Collusion Resistant Federated Learning with Oblivious Distributed Differential Privacy

David Byrd
Bowdoin College
United States
d.byrd@bowdoin.edu

Vaikkunth Mugunthan
MIT
United States
vaik@mit.edu

Antigoni Polychroniadou
antigoni.poly@jpmorgan.com
J.P. Morgan AI Research
United States

Tucker Balch
tucker.balch@jpmchase.com
J.P. Morgan AI Research
United States

ABSTRACT

Federated learning enables a population of distributed clients to jointly train a shared machine learning model with the assistance of a central server. The finance community has shown interest in its potential to allow inter-firm and cross-silo collaborative models for problems of common interest (e.g. fraud detection), even when customer data use is heavily regulated. Prior works on federated learning have employed cryptographic techniques to keep individual client model parameters private even when the central server is not trusted. However, there is an important gap in the literature: efficient protection against attacks in which other parties collude to expose an honest client's model parameters, and therefore potentially protected customer data.

We aim to close this collusion gap by presenting an efficient mechanism based on oblivious distributed differential privacy that is the first to protect against such client collusion, including the "Sybil" attack in which a server generates or selects compromised client devices to gain additional information. We leverage this novel privacy mechanism to construct an improved secure federated learning protocol and prove the security of that protocol. We conclude with empirical analysis of the protocol's execution speed, learning accuracy, and privacy performance on two data sets within a realistic simulation of 5,000 distributed network clients.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; • **Computing methodologies → Multi-agent systems**; **Discrete-event simulation**.

KEYWORDS

multi-agent, privacy, collusion, machine learning, federated

ACM Reference Format:
David Byrd, Vaikkunth Mugunthan, Antigoni Polychroniadou, and Tucker Balch. 2022. Collusion Resistant Federated Learning with Oblivious Distributed Differential Privacy. In *3rd ACM International Conference on AI in Finance (ICAIF '22), November 2–4, 2022, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3533271.3561754

1 INTRODUCTION

Many advances in financial machine learning require training models from data sets originally distributed across different devices, information silos, or even firms. Although the details vary by application, centralizing sensitive or valuable data away from its original location carries potentially significant consequences: loss of competitive advantage from inter-firm sharing, regulatory risk from cross-silo sharing, or customer information breaches from device data harvesting.

1.1 Privacy Preserving Federated Learning

Federated learning is a recent technique that addresses these concerns by training on the sensitive data of each customer, silo, or firm separately, without removing the data from its original location. Only the resulting model parameters (e.g. trained neural network weights) are transmitted and combined to produce a collaborative model. [1, 13] In this way, the data from each customer, silo, or firm can contribute to the solution of a collective problem such as fraud, credit origination, or trade execution, without exposing any of the source data. A typical approach is: The central server selects some clients (e.g. firms, silos, or devices) to train an improved model on their own local data, starting from the most recent shared model as a baseline. Each client sends its trained local model weights to the server, which computes an average-weight shared model. The new shared model is sent to all clients, and the process repeats.

1.2 Limitations of Prior Works

Some prior works have found that individual customer privacy can still be compromised by using the trained model to infer certain details of the training data set. [15, 16]. This problem has been alleviated with the incorporation to federated learning of *differential privacy*, which randomly perturbs values to guarantee statistical indistinguishability for individual inputs, and *secure multi-party computation* (MPC), which allows parties to collaboratively compute a common function of interest without revealing their private inputs. [9, 11] Prior works in privacy-preserving federated learning combine these techniques to provide strong protection against undesired inference by the server. [1, 12] However, collusion among enough clients can reveal the "noisy" parameters of an honest client, and the scale of added noise is limited by the need for an accurate shared model.

We primarily build on a recent line of research in the financial machine learning community that combined differential privacy and MPC to produce a computationally-efficient secure federated learning protocol. [3] Byrd et al. empirically demonstrated the efficiency and efficacy of their protocol using a version of an an open source agent-based simulation called ABIDES (https://github.com/abides-sim/abides) which they customized for federated learning. [2] In the non-collusion case where an untrustworthy server attempted to infer some client's true model parameters, they found that the irreversible added encryption values were eight orders of magnitude greater than any client's model weights, meaning no useful information could be recovered. However if there was collusion among the other clients, an honest client's individual model weights could be recovered with an error of about ±50%. In both cases, shared model inference accuracy remained similar to unsecured federated learning. The key limitations of this prior work were poor security against collusion among the client population and a lack of security proofs.

1.3 Contributions

We propose a novel, efficient mechanism that protects against any attempt to undermine differential privacy by collusion of $n-1$ out of n total clients. Unlike prior works, we offer a protocol where the differential privacy noise for each party is added in an *oblivious* way, such that no party can know how much noise it added to the calculation, and therefore cannot later assist in removing that noise during a potential collusion event. This greatly reduces the information that colluding clients can recover about the individual trained model parameters of an honest client. We provide both proofs and empirical evidence to support this claim.

We acknowledge that obliviousness *can* be achieved by running the noise generation algorithm inside the MPC, but such solutions are based on heavy cryptography machinery involving a significant amount of public key operations or incur increased communication complexity and are thus not efficient. [4, 12] In this work we focus on the concretely efficient aggregation protocol of Bonawitz et al. without drop-out parties which does not involve any public key operations in the learning phase. [1] We therefore provide the first practical protection against $n-1$ attacks by constructing an efficient oblivious distributed differentially private aggregation protocol.

2 BACKGROUND

In this section, we provide a brief, formal description of components critical to our approach. The first two sections of Byrd et al. and the third section of Bonawitz et al. contain more expansive foundational information for those new to secure federated learning. [1, 3]

2.1 Secure Multiparty Computation

Consider a group of mutually distrustful parties who need to compute a function jointly over their inputs, while maintaining the privacy of those inputs and ensuring the correctness of the result. If there is some trusted third party, the other parties can simply send their inputs to it for computation and communication of the results. Secure Multi-Party Computation (MPC) allows the trusted third party to be replaced with a secure interactive protocol: If n parties $P_1, \ldots, P_n$ hold private inputs $x_1, \ldots, x_n$ and wish to compute some arbitrary function $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, where the output of P_i is y_i, each party P_i can learn exactly y_i, and nothing else. [11]

As a simple example, if twenty banks wish to compute their aggregate customer count, they could each send their individual value to a central server to perform the sum and relay only the answer. The server will know each bank's true customer count. If the server is not trusted, each pair of banks could instead share a very large arbitrary number beforehand. If one bank from each pair adds this number, and the other bank subtracts this number, before sending their individual customer counts to the server, the final answer will not change, but the server can no longer determine anything about any bank's individual customer count.

2.2 Differential Privacy

If two databases that differ by only one element are statistically indistinguishable from each other, they can be said to uphold differential privacy. As a practical matter in financial machine learning, this could mean that one cannot infer from a trained model whether a particular customer record was present in the training data. In this work we use the Laplacian mechanism which preserves ϵ-differential privacy [9].

Definition 1. (ϵ-differential privacy [10]) A randomized mechanism $\mathcal{A}$ preserves ϵ-differential privacy (ϵ-DP) if for any two neighboring datasets D_1, D_2 that differ by one element, and for all subsets of possible answers $\mathcal{S} \subseteq Range(\mathcal{A})$, $\Pr[\mathcal{A}(D_1) \in \mathcal{S}] \leq e^{\epsilon} \Pr[\mathcal{A}(D_2) \in \mathcal{S}]$.

2.3 Federated Logistic Regression Classifiers

If one of n parties, called P_i, has a local observation set with m features $X^i_{1\ldots m}$ and matching binary labels y^i, it can use logistic regression to learn a boolean classifier with weights w_i by using gradient descent to iteratively optimize:

$$w_i = \arg\min_{w} \frac{1}{t_i} \sum_{k=1}^{t_i} log(1 + e^{-y_k^{(i)} f(x_k^{(i)})}) \tag{1}$$

where $f(x_k^{(i)}) = w^T x_k^{(i)}$ and P_i has t_i observations. If all n parties perform this local optimization and transmit their trained weights w_i to a central server, a federated logistic classifier can be obtained as:

$$W = \frac{1}{n} \sum_{i=1}^{n} w_i + \eta \tag{2}$$

where η is the sum of any noise added to attain differential privacy. We use a vector-based logistic regression implementation based on the method of Byrd et al. [3]

2.4 Network Topology & Threat Model

We model a common star network topology with a central server that is connected to all clients. The protocols that we describe and compare against are secure in the semi-honest model. A semi-honest adversary (whether client or server) follows the protocol correctly but tries to learn as much as possible about the inputs of other parties from the messages it receives. If there are multiple semi-honest corruptions, we allow the adversary to combine the views of the corrupted parties to potentially learn more information.

3 APPROACH

Our approach combines secure multi-party aggregation (Section 2.1) with oblivious distributed differential privacy (Section 2.2) to better secure federated learning against $n-1$ collusion attacks. We employ logistic regression (Section 2.3) as the local learning method, and each client update includes the weights of that logistic regression. The server receives the weights from all clients at each iteration and computes a new global model using the average of the client updates for each weight. In light of the literature demonstrating that private client data can be inferred from the trained model weights, the general task is to secure each client's locally trained model weights against discovery while still learning an accurate shared model. [15, 16] We acknowledge again that the collusion problem can be solved using generic MPC, but such generic solutions are impractical due to computational inefficiency. Our contribution is a practical and efficient solution to this problem using lightweight cryptographic tools.

3.1 Underlying Approach

We follow the existing methods of Byrd et al. (cited work's Sections 3.1 and 3.2) to eliminate per-weight and weighted average leakage during protocol execution. [3] We briefly summarize those methods here:

For each pair of clients c_i, c_j, there is a shared secret s, and the pair will report their locally trained model weights w_m^c (model parameter m for client c) as $w_m^i + s$ and $w_m^j - s$, respectively. Clients are able to locally generate subsequent shared secrets, so the pairwise client communication process occurs only once at the start of the protocol. The sum, and therefore mean, of the reported weights is unchanged from the true value, but the server cannot reconstruct any client's model weight.

Since the final shared value of each model weight must be exactly correct, clients could work together to remove their local weights and reveal the correct local weight of another client. To prevent this, each client additionally perturbs each reported weight with a random number drawn from a Laplace distribution with scale inversely proportional to the ϵ privacy loss parameter. Now only the "noisy" local weight of a client can be recovered, with the significant limitation that too much noise will cause the shared model to fail to converge.

3.2 Our Extended Approach

We introduce a novel and efficient oblivious distributed differentially private mechanism. Where in prior works, each client selects its own local noise, and could therefore remove it from the final model during a collusion attack, we offer a protocol where the noise for each party is added in an *oblivious* way. For each weight, each client receives a tuple of encrypted noise terms from each other client and adds only a subset of them. The party selecting noise terms to add does not know how much of the noise is "real" and how much is MPC encryption that will cancel out in the final model. Thus, a party P does not know the cleartext noise added to its own weight and the other parties do not know which noise term is chosen by P.

We now show that the information leakage on an honest client's weights after the collusion attack is smaller than previous approaches. Our task is to enable the parties to calculate the sum of their inputs (i.e., $W = \sum_{i=1}^{n} w_i$), while ensuring privacy for an honest party in the presence of a collusion attack given W. In prior works if $n-1$ parties collude, they subtract their weights and noise terms from W and then the final noise remaining in the transmitted weight of the honest party w_h is a single value chosen by the honest party. In our case, if $n-1$ parties collaborate then the final noise remaining in the transmitted weight of the honest party w_h is $n-1$ times larger, because the corrupted parties cannot subtract their noise terms.

At a high level, in our scheme, each client sends two encrypted noisy terms (permuted and randomized by the server) per model weight to the other clients, but each receiving client chooses only one of the two to add to each weight. Thus even if parties collude they cannot subtract a significant number of noise terms since they do not know which noise terms the honest client chose. For a protocol of n parties, we offer security against any $n-1$ semi-honest corruptions or collusion of the server with $n-1$ semi-honest corruptions.

4 SECURE WEIGHTED AVERAGE PROTOCOL

4.1 Our Protocol

In this section, we formally describe the weighted average protocol Π_{PPFL}, which results from our extensions to prior works, for federated logistic regression secure against $n-1$ collusion as performed by a set of clients $(P_1, \ldots, P_n)$ and a server S. The protocol is also concisely listed as Protocol 1 at the top of this page.

During setup, every pair of parties P_i and P_j will share some common randomness $r_{i,j} = r_{j,i}$. In the online weighted average phase, client P_i sends its weights masked with these common random strings, adding all $r_{i,j}$ for $j > i$ and subtracting all $r_{i,k}$ for $k < i$. That is, P_i sends to server S the following message for its data w_i: $y_i := (w_i + \sum_{j=i+1}^{n} r_{ij} - \sum_{k=1}^{i-1} r_{ki}) \bmod p$. Each weight received by the server is masked by $n-1$ large random numbers r, so it cannot accurately reconstruct any client's true model weight w_i. Because the total randomness applied to the weights sums to zero once the server computes W in round 2 of Π_{PPFL}, the averaged final model W will be identical to one calculated without security.

To establish common randomness r, each pair of parties run the standard Diffie-Hellman Key exchange protocol from the literature communicating via the server. [7]

The protocol is given for a single iteration of federated logistic regression. To avoid re-running the Diffie-Hellman key exchange in subsequent iterations, we follow the approach of Bonawitz et al. [1] To summarize, in the first iteration we use a pseudorandom generator $(r'_{i,j}, s) = PRG(r_{i,j})$ to both update the common randomness $r_{i,j} := r'_{i,j}$ and obtain a new random seed s. For subsequent iterations, instead of executing Π_{PPFL}.Setup, parties can run $PRG(s)$ to obtain a new $r'_{i,j}$ and the seed for the next iteration and so on. Thus

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

The protocol Π_{PPFL} runs with parties $P_1, \dots, P_n$ and a server S. It proceeds as follows:
Inputs: For $i \in [n]$, party P_i holds input dataset D_i.
Π_{PPFL}.Setup(1^λ): Each party P_i for $i \in [n]$ proceeds as follows for all $j \in [n]$ $(i \neq j)$:

- Generate random variables $\gamma^b_{i,j}$ and $\bar{\gamma}^b_{i,j}$ for $b \in \{0, 1\}$ from the gamma $\mathcal{G}(1/n, scale)$ distribution with $scale = 2/(n * len(D_i) * \alpha * \epsilon)$.
- Generate random masks $s_{i,j} \in \mathbb{Z}_q$.
- Compute masked noises $\eta^0_{i,j} = s_{i,j} + \gamma^0_{i,j} - \bar{\gamma}^0_{i,j}$ and $\eta^1_{i,j} = s_{i,j} + \gamma^1_{i,j} - \bar{\gamma}^1_{i,j}$.
- Run the Diffie-Hellman key Exchange protocol Π_{DH} to obtain a common shared key $r_{i,j}$ with each party P_j which is only known between parties P_i and P_j ($r_{i,j}$ is not known to S).
- Each party P_i sends $\eta^0_{i,j}, \eta^1_{i,j}$ to S who permutes and randomizes them and forwards to party P_j.

Given the above setup, we can compute the federated logistic regression model as follows:
Π_{PPFL}.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.
  We describe the algorithm for a single weight, denoted by w_i:
- Generate a random bit vector $b = (b_1, \dots, b_n)$ and send y_i to the server S: $y_i := w_i + \sum_{j=i+1}^{n} r_{i,j} - \sum_{k=1}^{i-1} r_{k,i} + \sum_{j=1}^{n} \eta^{b_j}_{j,i} - \sum_{j=1}^{n} s_{i,j} \mod p$.

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

Π_{PPFL}.*Output*($1^\lambda, W$): Each party P_i upon receiving W repeats WeightedAverage for the next weight or iteration of the logistic regression with locally updated common keys $r'_{i,j}$.

the parties need to run the exchange only once at the onset of the training.

We generate the Laplacian noise in a distributed way by the use of gamma distributions $\mathcal{G}$ given that the Laplace distribution $\mathcal{L}$ can be constructed as the sum of differences of i.i.d. gamma distributions. To run machine learning algorithms and the DP mechanism which computes on rational values, we use field elements in a finite field $\mathbb{Z}_q$ to represent the *fixed-point values*. Concretely, for a fixed-point value $\bar{x}$ with k bits in the integer part and f bits in the decimal part, we use the field element $x := 2^f \cdot \bar{x} \mod q$ in $\mathbb{Z}_q$ to represent it.

4.2 Security of our Protocol

We prove that our protocol protects the privacy of honest users in the semi-honest setting given the topology in Section 2.4. Following the standard idea-real paradigm, we will show that when executing the protocol with a set of parties $\mathcal{U}$ of size n with threshold $t < n$, the joint view of the server S and any set of less than t users does not leak any information about the other users' inputs except what can be inferred from the output of the computation.

The view of a party P_i consists of its internal state (including its input w_i and randomness) and all messages this party received from other parties. The messages *sent* by this party do not need to be part of the view because they can be determined using the other elements of its view. Given any subset C of corrupted parties out of the n parties in $\mathcal{U}$, let $\mathsf{REAL}^{\mathcal{U},t,k}_C(w_1, \dots, w_n)$ be a random variable representing the combined views of all parties in C in the above protocol execution, where the randomness is over the internal randomness of all parties. We will show that there exists an efficient simulator that, for every choice of the honest clients' inputs, outputs a simulation of the adversarial participants' view of the protocol run whose distribution is computationally indistinguishable from the distribution of the adversaries' view of the real protocol run.

The following theorem shows that the joint view of any subset of less than t users and the server can be simulated without knowing the secret input of any other users. In other words, the adversary controlling less than t users cannot learn anything other than the output of the computation. In our protocol we consider the leakage L learned from the difference of the noise terms η^0, η^1. Note that this leakage does not affect the error function given later in Definition 2.

Theorem 1 (Semi-Honest Security, against t clients). *For security parameter λ, an n-party protocol for an aggregation function f and all leakage L of size $O(n)$ is L-secure if there exists a PPT simulator* SIM *such that for all $k, t, \mathcal{U}$ where $t < |\mathcal{U}|$, all corrupted parties $C \subseteq \mathcal{U}$ and all $w_{\mathcal{U}}$, which denote all secret inputs of all users in all iterations k, letting $n = |\mathcal{U}|$, then the output of* SIM *is computationally indistinguishable from the output of* $\mathsf{REAL}^{\mathcal{U},t,k}$:

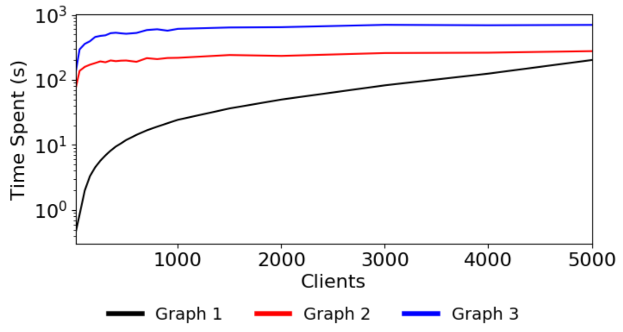
$$\mathsf{REAL}^{\mathcal{U},t,k}_C(\lambda, w_{\mathcal{U}}, L) \approx_c \mathsf{SIM}^{\mathcal{U},t,k}(\lambda, w_C, L)$$

Next we argue that the error term on the honest client's inputs after the collusion attack of t parties is larger than previous approaches. For this, we require the following additional property. Consider the case of $n-1$ collusion; we define collusion privacy as follows:

Collusion-Privacy: An n-party protocol provides *Collusion-Privacy*, for an aggregation function f and a probability distribution

		Client Components		
Users	Server	Setup	Training	Encrypt
200	20.3	193.0	7.5	2.9
400	42.7	404.0	7.7	5.6
600	61.4	631.6	7.4	8.3
800	81.5	832.1	7.4	11.1
1000	101.1	1,046.5	7.3	13.9
3000	303.8	3,185.0	7.5	41.3
5000	532.5	5,277.9	7.9	68.9

(a) Time (ms) per protocol component for Graph 1.



(b) Total protocol execution time (log scale) for Graphs 1-3.

Figure 1: Timing results for our oblivious distributed differential privacy protocol.

$\mathcal{D}$, if any adversary, who controls all parties except client P_h, learns no more than the honest party's values $w_h + \eta$ where $\eta \leftarrow \mathcal{D}$ and $f(w_1, \ldots, w_n)$.

To measure the error, we quantify the difference between $f(D)$ and its perturbed value $\hat{f}(D)$ which is the error introduced by the differential private mechanism of the secure aggregation protocol.

Definition 2. (Error function) Let $D \in \mathcal{D}$, $f : \mathcal{D} \rightarrow \mathbb{R}$, and let $\delta = \frac{|f(D)-\hat{f}(D)|}{|f(D)|+1}$ (i.e., the value of the error). The error function is defined as $\mu = \mathbb{E}(\delta)$. The expectation is taken on the randomness of $\hat{f}(D)$. The standard deviation of the error is $\sigma = \sqrt{Var(\delta)}$.

After the execution of Protocol 1, parties receive the noisy sum of their inputs, i.e., $W = \sum_{i=1}^{n} w_i$. In prior non-oblivious works if $n-1$ parties collaborate and remove their weights from w then the final noise added to the weight of the honest party w_h is a value from $\mathcal{L}(\lambda)$, and hence, the error is $\mu = \frac{1}{|W|+1}\mathbb{E}|\mathcal{L}(\lambda)| = \frac{\lambda}{|W|+1}$.

In our oblivious case, if $n-1$ parties collaborate then the final noise added to the weight of the honest party w_h is $n-1$ times larger than $\mathcal{L}(\lambda)$, and hence, the error is $\mu = \frac{1}{|W|+1}\mathbb{E}|\sum_{i=1}^{n-1}\mathcal{L}(\lambda)| = \frac{(n-1)\cdot\lambda}{|W|+1}$.

However, in practice even if the parties cannot subtract their exact noise terms they can still try to subtract the average of the noise terms, or one of the two noise terms, or use the leakage L to reduce the amount of error. In Section 5.6, we empirically show that such an attack is little better than the attack of subtracting nothing.

Note that the output of the aggregation protocol, $W + \eta$, is generated such that η follows exactly the same distribution in both non-oblivious and oblivious cases, but the noise left after an $n-1$ attack against the oblivious case is higher. Thus our oblivious extension to prior works provides security from collusion while imposing no additional accuracy loss to the shared weights of the final model.

5 EXPERIMENTS

We concretely implemented our protocol in the open source multi-agent simulator ABIDES, which had been previously adapted to the domain of federated learning by Byrd et al. [2, 3]. The code for our new protocol will be published on acceptance.

5.1 Experimental Datasets

We evaluated our protocol using two common machine learning datasets: Adult Census Income and Kaggle Credit Card Fraud. [6, 8]

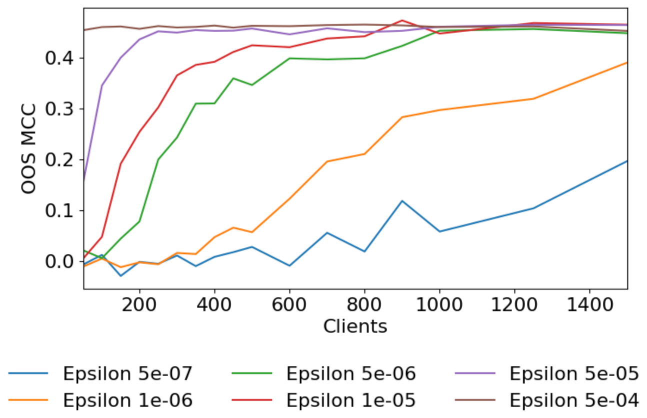
The census dataset provides 14 input features such as age, marital status, and occupation to predict a boolean output: whether an individual earns over $50K USD per year. We used a preprocessed version of the dataset from Jayaraman et al. following the method of Chadhuri et al. which transformed each categorical variable into a series of binary features, then normalized both features and examples, resulting in 104 features for consideration. [5, 12] We added a constant intercept feature to permit greater flexibility in the regression. Of the 45,222 records in our cleaned data set, there were 11,208 positive examples (about 25%), representing a moderately unbalanced dataset.

The credit card data set provides 26 transformed input features, representing the principal components of unknown original features, to predict a boolean output: whether a transaction was judged to be fraudulent. We used one additional feature without transformation (transaction amount) and excluded the timestamp feature entirely. Of the 284,807 records, only 492 (less than 0.2%) are labelled fraudulent, representing an extremely unbalanced dataset.

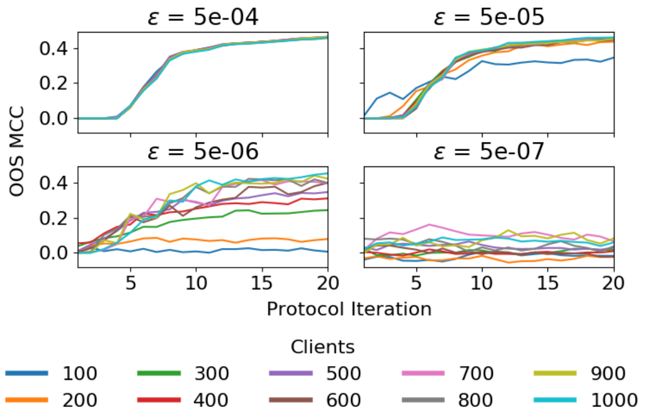
Apart from the mentioned preprocessing, both datasets were used in exactly the same way, with no changes to the protocol or simulation. For one complete simulation, 75% of the data was randomly selected for training, with each client receiving (at random) 200 training examples as its individual data each round. No client ever trained on the 25% holdout data. A custom client and server agent class were created for the simulation, which together implement Protocol 1 as listed herein.

5.2 Protocol Timing Results

We present empirical evidence to demonstrate the efficiency of our protocol by evaluating it on three different network graphs. In each case, the server is placed in New York City, while the clients are distributed around: New York City (**Graph 1**), London (**Graph 2**), or the entire world (**Graph 3**). The simulation customization includes pairwise connectivity, minimum latency, and stochastic additional latency per message, and properly delays protocol execution while messages are in simulated transit. Each simulation was conducted



(a) Performance of final shared model by ϵ privacy loss parameter.



(b) Performance per protocol iteration by ϵ and client count.

Figure 2: Out of sample Matthews Correlation Coefficient for our oblivious distributed differential privacy protocol.

on a 24-core Intel Xeon X5650 at 2.6GHZ with 128GB RAM and took between 32 seconds (100 clients) and 12 hours (5,000 clients) to complete.

In each experiment, there are 20 rounds of federated learning. Each client runs 50 logistic regression iterations within each round. Timing results are presented for the census dataset only, as the exact nature of the data should have no appreciable effect on protocol runtime. Figure 1a shows the time in milliseconds spent per protocol component for Graph 1 at varying client population sizes. The protocol components are:

- *Server*: data reception, storage, and aggregation tasks (mean time per protocol iteration).
- *Setup*: one-time Diffie-Hellman Key Exchange and noise generation (mean time per client party).
- *Training*: local logistic regression (mean time per client per protocol iteration).
- *Encryption*: applying encryption randomness, generating the next encryption randomness, and applying privacy noise (mean time per client per protocol iteration).

Figure 1b shows the estimated time required to execute the full protocol in the real world in each of our three network graph scenarios. This includes network latency and all client/server computation time, and is distinct from the time taken to run a simulation.

From Figure 1b, we conclude that as the number of clients increases, network latency becomes a less significant factor in total protocol execution time. (Mean pairwise latency is the primary difference among the network graphs.) From Figure 1a, we observe that the components driving computational time growth are performed only once per client (*Setup*) or only by the server (*Server*). We further note that the total number of messages sent by the n clients grows quadratically with n for the one-time *Setup* phase, and linearly with n for each *Encryption* phase. No messages are sent for the *Training* phase.

5.3 Protocol Accuracy Results

The MPC encryption in our protocol sums to zero in the final model, and therefore cannot cause accuracy loss. The differential privacy component causes accuracy loss inversely proportional to the ϵ privacy loss parameter, because a smaller ϵ leads to more added privacy noise. For example with $n = 200$ clients, final model accuracy worsens dramatically once $\epsilon < 5e - 5$.

In our Section 4.2 proofs, we show that our oblivious extension cannot cause additional accuracy loss, because the sum of our distributed privacy noise (difference of i.i.d. gamma distributions) equals the local privacy noise (Laplace distribution) of prior works.

For the sake of rigor, we nonetheless present empirical accuracy results from the census dataset. We measure the Matthews Correlation Coefficient (MCC) because it is a contingency method for calculating Pearson's r appropriate for imbalanced classification problems, and note that it shares the same interpretation. [14]

Figure 2a shows the out of sample MCC of our protocol's final shared model predictions against the correct values for a range of ϵ. Smaller ϵ values harm model accuracy, leading to a dynamic lower bound on useful values of ϵ: more clients allow smaller ϵ. For example when considering out of sample $MCC(n)$, with n being the client population size, in our experiments with $\epsilon = 1e - 5$: $MCC(100) = 0.005$, $MCC(200) = 0.254$, and $MCC(500) = 0.423$. For all evaluated client population sizes (50 to 5,000), models trained under Protocol 1 with $\epsilon \geq 5e - 4$ had similar accuracy to unsecured federated learning.

Figure 2b shows the impact ϵ can have on each round of federated learning: with $\epsilon = 5e - 4$ or $\epsilon = 5e - 7$, population size does not matter because either all tested populations succeed at learning or none do; but with $\epsilon = 5e - 6$, varying client population sizes learn at vastly different rates.

We also directly compared our oblivious protocol with $\epsilon = 5e - 4$ against unsecured federated logistic regression, both with 1000 clients, to determine the out of sample relative accuracy loss after the final round of learning. Our protocol experienced a relative mean squared error (MSE) loss of of $1.1e - 6$ and a relative MCC loss of 0.0018.

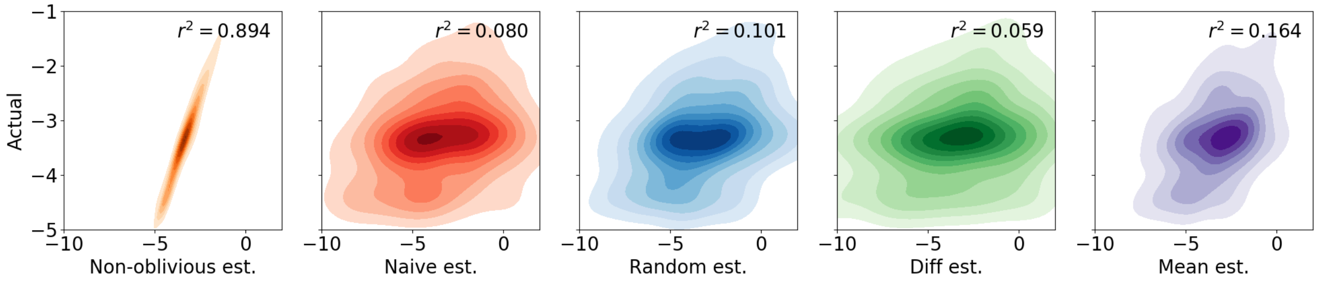


Figure 3: Density plot of actual versus estimated honest party weight for census data.

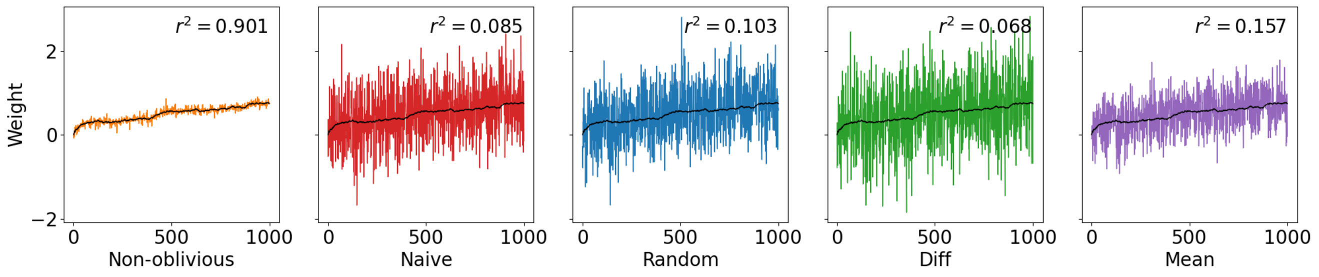


Figure 4: Actual (black) versus estimated (color) honest party weight for credit card data.

5.4 Adversarial Server Attack

Prior works like Byrd et al. and Bonawitz et al. discuss attacks from an untrusted server acting alone to infer the unencrypted weights of a particular client. [1, 3] To briefly summarize: for a single model weight M, the value transmitted by the honest party h to the server is $M_h = w_h + T_h + R_h$, where w_h is the honest party's original weight, T_h is the total privacy noise added by h, and $R_h = \sum_{c \in C} \pm r_{ch}$ for the set of all other clients C.

Under Protocol 1, one member of each client pair (h, c) will add r_{ch} and the other will subtract it. With just 100 participating clients, a single client's R_h is a summation of 99 values randomly generated (in our case) from the range $(0, 2^{32})$. The value of R_h is therefore orders of magnitude greater than the range of w_h. Since the server does not possess R_h or any of its components r_{ch}, it can make no meaningful inference about w_h. We do not present detailed analysis of this attack, as it is also successfully defended by the prior non-oblivious works that we cite.

5.5 Client Collusion attack

Our novel contribution is efficient defense against the $n-1$ attack, in which *all other clients* conspire to recover the unencrypted model weights of a single "honest" client. For example, if five banks collaborate to build a better credit card model, four might additionally conspire to recover information about the fifth bank's individual credit card transactions.

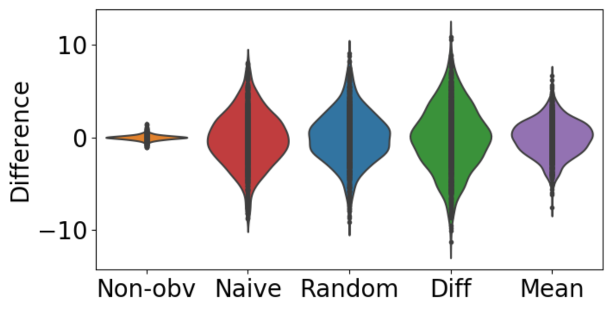
Let the honest client be h and the set of $n-1$ colluding clients be C. For a single model weight, let $F = w_h + W_C + T_h + T_C$ be the output of the aggregation protocol at the end of each iteration, where w_h is the honest party's original weight, T_h is the honest party's noise sum, $W_C = \sum_{c \in C} w_c$, $T_C = \sum_{c \in C} T_c$. Note that the sum of the randomness r, s is removed by design from the output when the computation is performed, so MPC cannot defend against this type of attack.

Under prior non-oblivious protocols in which clients generate and add their own noise locally, the colluding clients know W_C and T_C and can thus recover the honest client's noisy weight $w_h + T_h$.

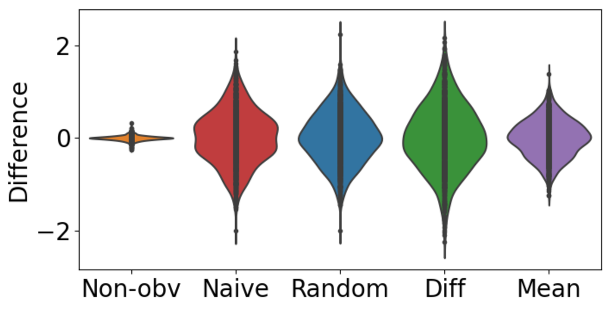
Under our oblivious protocol, the noise which cannot be subtracted in F has a more dispersed distribution that cannot be much narrowed by the colluding clients since h will have received from each colluding client c a choice of two difference of gamma privacy noises $\hat{\gamma}^0_{ch}$ and $\hat{\gamma}^1_{ch}$ and c will not know which was selected by h. For the most extreme Sybil attack, in which the server effectively controls the colluding clients, the protocol must include a secure shuffle component not present in our current simulations.

5.6 Evaluation of Collusion Attack

We proved the privacy improvement of our protocol in Section 4.2 and empirically demonstrate it here by evaluating five realistic attacks against an arbitrarily selected model weight. In each case, the colluding parties attempt to recover w_h and always remove W_C. In the **Non-Oblivious** case followed by prior works, the corrupted parties can accurately remove T_C. In the other four cases, Protocol 1



(a) Census data.



(b) Credit card fraud data.

Figure 5: Distribution of difference between estimated and actual honest party weight over 1,000 simulated attacks.

is attacked, and the corrupted parties must decide how to deal with the unknown noise choices by other parties i: under **Naive** they do nothing additional (i.e., they do not remove any noise terms); under **Random** each corrupted party c removes either $\hat{\gamma}^0_{ci}$ or $\hat{\gamma}^1_{ci}$ at random; and under **Diff** and **Mean** each corrupted party c removes the difference or mean of $\hat{\gamma}^0_{ci}$ and $\hat{\gamma}^1_{ci}$ respectively. These represent various efforts to estimate the level of noise which needs to be removed.

We consider the recovery attempt on both datasets across 1,000 full iterations of Protocol 1 with 100 clients participating. At the end of every iteration, 99 clients share information in an attempt to recover the unencrypted model weights of the one honest client. To simulate separate trials, clients retain no information across iterations. Privacy loss parameter $\epsilon = 5e-4$ was selected because it did not cause significant shared model accuracy loss for any tested number of clients.

Figures 3 and 4 show two different visualizations of the difference between the honest party's actual model weight and the collaborators' estimate of that weight. Figure 3 is a density plot of the census data recovery attempt and Figure 4 is a line plot of the credit card data recovery attempt.

Figures 5a and 5b summarize for both datasets the distribution of the difference between estimated and actual weights for each attack scenario using the same violin plot. For both datasets, the $n-1$ attack is successful (census $r^2 = 0.894$, credit card $r^2 = 0.901$) against the prior non-oblivious protocol, but not successful (census $r^2 = 0.164$ or worse, credit card $r^2 = 0.157$ or worse) against our new oblivious protocol.

5.7 Additional Residual Noise

In Section 5.5 we observed that the attackers cannot accurately remove T_h, the honest client's noise sum, because no conspirator c knows whether h added $\hat{\gamma}^0_{ch}$ or $\hat{\gamma}^1_{ch}$ to its weight.

In fact, under our protocol the server permutes and randomizes the MPC encrypted distributed noise choices, so the colluding clients cannot even remove T_C. When h generates $\hat{\gamma}^0_{hc}$, it is a composition:

$$\hat{\gamma}^0_{hc} = \gamma^0_{hc} + s_{hc} \tag{3}$$

where γ^0_{hc} is the initial noise choice generated from a difference of gamma distributions and s_{hc} is a large value. Honest party h computes $\hat{\gamma}^0_{hc}$ and $\hat{\gamma}^1_{hc}$ by adding the same s_{hc} to both γ^0_{hc} and γ^1_{hc}. It later subtracts s_{hc} from the transmitted version of its own corresponding weight to avoid disturbing the calculation.

Conspirator c knows $\hat{\gamma}^0_{hc}$ and $\hat{\gamma}^1_{hc}$ and its selection, but encryption randomness s_{hc} does not appear in the final model weight, only the unencrypted γ^0_{hc} or γ^1_{hc}. Conspirator c must thus remove γ^0_{hc} or γ^1_{hc} (as appropriate), and **not** $\hat{\gamma}^0_{hc}$ or $\hat{\gamma}^1_{hc}$. It cannot do this because it does not know s_{hc}, and therefore does not know γ^0_{hc} or γ^1_{hc}.

Thus under our Protocol 1, for each collaborator c, the additional error (beyond basic differential privacy noise) in the estimate of w_h will compose two factors:

(1) Not knowing whether h added $\hat{\gamma}^0_{ch}$ or $\hat{\gamma}^1_{ch}$.
(2) Being off by large encryption value s_{hc} when subtracting $\hat{\gamma}^0_{hc}$ or $\hat{\gamma}^1_{hc}$.

We evaluated the first error in this work and left detailed analysis of the second for future work, although it is already part of the additional security of our protocol. Thus our results in Section 5.6 represent an unrealistically *successful* attack and the model weight estimates will in reality be even worse.

We have also left the case where clients drop off during the protocol as future work, meaning our mechanism is well suited to financial applications for cross-silo or inter-firm federated learning, and less suited for mobile devices which can go offline.

6 CONCLUSION

We presented an efficient mechanism for oblivious distributed differential privacy that is the first to secure against $n-1$ collusion attacks on the clients' model parameters. We leveraged that mechanism to construct a more secure federated learning protocol suitable for use by financial firms with sensitive customer data. We detailed the protocol, proved its security against a semi-honest adversary, and empirically evaluated it in a realistic simulation on two datasets. We found that even with 5,000 clients, the full protocol can execute in less than two minutes with negligible model accuracy loss versus unsecured learning. Across all cases and both datasets, our simulated collusion attack was successful against prior works but unsuccessful against our new protocol.

ACKNOWLEDGMENTS

This material is based upon research supported by the National Science Foundation under Grant No. 1741026 and by a JP Morgan Fellowship.

This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

REFERENCES

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

[2] David Byrd, Maria Hybinette, and Tucker Hybinette Balch. 2020. ABIDES: Towards high-fidelity multi-agent market simulation. In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 11–22.

[3] David Byrd and Antigoni Polychroniadou. 2020. Differentially private secure multi-party computation for federated learning in financial applications. In *Proceedings of the First ACM International Conference on AI in Finance*. 1–9.

[4] Jeffrey Champion, Abhi Shelat, and Jonathan Ullman. 2019. Securely Sampling Biased Coins with Applications to Differential Privacy. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 603–614. https://doi.org/10.1145/3319535.3354256

[5] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. 2011. Differentially private empirical risk minimization. *Journal of Machine Learning Research* 12, Mar (2011), 1069–1109.

[6] Andrea Dal Pozzolo, Olivier Caelen, Reid A Johnson, and Gianluca Bontempi. 2015. Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*. IEEE, 159–166.

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

[8] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

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

[10] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*. Springer, 265–284.

[11] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. 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*. 218–229. https://doi.org/10.1145/28395.28420

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

[13] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[14] Brian W Matthews. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405, 2 (1975), 442–451.

[15] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2018. Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks. *arXiv preprint arXiv:1812.00910* (2018).

[16] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.