SMPAI: Secure Multi-Party Computation for Federated Learning

Vaikkunth Mugunthan[1,3]
vaik@mit.edu

Antigoni Polychroniadou[3]
antigoni.o.polychroniadou@jpmchase.com

David Byrd[2,3]
db@gatech.edu

Tucker Hybinette Balch[3]
tucker.balch@jpmchase.com

[1]Massachusetts Institute of Technology
[2]Georgia Institute of Technology
[3]J.P. Morgan AI Research

Abstract

Federated Learning is a technique that enables a large number of users to jointly learn a shared machine learning model, managed by a centralized server, while the training data remains on user devices. That said, federated learning reduces data privacy risks.

Privacy concerns still exist since it is possible to leak information about the training data set from the trained model's weights or parameters. Most federated learning systems use the technique of differential privacy to add noise to the weights so that it is harder to reverse-engineer the individual data sets. Differential privacy reduces the risk but does not eliminate leakage from the data. The combination of differential privacy and cryprogrpahy can eliminate leakage.

As opposed to prior works, we propose a new mechanism that protects against a wide range of attacks. Our mechanism is based on advanced cryptographic techniques, in particular, secure multiparty computation and differential privacy.

Our model has been developed and tested on the ABIDES environment simulating mobile device networks.

1 Introduction

Modern institutions routinely need to conduct analysis of large data sets stored across multiple servers or devices. A typical response is to combine those data sets into a single central database, but this approach introduces a number of privacy challenges: The institution may not have appropriate authority or permission to transfer locally stored information, the owner of the data may not *want*it shared, and centralization of the data worsens the potential consequences of a data breach. For example, the mobile app ai.type collected personal data from its users' phones and uploaded this information to a central database. Security researchers gained access to the database and obtained the names, email addresses, passwords, and other sensitive information of 31 million users of the Android version of the app. Such incidents highlight the risks and challenges associated with centralized data solutions.

One approach to mitigate the above concerns is to analyze the multiple data sets separately and share only the resulting insights from each analysis. This approach is realized in a recently-introduced technique called federated analysis. Federated *learning*, already adopted by large companies like

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Google, allows users to share insights (perhaps the parameters of a trained model) from the data on their laptops or mobile devices *without* ever sharing the data itself. For instance, email providers could use federated learning to reduce the amount of spam their customers receive. Instead of each provider using its own spam filter trained from its customers' reported spam email, the providers could combine their models to create a shared spam-detection mechanism, without sharing their individual customers' reported spam emails. It is still possible, however, for a malicious party to potentially compromise the privacy of the individual users by inferring details of a training data set from the trained model's weights or parameters. It is important to protect sensitive user information while still providing highly accurate inferences.

Simply anonymizing data is no longer enough to guarantee the privacy of individuals whose information has been collected, due to the increasing prevalence of database reconstruction attacks and re-identification from correlated data sets. Differential privacy [12] can help prevent such reverse-engineering by adding noise to the input data set, to intermediate calculations, or to the outputs. Then, even if the input data set is reconstructed, it is *not* the exact data of any user. More formally, differential privacy is a mathematical concept that guarantees statistical indistinguishability for individual inputs by perturbing values. The use of differentially-private machine learning algorithms in centralized settings is widely discussed in the literature, and the technique has been adopted by major companies; for example, Apple uses it in web search auto-completion. The application of differential privacy adds a layer of randomness so that adversaries with additional information still have uncertainty over the original value. There is an obvious trade-off: adding randomness to the collected data preserves user privacy at the cost of accuracy. Proper application of differential privacy ensures that meaningful insights can still be derived from the aggregated data.

One solution to the above problem, which guarantees privacy without compromising accuracy, is secure multi-party computation (MPC)[14, 6, 9, 2]. Using Secure MPC, multiple parties collaborate to compute a common function of interest without revealing their private inputs to other parties. An MPC protocol is considered *secure* if the parties learn only the final result, and no other information. For example, a group of employees might want to compute their average salary without any employee revealing their individual salary to any other employee. This task can be completed using MPC, such that the only information revealed is the result of the computation (i.e. the average salary). Tying these approaches together, we arrive at secure federated learning, which can be achieved as follows: after locally training a model on their individual data, users send the weights (parameters) of their model to the server using an encryption scheme which still allows the server to perform computations on the encrypted data; specifically, the server can compute a weighted average of all the encrypted weights received from users, but cannot discover the original weights for any user. A recent line of investigation has constructed secure federated learning using techniques from MPC.

Because MPC reveals information only about the final result, and not any of the inputs, a solution based on MPC seems ideal. However, in some scenarios, just the result can be enough to reveal information about the inputs. For example, in the case of employees computing their average salary, once the result is known, if all but one of the employees work together, they can easily determine the salary of the final employee. By applying differential privacy on top of MPC, we can construct a system that protects from even this type of extreme collusion attack. If we add noise to each input, the final calculation will still be accurate within known bounds, but we will eliminate the possible leakage of any inputs from the output. In the previous solution which used *only* differential privacy, the server would know the "noisy" private weights of each user. In the solution which combines MPC and differential privacy, the noisy weights sent to the server are also encrypted such that the server can only calculate the result, and cannot infer anything about even the noisy weights of any particular user. The system is thus now fully private.

While there have been previous constructions which combine distributed differential privacy [13, 8, 10, 3] and MPC, we take a different approach. Roughly, in prior works each client chooses his/her local differential private noise. Contrarily, we offer models where the differentially private noise for each party is unknown to the party and is generated in a distributed manner from other parties. We will see in the next section why this approach achieves stronger security guarantees than previous works.

2 Our Approach

In federated learning, a machine learning model is learnt in an iterative way, repeating four steps: 1) the server chooses a set of users to compute an updated model; 2) each client computes a local trained model update its local data; 3) the updates are sent to the server; and 4) the server aggregates these local updates to construct a global model. In this paper, we consider logistic regression. In particular, the client's update includes the weights of the logistic regression. The server receives the weights from all the clients at each iteration and computes the averaged model.

Eliminate weight leakage: In order to hide the weights from the server, we use secure MPC techniques, where each client sends his/her input to the server in an encrypted way. In particular, our secure weighted average protocol running across n clients is based on the protocol of [4]. Henceforth, all operations are performed modulo some bound p. In the setup phase, every pair of clients C_i and C_j will share some common randomness $r_{ij} = r_{ji}$ which is secret and known only to these two clients. Then at each iteration of the logistic regression, client C_i sends its weights masked with these common random strings, while adding all r_{ij} for $j > i$ and subtracting all r_{ik} for $k < i$. That is, C_i sends to S the following message for his/her data x_i:

$$\bar{w_i} := (w_i + \sum_{j=i+1}^{n} r_{ij} - \sum_{k=1}^{i-1} r_{ki}) \bmod p$$

In order to establish common randomness between each pair of clients, each pair of clients just run Diffie-Hellman Key agreement protocol [7] (see Section 4 for more details)

A 3-party example is given in Figure 1. Note that the weights are encrypted via the use of the common randomness. The values $\bar{w}$ reveal nothing about the weights w. Also in our protocol there is a way to reuse the common randomness for all the logistic regression iterations. That said, clients only need to communicate once at the beginning of the protocol. In the next iterations they only communicate with the server.

Eliminate weighted average leakage: In our protocol the weights are completely hidden from the server, nothing about them is being revealed. However, the output of the computation, i.e., the weighted average can reveal some information about the weights and subsequently the local data sets. Imagine a scenario where $n - 1$ out of the n clients collude. Then, based on the revealed weighted average, the $n - 1$ clients can learn the weight of the single non-colluding client. We would like to avoid such a leakage. To this end, we can apply differential privacy in the MPC protocol. That said, each client will send to the server 'noisy' encrypted weights. More specifically, each client chooses a local noise term and add the noise to his/her weights and then encrypt them. Now if $n - 1$ clients collude they will only be able to see the differential private weights of the single honest client, instead of the plaintext weights. This is the type of leakage that prior works allowed. For large values of the privacy loss parameter in differential privacy, $epsilon$, the accuracy is pretty high and hence the probability of inferring the original value increases.

However, we introduce a new differentially-private mechanism that reduces this type of leakage. We offer an additional layer of randomness by introducing distributed differentially privacy to the weights of the local models. In prior works, each client picks his/her local noise. Contrarily, we offer models which generate differentially private weights, where the differentially private noise for each party is generated in a distributed manner from other parties. More specifically, each client receives his/her noise from the other clients. We use encryption to ensure that clients do not know the partial noise values received from neighbouring parties. Hence, every honest participant is assured a privacy guarantee stronger than the existing differential privacy guarantee offered by existing implementations even if clients collude. In the worst case, when $n - 1$ parties collude they still cannot figure out the differentially private input of the other party as the honest party would have contributed to the noise of the other $n - 1$ dishonest parties (and thus the colluding parties cannot subtract it). In the case where each party generates its own differentailly private noise, the dishonest colluding parties can try to replicate the Laplace distribution for the noise generated and subtract it from the differentially private output set. Hence, having a better way to guess the original input of the honest party. This can be prevented in our case. Therefore, making it harder for the colluding adversaries. In particular, in our scheme, each client sends two encrypted noisy terms to the other clients but the clients choose to add to their weights only 1 out of the 2 terms received. In this way, once parties collude they

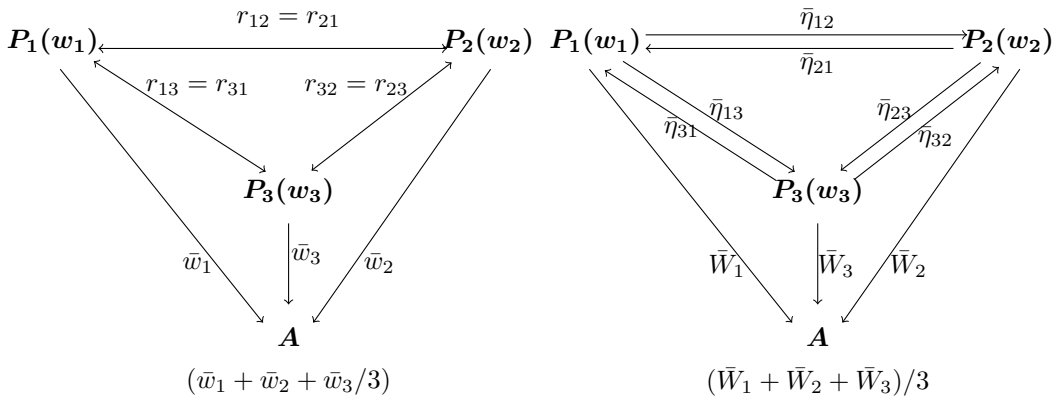

Figure 1: Secure 3-party weighted average protocol where $\bar{w}_1 = w_1 + r_{12} + r_{13}, \bar{w}_2 = w_2 - r_{21} + r_{23}, \bar{w}_3 = w_3 - r_{31} - r_{32}$.

cannot subtract a significant number of noise terms because they will simply do not know which noise term the honest clients chose. For example, in Figure 1 client P_1 receives encrypted noise terms $\bar{\eta}_{21} = (\bar{\eta}^0_{21}, \bar{\eta}^1_{21})$ and $\bar{\eta}_{31} = (\bar{\eta}^0_{31}, \bar{\eta}^1_{31})$ and it decides to add only 1-out-of-2 of them. For example, client P_1 will add to his encrypted weights $\bar{x}_1$, the noise term $\bar{\eta}^1_{2,1}$ and $\bar{\eta}^0_{3,1}$. The final weight sent to the server is $\overline{W}_1 = \bar{w}_1 + \bar{\eta}^1_{2,1} + \bar{\eta}^0_{3,1}$.

Our new differentially-private mechanisms: We device two different mechanisms to generate the distributed noise of each client.

Mechanism I: In this mechanism we make sure that each client's weights are differentially private. To achieve this mechanism, each client receives $n-1$ encrypted noise contributions from the other $n-1$ clients drawn from the difference of two gamma distributions. Then, each client adds the $n-1$ noise contributions to his/her weights. Note that the sum of the difference of gamma distributions is a Laplace distribution. The Laplace distribution achieves differential privacy. That said, the weights are converted to differentially private weights before they reach the server.

Mechanism II: In the second case, we offer a partial differentially private noise generation, whose total sum is differentially private. Thus only the final result added by the server is differentially private. To achieve this mechanism, each client receives an encrypted noise contributions from its neighbouring client drawn from a gamma distribution. Then, each client computes the difference of the received noise with a noise of his choice from the gamma distribution. Note that the difference of gamma distributions does not offer differential privacy. That said, the noisy weight sent to the server are not differentially private. We are the first to propose a distributed differentially private noise utilization in a multi-party setting for federated learning to reduce leakage.

Note that in both mechanisms the client chooses 1-out-of-2 encrypted noise contributions received from the other parties.

The second method has better utility than the first method since less noise is being added to the final weighted average. On the other hand, the first approach may leak less information, if $n-1$ parties collude since the weights sent to the server are already differential private, as opposed to the second mechanism in which they are not differential private.

Comparison of the two different mechanisms For the purposes of this introduction, we trained a toy simple logistic classifier using Andrew Ng's dataset [1] from his lecture on logistic regression in coursera. The dataset consists of two exam results of microchips in a factory and using the test results one will predict if the microchips are going to be rejected or admitted. The two axes in Fig. 2 depict the exam results (Exam1 and Exam 2). The green boundary represents the non-private classifier, the black boundary depicts the private classifier trained using the first mechanism, where each party adds differerntially private noise to his weights, and the yellow boundary represents the private classifier trained using the second mechanism.

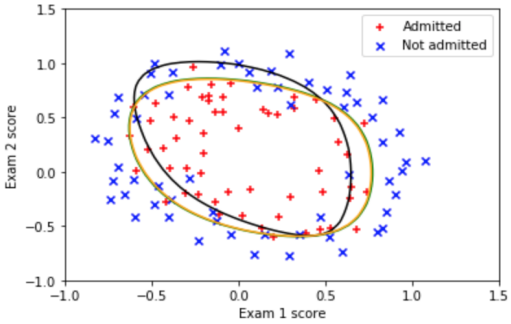

Figure 2: Accuracy comparison of our differerntially private mechanisms with non private classifier

In the next section we provide the formal definitions required to formally define our protocol in Section 4.

2.1 Differential Privacy

Definition 1. (ϵ-differential privacy) A randomized mechanism $\mathcal{A}$: $\mathcal{D} \to \mathcal{O}$ preserves ϵ-differential privacy (ϵ-DP) when there exists $\epsilon > 0$ such that,

$$\Pr\left[\mathcal{A}(D_1) \in \mathcal{T}\right] \leq e^{\epsilon} \Pr\left[\mathcal{A}(D_2) \in \mathcal{T}\right]$$

holds for every subset $\mathcal{T} \subseteq \mathcal{O}$ and for any two neighboring datasets $\mathcal{D}_1 \sim \mathcal{D}_2$.

Definition 2. (Global Sensitivity) For a real-valued query function $q : \mathcal{D} \to \mathbb{R}$, where $\mathcal{D}$ denotes the set of all possible datasets, the global sensitivity of q, denoted by Δ, is defined as

$$\Delta = \max_{\mathcal{D}_1 \sim \mathcal{D}_2} |q(\mathcal{D}_1) - q(\mathcal{D}_2)|,$$

for all $\mathcal{D}_1 \in \mathcal{D}$ and $\mathcal{D}_2 \in \mathcal{D}$.

2.1.1 Laplacian Mechanism

One of the most well-known differentially private mechanism is the Laplacian mechanism, which uses random noise X drawn from the symmetric Laplacian distribution. The zero-mean Laplacian distribution has a symmetric probability density function $f(x)$ with a scale parameter λ defined as:

$$f(x) = \frac{1}{2\lambda} e^{-\frac{|x|}{\lambda}}.$$

Given the global sensitivity, Δ, of the query function q, and the privacy parameter ϵ, the *Laplacian mechanism* $\mathcal{A}$ uses random noise X drawn from the Laplacian distribution with scale $\lambda = \frac{\Delta}{\epsilon}$. The Laplacian mechanism preserves ϵ-differential privacy.

2.2 Generating Laplace Random Variable from Gamma Random Variables

A Laplace random variable can be generated from the sum of n random variables as follows,

$$L(\mu, \lambda) = \frac{\mu}{n} + \sum_{k=1}^{n} \gamma_k - \gamma'_k$$

γ_k and γ'_k are Gamma distributed random variables with probability density functions defined as :

$$\frac{(1/s)^{1/n}}{\Gamma(1/n)} x^{1/n-1} e^{-x/s},$$

where $1/n$ is the shape parameter, s is the scale parameter and $\Gamma(k) = \int_0^\infty x^{k-1} e^{-x} dx$

2.3 Training Local Logistic Regression Classifiers

Logistic regression is a machine learning algorithm used to solve the problem of binary linear classification.

Let Parties $P_1, ..., P_n$ have datasets $DS_1, ..., DS_n$ where $DS_i = (x^{(i)}, y^{(i)})$ contains a set of instances $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_m^{(i)})$, where m is the number of features, and their corresponding labels $y^{(i)}$, for i in 1 to n.

Every party P_i makes use of their dataset $(x^{(i)}, y^{(i)})$ to learn an l_2 regularized logistic classifier with weights w'_{ij}. The weights are obtained by solving the following optimization problem

$$w'_{ij} = \arg\min_w Cost(w) = \arg\min_w \frac{1}{t_i} \sum_{k=1}^{t_i} log(1 + e^{-y_k^{(i)} f(x_k^{(i)})}) + \alpha w^T w, \tag{1}$$

where $f(x_k^{(i)}) = w^T x_k^{(i)}$, t_i is the number of training examples of P_i and $\alpha > 0$ is the regularization parameter.

In order to minimize the cost function, we make use of gradient descent, an iterative optimization algorithm. For a dataset DS_i, the optimal w is calculated iteratively as $w^{p+1} \leftarrow w^p - \beta \nabla Cost(w^p), p \geq 0$, where β is the learning rate, w^0 is assigned a random value, and $\nabla Cost$ is the gradient.

2.4 Differentially Private Federated Logistic Regression using Output Perturbation

Privacy-preserving federated learning allows large number of parties to learn a model while keeping their local training data private. Parties first train local models on their local data and coordinate with a server to obtain a global model. Given n parties, let $\hat{w_i}$, for $i \in 1$ to n, represent the local model estimator after minimizing the objective function.

$\hat{W} = \frac{1}{n} \sum_{i=1}^{n} \hat{w_i} + \eta$, η is the differentially private noise added to the cumulative model.

According to [10], for 1-$Lipschitz$ the global sensitivity for a multi-party setting $\frac{2}{n*k*\alpha}$, where k is the size of the smallest dataset amongst the k parties, α is the regularization parameter. Hence, $\eta = Laplace(\frac{2}{n*k*\alpha*\epsilon})$, where ϵ is the privacy loss parameter.

In our protocol, client will add noise to the weights of the trained model.

3 Experiments

In order to evaluate our method, we implement it in ABIDES, an agent-based interactive discrete event simulation framework. [5] The ABIDES platform was originally deployed for financial market simulation, but at its core provides a framework easily adapted to other domains. The system operates in a single-threaded manner to permit deterministic re-simulation in the presence of stochastic elements, but simulates the actions of tens of thousands of agents operating in parallel to one another. The simulation Kernel tracks time in nanoseconds and enforces "simulation physics" including configurable agent computation delays and pairwise communication latency among agents. All inter-agent communication passes through the Kernel in the form of timestamped messages in a priority queue. The nature of discrete event simulation permits efficient computation of sparse activity patterns at high time resolution. The ABIDES source code is freely available under a BSD-style license at `https://github.com/abides-sim/abides`. Using ABIDES we can simulate the latency across the different devices. Many prior works, including [4], on federated learning calculate the running time of their protocol ignoring the latency.

In Tables 1 and 2 we provide some preliminary results on the performance of our system. We used the CIFAR-10 dataset [11] to evaluate our experiments. Each party selects 6000 rows from the dataset, sets the learning rate to 0.5, sets the regularization parameter to 0.001, and number of iterations to train the model per round to 1000. The non private version gave us an accuracy of 89.1%. We compare the accuracy in Table 1 for different values of the privacy loss parameter, epsilon, and number of parties collaborating in the system. We clearly see that accuracy is proportional to number of parties and epsilon. $\mathsf{Accuracy_I}$ corresponds to mechanism I and $\mathsf{Accuracy_{II}}$ corresponds to mechanism II in which only the final output from the server is differentially private.

Table 1: Accuracy comparison of mechanisms I and II on the CIFAR-10 dataset

Epsilon	Number of Parties	$\mathsf{Accuracy_{II}}$	$\mathsf{Accuracy_{I}}$
0.01	150	58.1	36.57
0.1	50	63.65	40.05
0.1	100	89.05	47.45
0.1	150	89.09	54.8
0.1	100	89.09	60.0

Table 2: Running time Comparison

Number of Parties	Time taken for NYC	Time taken for Globe (sec)
50	34	211
100	35	217
150	46	226

Our evaluation computes the total time taken to receive updated mean weights from the server for 3 iterations. For communication within New York City clients, we simulated latencies by generating random numbers using the uniform distribution with a lower bound of 0.001 seconds and an upper bound of 0.003 seconds. For communication anywhere across the globe, we simulated latencies by generating random numbers using the uniform distribution with a lower bound of 0.01 seconds and an upper bound of 2 seconds. The comparison can be seen in Table 2. The experiments were implemented in a MacBook with 8GB of 1600MHz on board memory and an intel i5 core processor.

4 Secure Weighted Average Protocol

In this section we formally describe our weighted average protocol Π_{PPFL}, depicted in Protocol 1, for secure logistic regression performed by a set of clients $(P_1, \ldots, P_n)$ and a server S.

In the setup phase, every pair of parties P_i and P_j will share some common randomness $r_{i,j} = r_{j,i}$ which is secret and known only to these two parties. In the online weighted average phase, client P_i sends its weights masked with these common random strings, while adding all $r_{i,j}$ for $j > i$ and subtracting all $r_{i,k}$ for $k < i$.

In order to establish common randomness between each pair of parties, the two parties just run Diffie-Hellman Key agreement protocol [7]. Before proceeding with a formal description of the protocol, we first enumerate the cryptographic primitives we use:

- We assume the existence of an algorithm $\mathcal{G}(1^\lambda)$, where λ is the security parameter, that outputs a representation of a cyclic group $\mathbb{G}$ of order q (with $||q|| = \lambda$) for which the discrete logarithm problem is believed to be hard. Recall that a group $\mathbb{G}$ is cyclic if there exists a generator g such that $\{g^0, g^1, \ldots, g^{q-1}\} = \mathbb{G}$. Moreover, the discrete logarithm problem is believed to be hard if for every probabilistic polynomial time adversary A, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:

$$\Pr_{x \leftarrow \mathbb{Z}_q}[A(\mathbb{G}, g, q, g^x) = x] = \mathsf{negl}(\lambda)$$

- A key derivation function $H : \mathbb{G} \to \{0,1\}^\lambda$. It is assumed that if h is distributed uniformly in $\mathbb{G}$, then $H(h)$ is distributed uniformly in $\{0,1\}^\lambda$.
- A pseudorandom generator with double expansion, i.e., $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$. It is assumed that for every distinguisher D there exists a negligible function $\mathsf{negl}(\cdot)$ such that:

$$\left| \Pr_{s \leftarrow \{0,1\}^\lambda}[D(G(s)) = 1] - \Pr_{r \leftarrow \{0,1\}^{2\lambda}}[D(r) = 1] \right| = \mathsf{negl}(\lambda) \ .$$

We are now ready to provide the detailed description of the weighted average protocol:

Protocol 1 Privacy-Preserving Federated Logistic Regression Protocol Π_{PPFL} for a single iteration

The protocol Π_{PPFL} runs with parties $P_1, \ldots, P_n$ and a server S. It proceeds as follows:
Inputs: For $i \in [n]$, party P_i holds input dataset D_i.
Public Parameters: $(\mathbb{G}, g, q)$ generated by $\mathcal{G}(1^\lambda)$ and modulo p.
$\boldsymbol{\Pi_{\mathsf{PPFL}}.\mathsf{Setup}(1^\lambda)}$:

Round 1: Each party P_i for $i \in [n]$ proceeds as follows:

- Choose n secrets $a_{i,1}, \ldots, a_{i,n}$ uniformly and independently at random from $\mathbb{Z}_q$ and computes $(pk_{i,1}, \ldots, pk_{i,n}) = (g^{a_{i,1}} \bmod p, \ldots, g^{a_{i,n}} \bmod p)$.
- Generate gamma random variables $\gamma^b_{i,j}(1/n, scale)$ and $\bar{\gamma}^b_{i,j}(1/n, scale)$, for $b \in \{0,1\}$ with $scale = 2/(n * len(DS_i) * \alpha * \epsilon)$.
- For all $j \in [n]$:
  (a) Generate random masks $s_{i,j} \in \mathbb{Z}_q$.
  (b) Compute noise $\eta^0_{i,j} = s_{i,j} + \gamma^0_{i,j} - \bar{\gamma}^0_{i,j}$.
  (c) Compute noise $\eta^1_{i,j} = s_{i,j} + \gamma^1_{i,j} - \bar{\gamma}^1_{i,j}$.
- Each party P_i sends $pk_{i,j}$ and $\eta^0_{i,j}, \eta^1_{i,j}$ to party P_j.

Round 2: Each party P_j for $j \in [n]$ proceeds as follows:

- Upon receiving all values $(pk_{1,j}, \ldots, pk_{n,j})$, compute the shared common keys $r_{i,j}$ for all $i \in [n]$ as follows:
  (a) Using the secret $a_{j,i}$ compute $c_{i,j} = c_{j,i} = (pk_{i,j})^{a_{j,i}} = (g^{a_{i,j}})^{a_{j,i}} \bmod p$.
  (b) Let $c_{1,j}, \ldots, c_{n,j}$ be the set of all common keys. Use a key-derivation function and set $r_{i,j} = r_{j,i} = H(c_{i,j})$

Given the above setup, we can compute the federated logistic regression model as follows:
$\boldsymbol{\Pi_{\mathsf{PPFL}}.\mathsf{WeightedAverage}(D_i, \{r_{i,j}\}_{j\in[n]})}$:

Round 1: Each party P_i proceeds as follows:

- Compute the weights W_i, using Equation (1), of the local logistic classifier obtained by implementing regularized logistic regression on input D_i.
  The next steps are repeated per weight. Without loss of generality we describe the algorithm for a single weight, denoted by w_i.
- Generate a random bit vector $b = (b_1, \ldots, b_n)$ and compute

$$y_i := w_i + \sum_{j=i+1}^{n} r_{i,j} + \sum_{k=1}^{i-1} r_{k,i} + \sum_{j=1}^{n} \eta^{b_i}_{i,j} \bmod p\,.$$

- Send y_i to the server.

Round 2: The server computes $W = \big(\sum_{i=1}^{n} y_i \bmod p\big)/n$ and sends W to all parties.

$\boldsymbol{\Pi_{\mathsf{PPFL}}.Output(1^\lambda, W)}$: Each party P_i upon receiving W runs the next iteration of the logistic regression repeating $\Pi_{\mathsf{PPFL}}.WeightedAverage(1^\lambda)$.

The above protocol is described for a single iteration of the logistic regression. To perform the next iteration the algorithm $\Pi_{\mathsf{PPFL}}.\mathsf{Setup}$ is not repeated. Instead, the parties can use the common keys $r_{i,j}$ to generate different common keys for the next iteration. More specifically, in the first iteration of the logistic regression we use a pseudorandom generator $(r'_{i,j}, s) = G(r_{i,j})$ and update the common randomness $r_{i,j} := r'_{i,j}$. For the next iterations, run $G(s)$ to obtain the new $r'_{i,j}$ and the seed for the next iteration and so on. That said, the parties need to communicate with each other only at the beginning of the training. Moreover, given an upper bound on the number of iterations all the noise terms η can be exchanged in the $\Pi_{\mathsf{PPFL}}.\mathsf{Setup}$. Furthermore, logistic regression is performed over real numbers, in our implementation we use fix point arithmetic representation to perform the operation modulo p.

References

[1] Logistic Regression Dataset, howpublished = `https://github.com/ankitraj7217/andrew-ng-week-3-logistic-regression/blob/master/ex2data2.txt`, note = Accessed: 2019-09-16.

[2] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10, 1988.

[3] Vincent Bindschaedler, Shantanu Rane, Alejandro E Brito, Vanishree Rao, and Ersin Uzun. Achieving differential privacy in secure multiparty data aggregation protocols on star networks. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 115–125. ACM, 2017.

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

[5] David Byrd, Maria Hybinette, and Tucker Hybinette Balch. ABIDES: towards high-fidelity market simulation for AI research. *CoRR*, abs/1904.12066, 2019.

[6] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract). In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, page 462, 1987.

[7] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[8] 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*, pages 486–503. Springer, 2006.

[9] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.

[10] Bargav Jayaraman, Lingxiao Wang, David Evans, and Quanquan Gu. Distributed learning without distress: Privacy-preserving empirical risk minimization. In *Advances in Neural Information Processing Systems*, pages 6343–6354, 2018.

[11] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

[12] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *FOCS*, volume 7, pages 94–103, 2007.

[13] Elaine Shi, T-H Hubert Chan, Eleanor Rieffel, and Dawn Song. Distributed private data analysis: Lower bounds and practical constructions. *ACM Transactions on Algorithms (TALG)*, 13(4):50, 2017.

[14] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167, 1986.