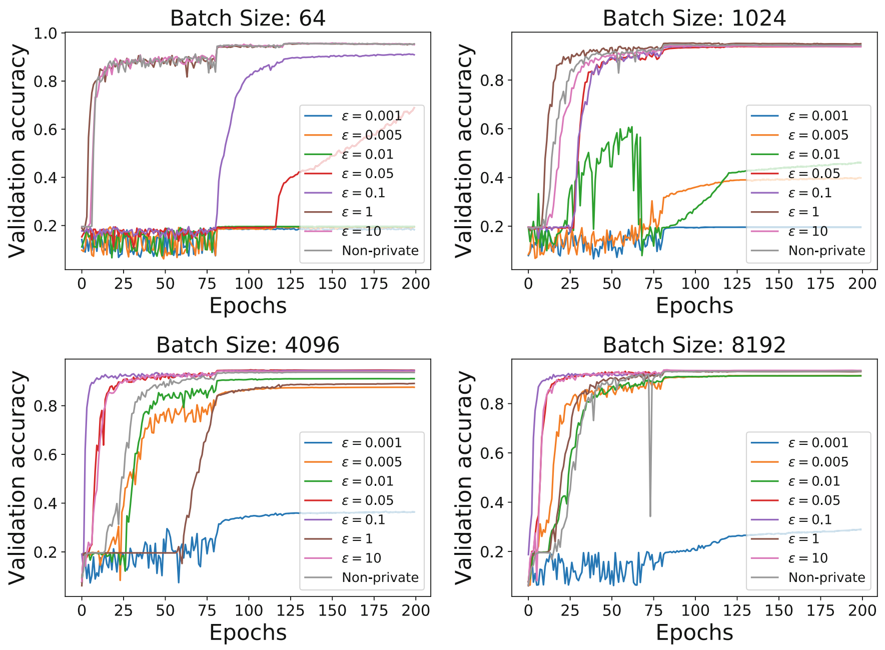

Fig. 14.2 Validation accuracy of Resnet18 on CIFAR10 per epoch for different batch sizes (64, 1K, 4K, and 8K) and ϵ (from 10 down to 0.001). Lower ϵ implies more noise added and hence more privacy

of ϵ. We see this phenomenon throughout Figs. 14.1 and 14.2 for all the models for a given batch size.

14.3.3.2 Accuracy vs Batch Size (Fixed ϵ)

An important observation from Figs. 14.1 and 14.2 is that even though the model accuracy levels for the non-private versions are pretty much unchanged for various values of batch sizes, the private version of training is highly sensitive to the batch size S. We have been able to achieve similar accuracy levels for private version of the models as those of the non-private versions even at very low values of ϵ by simply increasing the batch size. This can be prominently observed in the case of CIFAR10+Resnet18 in Fig. 14.1 for $\epsilon = 0.005$ and 0.01 (green and orange plots in Fig. 14.1, viewed left to right) but is true for all values of ϵ.

Quantitatively, for greater privacy corresponding to $\epsilon = 0.01$ and 0.005, final accuracy increases by approx. **37%** and **78.5%** by increasing batch size from 1024 to 8192, respectively, for CIFAR10+Resnet18. Similar trends can be observed from Fig. 14.2. Overall, while differential privacy is a promising approach, it does seem to require careful hyperparameter tuning to minimize impact on accuracy. This may involve spawning multiple FL jobs corresponding to different hyperparameters.

14.4 Additive Homomorphic Encryption

Research on secure and private federated learning and gradient descent is predominantly based either on (1) clever use of cryptography—homomorphic encryption and secure multi-party computation [3, 8, 10, 13, 16, 25, 26, 38] or on (2) modifying model parameters or gradients through the addition of statistical noise to get differential privacy [1, 2, 43]. Some techniques [45, 47] combine both.

Homomorphic encryption allows computation on ciphertexts, generating an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on the plaintext [18]. Fully homomorphic encryption is expensive, in terms of both encryption/decryption time and the size of the ciphertext. However, averaging gradient vectors in federated gradient descent requires only addition (division by the total number of participants can be done before or after encrypted aggregation). Hence, we can easily employ additive homomorphic encryption like the Paillier cryptosystem [22] to ensure privacy of gradients during federated training. The Paillier cryptosystem is an asymmetric algorithm for public key cryptography. Given only the public key and the encryption of m_1 and m_2, one can compute the encryption of $m_1 + m_2$. The definition of homomorphic encryption beautifully illustrates the challenge in applying it to private gradient aggregation. All participants have to encrypt their gradients/parameters with the *same* public key. This needs a trusted key generator/distributor. But, if all the participants send their Paillier encrypted gradients to this key generator/distributor, it can decrypt them and potentially reverse engineer the data. So, ideally, aggregation has to happen outside the key generator/distributor. Furthermore, the key generator/distributor has to be completely trusted to not leak the private key to any participant. Several designs are possible to satisfy these constraints. In this section, we describe one system—Mystiko [20] as an illustration—and compare it with differential privacy and secure multi-party computation (SMC).

We emphasize that Mystiko is one of many systems that use Paillier cryptography to secure aggregation. In [3, 38], the participants jointly generate a Paillier key pair and send the encrypted gradient vectors to the coordinator who is completely untrusted, except to add Paillier encrypted weights. The participants can then decrypt the aggregated gradient vectors. This, however, requires each participant to collaborate with the others to generate the Paillier keys and a high level of trust that participants do not collude with the untrusted coordinator to decrypt individual gradient vectors. One untrusted participant can leak the Paillier keys and potentially lead to privacy loss.

Secure multi-party computation (SMC) is a subfield of cryptography with the goal of creating methods for parties to jointly compute a function over their inputs while keeping those inputs private. Unlike traditional cryptographic tasks, where cryptography assures security and integrity of communication or storage and the adversary is outside the system of participants (an eavesdropper on the sender and receiver), the cryptography in this model protects participants' privacy from each other. SPDZ [10], and its variants (Overdrive [16, 25, 26]), optimizes classic SMC protocols. The advantage of such protocols is that they work with any number of honest+curious peers, do not change final accuracy of the trained model, and require a large number of colluding peers to break. The drawback, however, is efficiency—SMC protocols are computationally expensive (Sect. 14.6).

14.4.1 Participants, Learners, and Administrative Domains

Logically speaking, it is helpful to define federated learning algorithms in terms of *administrative domains*. An administrative domain is a *set of computing entities* (servers, VMs, desktops, laptops, etc.). Each entity inside an administrative domain trusts the other, malice is not a concern, and there are no legal/regulatory hurdles to sharing data and information derived from data. Note that an administrative domain does not necessarily mean a company or a non-corporate organization. It may be a project within a company handling confidential data; it may be not be located within a corporate datacenter, and instead be associated with an account on the public cloud. An organization can have multiple administrative domains. Each *participant* in a federated learning algorithm corresponds to an administrative domain. Computationally, the actual learning process (typically running on a GPU) performing the neural network training is called a *Learner*. Each learner works on (a batch of) data within the participant to compute the gradient vector.

14.4.2 Architecture

The main characteristics of any privacy preserving federated learning scheme revolve around (a) what methodology is used to encrypt the data (or noise addition) of the participants and (b) the communication protocol being used among the participants and the coordinator (if any) for aggregation of the model parameters or gradients. In MYSTIKO all participants encrypt individual data using a single Paillier public encryption key, adding encrypted gradient vectors and decrypting only the sum. Thus, only the participants are able to view their individual data, ensuring privacy. The question now is (1) how to distribute a common Paillier public key to all participants while keeping the corresponding private key secret? and (2) how to prevent anyone from decrypting individual weights? These are explained in the following sections.

For simplicity, we assume that there is exactly one learner per participant. We will relax this assumption in Sect. 14.4.4. MYSTIKO is typically deployed as a cloud service that mediates several participants. It involves a Job Manager, Membership Manager, a Key Generator, and a Decryptor. The Membership Manager is responsible for establishing the relationship between each participant and MYSTIKO and also keeping track of participants that belong to each federated learning job. The Job Manager manages an FL job through its lifecycle—it keeps track of participants, helps participants agree on hyperparameters, detects failures, and updates to memberships. While the focus of this chapter is attacks on the privacy of data from within a federation, traditional communication security is nevertheless essential to prevent outside attacks on the federation. For this, MYSTIKO and the participants (learners) agree to use a common public key infrastructure (PKI) [44]. The PKI helps ensure confidentiality of communications between the MYSTIKO components and the learners and also helps bootstrap the Paillier infrastructure. The PKI provides certification authorities (CAs), along with corresponding intermediate and Root CAs, creating a web of trust between the learners and MYSTIKO. MYSTIKO creates a bidirectional TLS channel [39] using the PKI for the security of control messages. The TLS channel is created using strong but ordinary (non-homomorphic) cryptographic algorithms (e.g., RSA for key exchange/agreement and authentication, AES for message confidentiality, and SHA for message authentication [39, 44]). The TLS channel is not used for gradient aggregation, but rather for all other communications, like registration of learners with the MYSTIKO topology formation, rank assignment, transmitting the Paillier public key to each learner, and transmitting the decrypted aggregated gradient vector to each learner.

14.4.3 MYSTIKO *Algorithms*

In this section, we describe MYSTIKO algorithms, starting with the simple ring-based algorithm, before adding parallelism and resiliency through broadcast and All-Reduce based communication.

14.4.3.1 Basic Ring-Based Algorithm

The basic ring-based aggregation algorithm is illustrated in Figs. 14.3 and 14.4. This algorithm operates across P participants, each in its own administrative domain and represented by a learner (L). The algorithm starts with each participant registering with MYSTIKO. MYSTIKO acts as the coordinator. The learners need not fully trust MYSTIKO; they only need to trust it to generate good encryption key pairs, keep the private key secret and follow the protocol.

MYSTIKO's Membership Manager starts the federated learning protocol once all expected learners are registered. The first step is to arrange the learners along a ring topology (Fig. 14.3). This can be done in several straightforward ways: (1) by location—minimizing geographic distance between participants, (2) by following a hierarchy based on the name of the participants (ascending or descending order), or

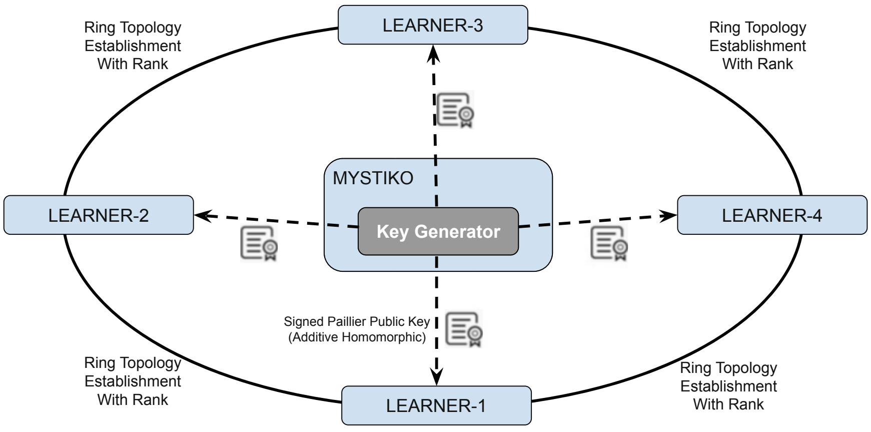
LEARNER-3
Ring Topology Establishment With Rank
Ring Topology Establishment With Rank
MYSTIKO
Key Generator
LEARNER-2
LEARNER-4
Signed Paillier Public Key (Additive Homomorphic)
Ring Topology Establishment With Rank
Ring Topology Establishment With Rank
LEARNER-1

Fig. 14.3 Topology establishment and key distribution

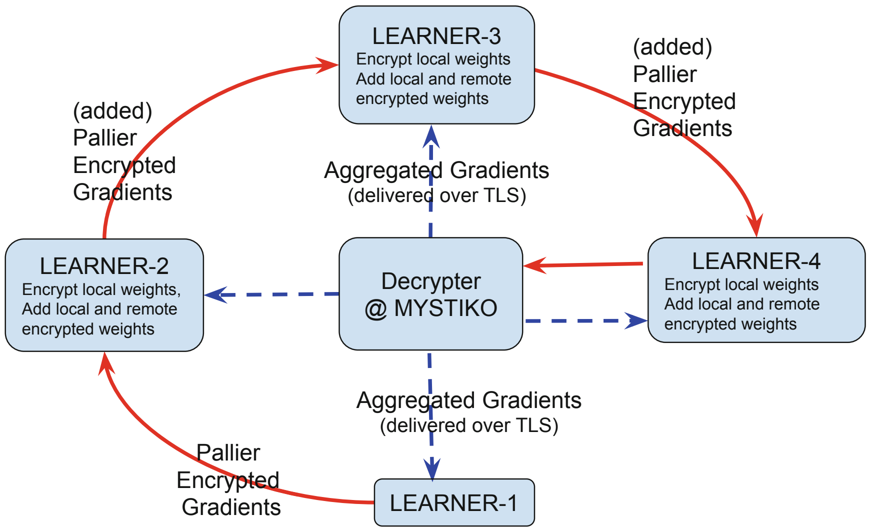
LEARNER-3
Encrypt local weights
Add local and remote encrypted weights
(added) Pallier Encrypted Gradients
(added) Pallier Encrypted Gradients
Aggregated Gradients (delivered over TLS)
LEARNER-2
Encrypt local weights, Add local and remote encrypted weights
Decrypter @ MYSTIKO
LEARNER-4
Encrypt local weights
Add local and remote encrypted weights
Aggregated Gradients (delivered over TLS)
Pallier Encrypted Gradients
LEARNER-1

Fig. 14.4 Basic aggregation protocol over a ring topology

(3) by using consistent hashing [24] on the name/identity of the participants. Once the learners have been arranged in a ring, each learner gets a rank (from 1 to P) based on its position in the ring.

MYSTIKO's Paillier Key Generator, which generates a Paillier public and private key pair for each federated learning job. Typically, a unique Paillier key pair is generated for each federated learning job and the Paillier public key securely distributed (over TLS) to all the learners. For long jobs, a separate key pair may be generated either once every epoch or once every h minutes (this is configurable). MYSTIKO's Decryptor is responsible for decrypting the Paillier encrypted aggregated gradient vector for distribution to the learners. Each learner receives a Paillier encrypted gradient vector from the previous learners on the ring, encrypts its own gradient vector with the Paillier public key, and adds (aggregates) the two Paillier encrypted vectors. This aggregated, Paillier encrypted gradient vector is then transmitted to the next learner on the ring. The last learner on the ring transmits the fully aggregated, encrypted gradient vector to MYSTIKO for decryption. MYSTIKO's Decryptor decrypts the aggregated vector and transmits the same securely over TLS to each of the learners.

Security Analysis Data never leaves a learner and by extension any administrative domain. This ensures privacy of data, provided each server inside the administrative domain has adequate defenses against intrusion. Unencrypted gradient vectors do not leave the learner. If there are P participants, in $P - 1$ cases, only aggregated Paillier encrypted gradient vectors leave the learner. Only MYSTIKO has the private key to decrypt these. For the first learner in the ring, the non-aggregated gradient vector is transmitted to the second learner, but it is Paillier encrypted and cannot be decrypted by the same. In fact, none of the participants are able to view even partially aggregated gradient vectors. After decryption, the aggregated gradient vector is distributed securely to the participants over TLS. Reverse engineering attacks, like the ones in [15] and [14], are intended to find the existence of a specific data record in a participant or to find data items that lead to specific changes in gradient vectors; both of which are extremely difficult when several gradient vectors computed from large datasets are averaged [21]. Decryption after averaging ensures the privacy of gradients. MYSTIKO only sees aggregated gradients and cannot get access to individual learner's data or gradients.

Colluding Participants The basic ring-based algorithm is resistant to collusion among $P - 2$ participants. That is, assuming an honest uncorrupted MYSTIKO deployment, it can be broken only if $P - 1$ learners collude. For learner L_i's gradients to be exposed, learners $L_1, L_2, \ldots, L_{i-1}$ and $L_{i+1}, \ldots, L_P$ should collude, i.e., all of them should simply pass on incoming encrypted gradient vector to the next learner, without adding any gradient of their own.

Fault Tolerance The disadvantage of a ring-based aggregation algorithm is that rings can break; for good performance, it is essential that the connectivity between each learner and MYSTIKO remains strong. Traditional failure detection techniques, based on heartbeats and estimation of typical round-trip times, may be used. Distributed synchronous gradient descent consists of a number of iterations, with gradients being averaged at the end of each iteration. If the failure of a learner is detected, the averaging of gradient vectors is paused until the learner is eliminated from the ring by the MYSTIKO's Membership Manager, or connection to the learner is established again. Pausing gradient averaging can also be done when connection to the MYSTIKO is temporarily lost.

14.4.3.2 Broadcast Algorithm

One of the main drawbacks of the ring-based algorithm is the establishment and maintenance of the ring topology. To mitigate this, an alternative is to use group membership and broadcast. Except for the establishment of the topology, the setting remains the same. Learners register with the MYSTIKO's Membership Manager, agree on a common PKI, and know the identity and number of participants. MYSTIKO generates a Paillier public–private key pair for each federated job and distributes the public key securely to each learner.

Each learner Paillier encrypts its gradient vector and broadcasts the encrypted vector to all other learners. Each learner, upon receipt of encrypted vectors from $P - 1$ learners, adds them and sends the Paillier encrypted sum to the MYSTIKO for decryption. After decryption, the aggregated gradient vector is transmitted securely to all learners over TLS. The broadcast algorithm is redundant and wasteful, as every learner computes the aggregate. But, with redundancy comes increased failure resiliency. With the ring, the failure of one participant can lead to partial loss of aggregated gradients, which is not the case for broadcast.

Colluding Participants The objective of breaking this algorithm is to determine the plaintext gradient vector of a specific LA. This algorithm is resistant to collusion and can be broken only if $P - 1$ participants collude, which is highly unlikely. Also, in the event that $P - 1$ participants collude to Paillier encrypt zero vectors instead of their actual gradient vectors, the broadcasted Paillier ciphertexts from all the $P - 1$ learners will be the same, which serves as a red flag enabling collusion detection. In fact, given that data is likely to be different at each participant, getting exactly the same Paillier encrypted gradient vector from even two learners is red flag.

14.4.3.3 All-Reduce

Ring-based All-Reduce [28] is essentially a parallel version of the ring-based aggregation protocol described in Sect. 14.4.3.1. It is illustrated in Fig. 14.5. The problem with the basic ring protocol in Sect. 14.4.3.1 is that each learner has to wait for its predecessor. However, in All-Reduce, the Paillier encrypted gradient vector is divided into P chunks where P is the number of participants. All learners then aggregate Paillier encrypted chunks in parallel. For example, in Fig. 14.5, there are three learners, and the gradient vectors are divided into three chunks each. Learner-2 does not wait for the entire vector of Learner-1 to be received. Instead, while it is receiving the first chunk of Learner-1, it transmits its own second chunk to Learner-3, which in parallel transmits its third chunk to Learner-3. In Step 2, Learner-2 transmits the partially aggregated chunk-1 to Learner-3, which transmits partially aggregated chunk-2 to Learner-1, which transmits the partially aggregated chunk-3 to Learner-2. At the end of Step-2, each learner has Paillier encrypted, aggregated chunks, which are transmitted to MYSTIKO's Decryptor for concatenation and decryption.

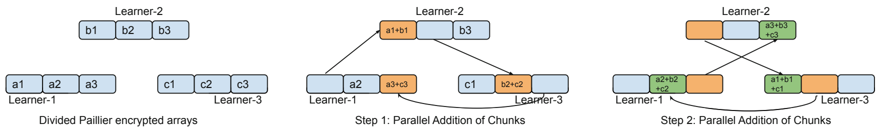
Learner-2
b1
b2
b3
a1
a2
a3
Learner-1
c1
c2
c3
Learner-3
Divided Paillier encrypted arrays
a1+b1
a3+c3
b2+c2
Step 1: Parallel Addition of Chunks
a3+b3 +c3
a2+b2 +c2
a1+b1 +c1
Step 2: Parallel Addition of Chunks

Fig. 14.5 MYSTIKO ring All-Reduce over Paillier encrypted arrays

Security Analysis We note that All-Reduce is the most efficient MYSTIKO protocol. With P learners, All-Reduce is essentially an instantiation of $P - 1$ rings (of Sect. 14.4.3.1), all operating in parallel. In Fig. 14.5, the first ring starts at the first chunk of Learner-1, the second ring starts at the second chunk of Learner-2, and the third starts at the third chunk of Learner-3. This implies that the security guarantees of All-Reduce are the same as that of the basic ring protocol.

14.4.4 Multiple Learners Per Administrative Domain

For presentation simplicity, we have assumed that there is exactly one learning process (learner) per participant. More realistically, within an administrative domain, data is partitioned among servers and multiple training processes (learners), which periodically synchronize their gradient vectors using an aggregator process local to the administrative domain. This is done for various reasons, including datasets being large, compute resources being cheap and available, and the desire to reduce training time. MYSTIKO's protocols can be applied in a straightforward manner to this case, with MYSTIKO's protocols running between local aggregators (LAs) instead of between learners. Local aggregation is not Paillier encrypted and non-private because compute resources within an administrative domain are trusted and can share even raw data. But aggregation between LAs follows MYSTIKO's protocols. This is illustrated in Fig. 14.6.

14.5 Trusted Execution Environments

A trusted execution environment (TEE) [40, 46] is a secure area of a main processor. TEEs are isolated execution environment that provide key security features such as isolated execution, integrity of applications executing with the TEE, along with confidentiality of their data assets. TEEs establish an isolated execution environment that runs in parallel with the standard operating system, such as Linux and Microsoft Windows; its aim is to defend sensitive code and data against privileged software attacks from a potentially compromised native OS. ARM TrustZone, IBM Hyperprotect, and Intel SGX are examples of TEE technologies, which use a combination of hardware and software mechanisms to protect sensitive assets. TEEs

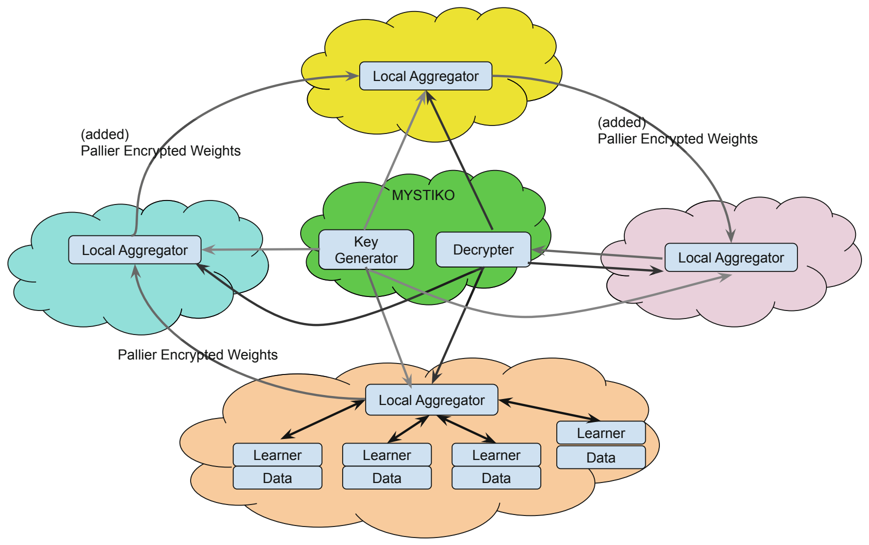

Fig. 14.6 Federating gradient descent

are often designed so that only trusted applications, whose integrity is verified at load time, can access the full power of the server's processors, peripherals, and memory. Hardware isolation provided by the TEE protects the applications inside it from other installed applications (including malware and viruses) on the host operating system. If multiple applications are contained within a TEE, software and cryptographic isolation often protect them from each other.

To prevent the simulation of TEEs with attacker- or user-controlled software on the server, TEEs involve a "hardware root of trust." This involves embedding a set of private keys directly into the TEE at the time of chip manufacturing, using one-time programmable memory. These cannot be changed, even after device resets or restarts. The public counterparts of these keys reside in a manufacturer database, together with a non-secret hash of a public key belonging to the trusted party (usually the chip vendor) which is used to sign trusted firmware alongside the circuits doing cryptographic operations and controlling access. The TEE hardware is designed in a way which prevents all software not signed by the trusted party's key from accessing the privileged features. The public key of the vendor is provided at runtime and hashed; this hash is then compared to the one embedded in the chip. If the hash matches, the public key is used to verify a digital signature of trusted vendor-controlled firmware (such as a chain of bootloaders on Android devices or 'architectural enclaves' in SGX). The trusted firmware is then used to implement remote attestation.

In this chapter, we describe how one system—TRUDA [7]—leverages Trusted Execution Environments (TEEs) to protect the model fusion process. Other examples include [6, 49] and [36]; each of which also has several optimizations. We restrict our treatment here to the basic aggregation process with TEEs using TRUDA as an example. TRUDA runs every aggregator within an encrypted virtual machine (EVM) via AMD Secure Encrypted Virtualization (SEV). All in-memory data are kept encrypted at runtime during model aggregation. To bootstrap trust between parties and aggregators, it uses a two-phase attestation protocol and develops a series of tools for integrating/automating confidential computing in FL. Each party can authenticate trustworthy aggregators before participating in FL training. End-to-end secure channels, from the parties to the EVMs, are established after attestation to protect model updates in transit.

14.5.1 Trustworthy Aggregation

TRUDA enforces cryptographic isolation for FL aggregation via SEV. The aggregators execute within EVMs. Each EVM's memory is protected with a distinct ephemeral VM Encryption Key (VEK). Therefore, TRUDA can protect the confidentiality of model aggregation from unauthorized users, e.g., system administrators, and privileged software running on the hosting servers. AMD provides attestation primitives for verifying the authenticity of individual SEV hardware/firmware. TRUDA employs a new attestation protocol upon the primitives to bootstrap trust between parties and aggregators in the distributed FL setting. This FL attestation protocol consists of two phases:

Phase 1: Launching Trustworthy Aggregators First, TRUDA securely launches SEV EVMs with aggregators running within. To establish the trust of EVMs, attestation must prove that (1) the platform is an authentic AMD SEV-enabled hardware providing the required security properties and (2) the Open Virtual Machine Firmware (OVMF) image to launch the EVM is not tampered. Once the remote attestation is completed, TRUDA provisions a secret, as a unique identifier of a trustworthy aggregator, to the EVM. The secret is injected into EVM's encrypted physical memory and used for aggregator authentication in Phase 2.

The EVM owner instructs the AMD Secure Processor (SP) to export the certificate chain from the Platform Diffie-Hellman (PDH) Public Key down to the AMD Root Key (ARK). This certificate chain can be verified by the AMD root certificates. The digest of OVMF image is also included in the attestation report along with the certificate chain.

The attestation report is sent to the attestation server, which is provisioned with the AMD root certificates. The attestation server verifies the certificate chain to authenticate the hardware platform and check the integrity of OVMF firmware. Thereafter, the attestation server generates a launch blob and a Guest Owner Diffie–Hellman Public Key (GODH) certificate. They are sent back to the SP on the aggregator's machine for negotiating a Transport Encryption Key (TEK) and a Transport Integrity Key (TIK) through Diffie–Hellman Key Exchange (DHKE) and launching the EVMs.

TRUDA retrieves the OVMF runtime measurement through the SP by pausing the EVM at launch time. It sends this measurement (along with the SEV API version and the EVM deployment policy) to the attestation server to prove the integrity of UEFI booting process. Only after verifying the measurement, the attestation server generates a packaged secret, which includes an ECDSA prime251v1 key. The hypervisor injects this secret into the EVM's physical memory space as a unique identifier of a trusted aggregator and continues the launching process.

Phase 2: Aggregator Authentication Parties participating in FL must ensure that they are interacting with trustworthy aggregators with runtime memory encryption protection. To enable aggregator authentication, in Phase 1, the attestation server provisions an ECDSA key as a secret during EVM deployment. This key is used for signing challenge requests and thus serves to identify a legitimate aggregator. Before participating in FL, a party first attests an aggregator by engaging in a challenge response protocol. The party sends a randomly generated nonce to the aggregator. The aggregator digitally signs the nonce using its corresponding ECDSA key and then returns the signed nonce to the requesting party. The party verifies that the nonce is signed with the corresponding ECDSA key. If the verification is successful, the party then proceeds to register with the aggregator to participate in FL. This process is repeated for all aggregators.

After registration, end-to-end secure channels can be established to protect communications between aggregators and parties for exchanging model updates. TRUDA enables TLS to support mutual authentication between a party and an aggregator. Thus, all model updates are protected both within EVMs and in transit.

14.6 Comparing HE- and TEE-Based Aggregation with SMC

In this section, we compare all the three Mystiko algorithms with a state-of-the-art protocol for secure multi-party computation (SPDZ [10, 11, 25]), and schemes for differential privacy (DP) through the addition of statistical noise. We employ a variety of image processing neural network models and datasets of various sizes: (1) 5-Layer CNN (small, 1MB) trained on MNIST dataset (60K handwritten digit images) (2) Resnet-18 (small–medium, 50MB) trained on the SVHN dataset (600K street digit images), (3) Resnet-50 (medium-sized model, 110 MB) trained on CIFAR-100 dataset (60K color images of 100 classes), and (4) VGG-16 (large model, 600MB) trained on Imagenet-1K dataset (14.2 million images of 1000 classes).

Experiments were conducted on a 40-machine cluster to evaluate all the algorithms on a varying number of participants from 2 to 40. No more than one participant was ever run on any machine, each of which was equipped with 8 Intel Xeon E5-4110 (2.10 GHz) cores, 64GB RAM, 1 NVIDIA V100 GPU, and a 10GbE network link. The machines were spread over four datacenters, and in every experiment, participants were uniformly distributed across datacenters. In every experiment, the dataset was *uniformly and randomly* partitioned across participants. Mystiko was executed on a dedicated machine in one datacenter. All data points henceforth are computed by averaging 10 experiment runs.

14.6.1 Comparing MYSTIKO *and SPDZ*

In federated learning, learners (or local aggregators) learn on local data for a specific number of iterations before federated gradient aggregation and model update. Privacy loss happens during gradient aggregation, which is where MYSTIKO and other systems like SPDZ intervene. Hence, we use the following two metrics to evaluate MYSTIKO and SPDZ: (1) total synchronization time, which measures the total time needed for privacy preserving gradient transformations (Paillier encryption in MYSTIKO, share generation in SPDZ, etc.) and the time required to communicate the transformed gradients to participants for federation and (2) communication time, which only measures communication time.

Figure 14.7 plots total synchronization time and communication time against the number of parties involved in federation, for all of our model/dataset combinations. From Fig. 14.7, we observe that All-Reduce is the most scalable of all the protocols, as the number of participants increases. This is mainly because it is a parallel protocol, where each learner/LA is constantly transmitting a small portion of the gradient array. The basic ring protocol is the least scalable because it is sequential.

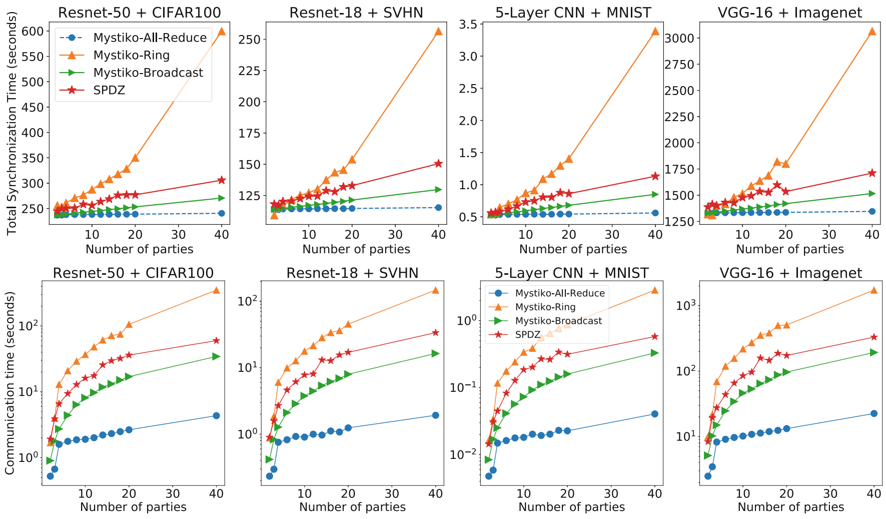

Fig. 14.7 MYSTIKO: total synchronization time (seconds) vs. number of parties (top plots) and total communication time (seconds) vs. number of parties (bottom). Recall that total synchronization time is the sum of the communication time and the gradient transformation time

Table 14.1 MYSTIKO: performance slowdown of Broadcast, Ring, and SPDZ relative to All-Reduce. Full trend available at Fig. 14.7

Communication time

	MYSTIKO			
# Parties	All-reduce	Broadcast	Ring	SPDZ
20	1	6.4–7.2×	38.2–39.9×	13.1–14.2×
40	1	7.9–8.6×	70.5–80.8×	13.8–14.7×

Total synchronization time

	MYSTIKO			
# Parties	All-reduce	Broadcast	Ring	SPDZ
20	1	12–51%	27–100%	2.2–6.1×
40	1	6–25%	15–59%	1.3–2.6×

Broadcast performs and scales better than the basic ring protocol because each participant is broadcasting without waiting for the others. SPDZ performs and scales worse than broadcast because its communication pattern is close to (but not exactly) a dual broadcast—each item of the gradient vector at each participant is split into secret shares and broadcast to the other participants; after secure aggregation, the results are broadcast back. MYSTIKO obviates the need for dual broadcast through the use of Paillier encryption and centralized decryption of aggregated gradients. Table 14.1 illustrates the performance impact of using other protocols for two cases (20 and 40 parties from Fig. 14.7).

However, the "enormous" speedups of using All-Reduce do not materialize when total synchronization time is considered. The scalability trends among the four protocols remain the same; the speedups in total synchronization time *remain significant* (as elucidated for two cases in Table 14.1). But the speedups are lower than the speedups due to communication. This demonstrates that the predominant overhead of private gradient descent in MYSTIKO and SPDZ vs. non-private gradient descent is gradient transformation prior to communication. From Fig. 14.7 and Table 14.1, we also observe that for small models (5-Layer CNN and Resnet-18), communication time plays a larger role. But for large models (Resnet-50 and VGG-16), gradient transformation plays a larger role.

Lastly, we observe that when compared to training time (illustrated using epoch time), synchronization time for private gradient descent is significantly larger than non-private gradient descent. This is primarily because training happens on V100 GPUs (with thousands of cores), while gradient transformation happens on CPUs. While there is a GPU accelerated version of fully homomorphic encryption ([9], which has worse performance than Paillier on CPUs), we are not aware of any GPU accelerated version of the Paillier algorithm.

14.6.2 Overheads of Using TEEs: AMD SEV

Compared to MYSTIKO and SPDZ, the biggest advantage is that the overheads are very low. The overhead of using TEEs comes from performing aggregation inside the EVMs. Overhead, measured as an increase in end-to-end latency, was between 2% and 4% per federated synchronization round [7]. And there was no difference in accuracy or convergence rate [7].

14.7 Concluding Remarks

In this chapter, we have examined various options for private parameter aggregation for federated learning. It is clear that each method has a unique set of advantages and disadvantages, and the search for a perfect solution is an active area of research. Differential privacy is very promising from an information leakage standpoint, but it (1) decreases the accuracy of the model and (2) involves nontrivial hyperparameter tuning (batch size, learning rate schedule) to obtain optimal results (or even near-optimal results). Hyperparameter tuning by trying different parameters may not be possible in federated settings due to the fact that all participants are not guaranteed to be available for extended time periods and also because running multiple experiments increases overall latency. Homomorphic encryption and secure multi-party computation do not alter accuracy or convergence rate and do not require hyperparameter tuning. But they incur high runtime overhead—using additive homomorphic encryption can minimize this to a great extent as illustrated by MYSTIKO, if the aggregation protocol is carefully designed. Finally, using TEEs has the potential to make aggregation overhead negligible without incurring any accuracy or convergence penalty, but it does require the use of specialized hardware.

References

1. Abadi M, Chu A, Goodfellow I, McMahan HB, Mironov I, Talwar K, Zhang L (2016) Deep learning with differential privacy. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, association for computing machinery, New York, NY, CCS '16, pp 308–318
2. Agarwal N, Suresh AT, Yu FXX, Kumar S, McMahan B (2018) cpSGD: Communication-efficient and differentially-private distributed SGD. In: NeurIPS 2018
3. Aono Y, Hayashi T, Trieu Phong L, Wang L (2016) Scalable and secure logistic regression via homomorphic encryption. In: Proceedings of the Sixth ACM conference on data and application security and privacy. Association for Computing Machinery, New York, NY, CODASPY '16, pp 142–144
4. Bonawitz K, Eichner H, Grieskamp W, Huba D, Ingerman A, Ivanov V, Kiddon C, Konečnỳ J, Mazzocchi S, McMahan HB et al (2019) Towards federated learning at scale: System design. Preprint. arXiv:190201046

5. Bottou L (1998) On-line learning and stochastic approximations. In: On-line learning in neural networks. Cambridge University Press, New York
6. Chen Y, Luo F, Li T, Xiang T, Liu Z, Li J (2020) A training-integrity privacy-preserving federated learning scheme with trusted execution environment. Inf Sci 522:69–79
7. Cheng P, Eykholt K, Gu Z, Jamjoom H, Jayaram KR, Valdez E, Verma A (2021) Separation of powers in federated learning. CoRR abs/2105.09400. https://arxiv.org/abs/2105.09400, http://2105.09400
8. Cramer R, Damgrd IB, Nielsen JB (2015) Secure multiparty computation and secret sharing, 1st edn. Cambridge University Press, Cambridge
9. Dai W, Sunar B (2016) cuHE: A homomorphic encryption accelerator library. In: Cryptography and information security in the Balkans. Springer International Publishing
10. Damgård I, Pastro V, Smart N, Zakarias S (2012) Multiparty computation from somewhat homomorphic encryption. In: Proceedings of the 32nd annual cryptology conference on advances in cryptology — CRYPTO 2012 - Volume 7417. Springer-Verlag, Berlin, Heidelberg, pp 643–662
11. Data61 C (2020) Multi-Protocol SPDZ. https://github.com/data61/MP-SPDZ
12. Dwork C, Roth A (2014) The algorithmic foundations of differential privacy. Found Trends Theor Comput Sci 9(3–4):211–407
13. Evans D, Kolesnikov V, Rosulek M (2018) A pragmatic introduction to secure multi-party computation. Found Trends Privacy Secur 2:70
14. Fredrikson M, Lantz E, Jha S, Lin S, Page D, Ristenpart T (2014) Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In: 23rd USENIX security symposium (USENIX Security 14). USENIX Association, San Diego, CA, pp 17–32. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/fredrikson_matthew
15. Fredrikson M, Jha S, Ristenpart T (2015) Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. Association for Computing Machinery, New York, NY, CCS '15, pp 1322–1333
16. Garg S, Sahai A (2012) Adaptively secure multi-party computation with dishonest majority. In: Safavi-Naini R, Canetti R (eds) Advances in Cryptology – CRYPTO 2012. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 105–123
17. Geiping J, Bauermeister H, Dröge H, Moeller M (2020) Inverting gradients–how easy is it to break privacy in federated learning? Preprint. arXiv:200314053
18. Gentry C (2010) Computing arbitrary functions of encrypted data. Commun ACM 53(3):97–105
19. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR), pp 770–778
20. Jayaram KR, Verma A, Verma A, Thomas G, Sutcher-Shepard C (2020) Mystiko: Cloud-mediated, private, federated gradient descent. In: 2020 IEEE 13th international conference on cloud computing (CLOUD). IEEE Computer Society, Los Alamitos, CA, pp 201–210
21. Jayaraman B, Evans D (2019) Evaluating differentially private machine learning in practice. In: 28th USENIX Security Symposium (USENIX Security 19), USENIX Association, Santa Clara, CA, pp 1895–1912. https://www.usenix.org/conference/usenixsecurity19/presentation/jayaraman
22. Jost C, Lam H, Maximov A, Smeets BJM (2015) Encryption performance improvements of the paillier cryptosystem. IACR Cryptology ePrint Archive. https://eprint.iacr.org/2015/864
23. Kairouz P, McMahan HB, Avent B, Bellet A, Bennis M, Bhagoji AN, Bonawitz K, Charles Z, Cormode G, Cummings R et al (2019) Advances and open problems in federated learning. Preprint. arXiv:191204977
24. Karger D, Lehman E, Leighton T, Panigrahy R, Levine M, Lewin D (1997) Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the twenty-ninth annual ACM symposium on theory of computing. Association for Computing Machinery, New York, NY, STOC '97, pp 654–663

25. Keller M, Yanai A (2018) Efficient maliciously secure multiparty computation for ram. In: EUROCRYPT (3). Springer, pp 91–124. https://doi.org/10.1007/978-3-319-78372-7_4
26. Keller M, Pastro V, Rotaru D (2018) Overdrive: Making SPDZ great again. In: Nielsen JB, Rijmen V (eds) Advances in cryptology – EUROCRYPT 2018. Springer International Publishing, Cham, pp 158–189
27. Krizhevsky A (2009) Learning multiple layers of features from tiny images. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf, https://www.cs.toronto.edu/~kriz/cifar.html
28. Kumar V, Grama A, Gupta A, Karypis G (1994) Introduction to parallel computing: design and analysis of algorithms. Benjamin-Cummings Publishing, California
29. Lecun Y, Bengio Y, Hinton G (2015) Deep learning. Nat Cell Biol 521(7553):436–444
30. Lee J, Clifton C (2011) How much is enough? choosing ϵ for differential privacy. In: Lai X, Zhou J, Li H (eds) Information security. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 325–340
31. Marr B (2018) 27 Incredible examples of AI and machine learning in practice. Forbes Magazine
32. McMahan HB, Andrew G (2018) A general approach to adding differential privacy to iterative training procedures. CoRR abs/1812.06210. http://arxiv.org/abs/1812.06210
33. Melis L, Song C, De Cristofaro E, Shmatikov V (2019) Exploiting unintended feature leakage in collaborative learning. In: 2019 IEEE symposium on security and privacy. IEEE, pp 691–706
34. Millard C (2013) Cloud computing law. Oxford University Press
35. Mironov I (2017) Rényi differential privacy. In: 2017 IEEE 30th computer security foundations symposium (CSF), pp 263–275
36. Mo F, Haddadi H, Katevas K, Marin E, Perino D, Kourtellis N (2021) PPFL: Privacy-preserving federated learning with trusted execution environments. In: Proceedings of the 19th annual international conference on mobile systems, applications, and services. Association for Computing Machinery, New York, NY, MobiSys ’21, pp 94–108
37. Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading digits in natural images with unsupervised feature learning NIPS workshop on deep learning and unsupervised feature learning 2011
38. Phong LT, Aono Y, Hayashi T, Wang L, Moriai S (2018) Privacy-preserving deep learning via additively homomorphic encryption. Trans Inf Forens Secur 13(5):1333–1345
39. Rescorla E (2018) The transport layer security (TLS) protocol version 1.3. RFC 8446
40. Sabt M, Achemlal M, Bouabdallah A (2015) Trusted execution environment: What it is, and what it is not. In: 2015 IEEE Trustcom/BigDataSE/ISPA, vol 1, pp 57–64. https://doi.org/10.1109/Trustcom.2015.357
41. Shokri R, Shmatikov V (2015) Privacy-preserving deep learning. In: ACM CCS ’15
42. Shokri R, Stronati M, Song C, Shmatikov V (2017) Membership inference attacks against machine learning models. In: 2017 IEEE symposium on security and privacy (SP), pp 3–18
43. Song S, Chaudhuri K, Sarwate A (2013) Stochastic gradient descent with differentially private updates. In: 2013 IEEE global conference on signal and information processing, GlobalSIP 2013 - Proceedings, 2013 IEEE global conference on signal and information processing, GlobalSIP 2013 - Proceedings, pp 245–248
44. Stallings W (2013) Cryptography and network security: principles and practice, 6th edn. Prentice Hall Press, Upper Saddle River
45. Truex S, Baracaldo N, Anwar A, Steinke T, Ludwig H, Zhang R, Zhou Y (2019) A hybrid approach to privacy-preserving federated learning. In: Proceedings of the 12th ACM workshop on artificial intelligence and security. Association for Computing Machinery, New York, NY, AISec’19, pp 1–11
46. Volos S, Vaswani K, Bruno R (2018) Graviton: Trusted execution environments on GPUs. In: 13th USENIX symposium on operating systems design and implementation (OSDI 18). USENIX Association, Carlsbad, CA, pp 681–696. https://www.usenix.org/conference/osdi18/presentation/volos
47. Xu R, Baracaldo N, Zhou Y, Anwar A, Ludwig H (2019) Hybridalpha: An efficient approach for privacy-preserving federated learning. In: Proceedings of the 12th ACM workshop on

artificial intelligence and security. Association for Computing Machinery, New York, NY, AISec'19, pp 13–23. https://doi.org/10.1145/3338501.3357371

48. Yin H, Mallya A, Vahdat A, Alvarez JM, Kautz J, Molchanov P (2021) See through gradients: Image batch recovery via GradInversion. Preprint. arXiv:210407586
49. Zhang X, Li F, Zhang Z, Li Q, Wang C, Wu J (2020) Enabling execution assurance of federated learning at untrusted participants. In: IEEE INFOCOM 2020 - IEEE conference on computer communications, pp 1877–1886
50. Zhao B, Mopuri KR, Bilen H (2020) iDLG: Improved deep leakage from gradients. Preprint. arXiv:200102610
51. Zhu L, Liu Z, Han S (2019) Deep leakage from gradients. In: Advances in neural information processing systems, pp 14774–14784