Practical Collaborative Learning for Crowdsensing in the Internet of Things with Differential Privacy

Yuanxiong Guo and Yanmin Gong
School of Electrical and Computer Engineering
Oklahoma State University, Stillwater, OK 74078
Email: {richard.guo, yanmin.gong}@okstate.edu

***Abstract*—Machine learning is increasingly used to produce predictive models for crowdsensing applications such as health monitoring and query suggestion. These models are more accurate when trained on large amount of data collected from different sources. However, such massive data collection presents serious privacy concerns. The personal crowdsensing data such as photos, voice records, and locations is often highly sensitive, and once being sent out to the collecting companies, falls out of the control of the crowdsensing users who own it. This may preclude the practice of transmitting all user data to a central location and training there using conventional machine learning approaches.**

In this paper, we advocate an alternative approach that leaves data stored on the user side and learns a shared model by coordinating local training of crowdsensing users in an iterative process. Specifically, we focus on regularized empirical risk minimization and propose an efficient scheme based on decomposition that enables multiple crowdsensing users to jointly learn an accurate learning model for a given learning objective without sharing their private crowdsensing data. We exploit the fact that the optimization problems used in many learning tasks are decomposable and can be solved in a parallel and distributed way by the alternating direction method of multipliers (ADMM). Considering the heterogeneity of different user devices in practice, we propose an asynchronous ADMM algorithm to speed up the training process. Our scheme lets users train independently on their own crowdsensing data and only share some updated model parameters instead of raw data. Moreover, secure computation and distributed noise generation are novelly integrated in our scheme to guarantee differential privacy of the shared parameters in the execution of the asynchronous ADMM algorithm. We analyze the privacy guarantee and demonstrate the privacy-utility trade-off of our privacy-preserving collaborative learning scheme empirically based on real-world data.

I. Introduction

With the proliferation of smart devices that have built-in sensors, Internet connectivity, and programmable computation capability, the concept of crowdsensing, where individual devices collectively sense, share, and analyze data to learn phenomena of common interests, is gaining popularity and driving the evolution of the Internet of things [1]. Although often used for collecting and analyzing aggregate statistics from a group of participants, crowdsensing can also perform more complex tasks over the collected data via machine learning. For instance, Google is collecting user input data in Gboard on Android to improve Gboard's query suggestion model [2].

It is well-known that machine learning models become more accurate as the training data grows bigger and more diverse. The current practice of combining data from different sources or individuals is to send it to a central place (e.g., a cloud datacenter) and then analyze it there. Although highly efficient, such massive centralized collection of data from millions of crowdsensing participants has raised several issues. First, personal crowdsensing data such as photos, voice records, and locations is often highly sensitive, and could leak a lot of private information about the users. Storing such highly sensitive data in a single place presents a large privacy risk to attackers. Second, once the data is sent out to the collecting companies which may keep it indefinitely, crowdsensing users who actually own the data have no control of it. They can neither delete the data nor restrict its usage. Third, the predictive models learned from crowdsensing data are proprietary to the collecting companies, and crowdsensing users have to send queries to the companies later for using these models.

We investigate an alternative scheme based on collaborative learning to address the aforementioned issues of centralized collection and learning in crowdsensing. In our approach, multiple crowdsensing users collaboratively train a learning model under the coordination of a cloud server in an iterative manner. Each user has a training dataset that is never uploaded to the server. Instead, at each iteration of the training process, each user locally trains a model based on its own dataset and some shared information received from the cloud server, and then only shares some updated model parameters with the server. At the end of the iterative process, both users and the server would obtain the accurate learning model. Our approach follows the principle of *focused collection* recommended by the 2012 White House report on privacy of consumer data [3]. Security and privacy risks are also reduced in our approach through limiting the attack surface to only users, rather than users and the cloud server.

Specifically, we focus on regularized empirical risk minimization [4], which have been widely adopted in many learning applications. We exploit the fact that the regularized empirical risk minimization (ERM) problems can be decomposed based on the alternating direction method of multipliers (ADMM) and then iteratively solved in a parallel and distributed way by users using their local datasets. Therefore, the model could be learned without the need for direct access to the raw training data, providing some privacy for the users. However, there are two challenges in using the ADMM. First, user devices are heterogeneous in their computation capabilities. It

978-1-5386-4586-4/18/$31.00 ©2018 IEEE

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 28,2021 at 15:26:30 UTC from IEEE Xplore. Restrictions apply.

could be very slow for the cloud server to wait for updates from all users before proceeding to the next iteration. Second, during the iterative process, users needs to send their updated model parameters to the server, which may leak some private information about their datasets. We tackle the first challenge by adopting an asynchronous ADMM algorithm to solve the optimization problem, which does not require the server to receive all user updates before proceeding and thus speeds up the iterative process. Moreover, a differentially private parameter sharing mechanism is designed in the asynchronous algorithm to tackle the second challenge.

In summary, the main contributions of this paper are as follows.

- We propose an alternative approach to centralized data collection for privacy-preserving regression learning over crowdsensing data. Our approach enables multiple crowdsensing users to jointly learn an accurate model without sharing their private datasets.
- Considering the heterogeneity of user devices, we develop an asynchronous distributed ADMM algorithm to enable efficient collaborative learning. The proposed asynchronous algorithm can reduce the waiting time of the server to proceed to the next iteration and greatly speed up the iterative process.
- We design a differentially private parameter sharing mechanism based on secure computation and distributed noise generation to limit the private information leakage during the execution of the asynchronous ADMM algorithm in a rigorous way.
- We conduct an extensive empirical evaluation of the proposed approach based on real-world data to demonstrate its effectiveness.

The rest of the paper is organized as follows. Section II gives some background and preliminaries on ADMM and differential privacy. Section III describes the system architecture, threat model, design goals, and gives an overview of the proposed approach used in this paper. The collaborative learning scheme based on asynchronous distributed ADMM is presented in Section IV. The differentially private parameter sharing mechanism to minimize the indirect information leakage during the execution of the asynchronous algorithm is illustrated in Section V. Evaluation results based on real-world data are described in Section VI, and related work is reviewed in Section VII. Finally, Section VIII gives the conclusion of this paper.

II. Background and Preliminaries

A. Alternating Direction Method of Multipliers

The ADMM is an algorithm suitable for distributed optimization and has been widely used in applied statistics and machine learning [5]. Problems that can be solved by ADMM have the following form:

$$\begin{aligned} \min_{x,z} \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = c, \end{aligned} \tag{1}$$

where the objective function is separable over two sets of variables x and z. The augmented Lagrangian of the above problem is

$$L_\rho(x, z, y) := f(x) + g(z) + y^T(Ax + Bz - c) + (\rho/2)\|Ax + Bz - c\|_2^2, \tag{2}$$

where $\rho > 0$ is the penalty parameter and y is the dual variable corresponding to the constraint $Ax + Bz = c$. At each iteration k, ADMM solves (1) by the following steps:

$$x^{k+1} := \operatorname*{argmin}_x L_\rho(x, z^k, y^k), \tag{3}$$

$$z^{k+1} := \operatorname*{argmin}_z L_\rho(x^{k+1}, z, y^k), \tag{4}$$

$$y^{k+1} := y^k + \rho(Ax^{k+1} + Bz^{k+1} - c). \tag{5}$$

Similar to dual descent algorithm, it consists of x-minimization step (3), z-minimization step (4), and a dual variable update (5). However, different from dual descent algorithm, in ADMM x and z are updated in an alternating or sequential fashion, which allows for decomposition when f or g are separable. The convergence of ADMM can be proved under very mild assumptions, which generally hold in practice. Moreover, ADMM converges to modest accuracy, which is sufficient for many applications, within a few tens of iterations in many cases [5].

B. Differential Privacy

Differential privacy [6] was originally proposed for the setting where a trusted data curator with a database containing records from multiple individuals publishes perturbed statistics derived from the database using a randomized mechanism. Formally speaking, a randomized mechanism $\mathcal{M}$ satisfies (ϵ, δ)-differential privacy if for all adjacent databases $D, D' \in \mathcal{D}$ that differ in exactly one record, and all possible output subset $O \subseteq \text{range}(\mathcal{M})$,

$$\Pr[\mathcal{M}(D) \in O] \leq e^\epsilon \Pr[\mathcal{M}(D') \in O] + \delta. \tag{6}$$

Specifically, consider any function g that takes as input a database $D \in \mathcal{D}$ and outputs a numeric vector $o \in \mathbb{R}^d$. The *Gaussian Mechanism* gives a general method for generating a privacy-preserving approximation to the function g by adding zero-mean Gaussian noise to each of the d coordinates of the output vector o. Formally, we have the following theorem:

Theorem 1. *Let $\epsilon \in (0, 1)$ and $\delta > 0$ be arbitrary. Then the Gaussian Mechanism which adds Gaussian noise $\mathcal{N}(0, \sigma^2)$ to each coordinate of the function output is (ϵ, δ)-differentially private if the scale of the Gaussian noise satisfies*

$$\sigma \geq \sqrt{2\ln(1.25/\delta)}\frac{\Delta_2(g)}{\epsilon}. \tag{7}$$

Here $\Delta_2(g)$ is the ℓ_2-sensitivity of the function g defined as

$$\Delta_2(g) := \max_{D,D'} \|g(D) - g(D')\|_2, \tag{8}$$

where D and D' are any two adjacent databases in $\mathcal{D}$ which differ in only one record.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 28,2021 at 15:26:30 UTC from IEEE Xplore. Restrictions apply.

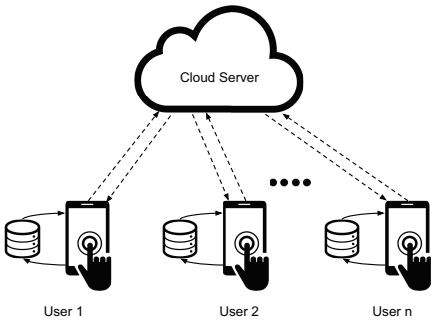

Fig. 1. The crowdsensing system model.

Due to its robustness against adversarial background knowledge, differential privacy has been increasingly accepted as the *de facto* standard for private data analysis and deployed in some real-world scenarios [7], [8]. Moreover, differential privacy has some useful properties. First, it is immune to post-processing, which is formalized below:

Theorem 2 (Post-processing)**.** *Suppose a mechanism $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ preserves (ϵ, δ)-differential privacy. Then for any function $h : \mathcal{R} \to \mathcal{R}'$, the (functional) composition $h \circ M : \mathcal{D} \to \mathcal{R}'$ also preserves (ϵ, δ)-differential privacy.*

Second, it is easy to combine several differentially private mechanisms as shown in the following composition rule:

Theorem 3 (Sequential Composition)**.** *Let $\mathcal{M}_1 : \mathcal{D} \to \mathcal{R}_1$ be an (ϵ_1, δ_1)-differentially private mechanism, and let $\mathcal{M}_2 : \mathcal{D} \to \mathcal{R}_2$ be an (ϵ_2, δ_2)-differentially private mechanism. Then their combination, defined to be $\mathcal{M}_{1,2} : \mathcal{D} \to \mathcal{R}_1 \times \mathcal{R}_2$ by the mapping $\mathcal{M}_{1,2}(D) = (\mathcal{M}_1(D), \mathcal{M}_2(D))$ is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-differentially private.*

III. System Models and Solution Overview

In this section, we first present the system architecture for privacy-preserving collaborative learning and then describe the threat model and design goals we aim to achieve in this paper.

A. System Architecture

We consider a crowdsensing system consisting of a cloud server and a set of users $[n] := \{1, \ldots, n\}$ as depicted in Fig. 1. Each user $i \in [n]$ in the system has a local sensing dataset $D_i = \{(x_{ij}, y_{ij}) \in \mathcal{X} \times \mathcal{Y} : \forall j \in [J_i] := \{1, \ldots, J_i\}\}$ stored on its device, where m_i is the total number of training samples in the dataset, $x_{ij} \in \mathcal{X}$ is the d-dimensional feature vector of the j-th training sample, and $y_{ij} \in \mathcal{Y}$ is the corresponding label of the j-th training sample. The sets $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y} \subseteq \mathbb{R}$ are the feature space and label space, respectively. For each training sample (x_{ij}, y_{ij}), we assume without loss of generality that $\|x_{ij}\|_2 \leq 1$ since any feature vector can be normalized to enforce this assumption.

The cloud server coordinates the collaborative learning among the users in the system. Each user can communicate directly with the server over a public network. The goal of the system is to learn a predictive model h from the collective sensing dataset of all users $\cup_{i=1}^{n} D_i$ that predicts the label y given a feature vector x. Consider a linear predictor h with a tunable vector w, and $h(x; w) := w^T x$ denotes the predicted label given a feature vector x. We also define a loss function l that measures the difference between the observed label and the predicted label. Similar to [4], we assume that the loss function $l(\cdot)$ is convex and differentiable with $|l'(\cdot)| \leq 1$. As a common method in statistical learning [9], a wide range of learning tasks can be represented by choosing specific forms of l, and then finding the optimal parameter w^* to minimize the regularized empirical risk over the collective dataset:

$$\min_{w} \sum_{i=1}^{n} \sum_{j=1}^{J_i} l(w^T x_{ij}, y_{ij}) + \beta r(w), \tag{9}$$

where $r(w)$ is the regularizer (such as ℓ_1-norm and ℓ_2-norm) to prevent over-fitting, and $\beta > 0$ is a parameter which controls the weight of the regularizer. Some common examples of learning tasks include linear regression where $l(h(x_{ij}; w), y_{ij}) = (y_{ij} - x_{ij}^T w)^2$, $y_{ij} \in \mathbb{R}$, logistic regression where $l(h(x_{ij}; w), y_{ij}) = \log\big(1 + \exp\big(-y_{ij} x_{ij}^T w\big)\big)$, $y_{ij} \in \{-1, 1\}$, and support vector machines where $l(h(x_{ij}; w), y_{ij}) = \max\{0, 1 - y_{ij} x_{ij}^T w\}$, $y_{ij} \in \{-1, 1\}$.

B. Threat Model

We assume the following threat model. In the system, the cloud server is assumed to be *honest-but-curious*. That means the server will honestly follow the designed protocol and make the correct computations. However, it is curious about the user private sensing data and may infer it from the shared messages during the execution of the protocol. The cloud can also have arbitrary auxiliary information, which can be obtained from public datasets or personal knowledge about a specific user. There might exist a passive outside attacker who can eavesdrop all shared messages in the execution of the protocol, but it will not actively inject false messages into or interrupt the message transmission in the protocol.

All users in the system are assumed to be *honest-but-curious* as well. However, some of them may collude with the server to infer private information about a specific user. In this paper, we do not consider malicious users who, for instance, may launch data pollution attack by lying about their private datasets or returning the incorrect computed results to mess up the learning process. It will be left as our future work.

C. Design Goal

Our goal is to design a scheme that enables multiple users to jointly learn an accurate machine learning model for a given learning task while preserving differential privacy of their sensing datasets. Moreover, the communication and computation overhead should be kept low considering the resource constraints of users in practice.

D. Solution Overview

To address the privacy concern of users about exposing their private datasets during the collaborative learning process

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 28,2021 at 15:26:30 UTC from IEEE Xplore. Restrictions apply.

while obtaining the accurate learning model, in this paper we adapt tools from ADMM, secure computation, and differential privacy to develop a novel approach with high accuracy and rigorous privacy guarantee. Our proposed approach consists of two essential components: an asynchronous distributed ADMM algorithm to solve the optimization problem (9) that enables collaborative learning without directly sharing private user datasets, and a differentially private parameter sharing mechanism to limit the indirect information leakage during the execution of the asynchronous algorithm.

IV. Collaborative Learning based on Distributed and Asynchronous ADMM

To apply ADMM for collaborative learning, we first reformulate (9) into the following equivalent consensus problem:

$$\min_{w_0, w_i, \forall i \in [n]} \quad \sum_{i\in[n]} \sum_{j\in[J_i]} l(w^T x_{ij}, y_{ij}) + \beta r(w_0) \tag{10a}$$

$$\text{s.t.} \quad w_i = w_0, i = 1, \ldots, n, \tag{10b}$$

where w_i is the i-th user's local copy of the model to be learned, and all of them need to reach consensus with the global copy of the model w_0 at the server side. We define $l(D_i; w_i) := \sum_{j\in[J_i]} l(w^T x_{ij}, y_{ij})$ as the loss function of the i-th user given its local model parameter w_i.

Problem (10) is in the standard form and can be efficiently solved by the ADMM algorithm (3)–(5) in the following parallel and distributed way:

$$w_0^{k+1} := \underset{w_0}{\text{argmin}} \ \beta r(w_0) + \frac{n\rho}{2}\left\| w_0 - \overline{w}^k - \overline{\lambda}^k \right\|_2^2$$

$$w_i^{k+1} := \underset{w_i}{\text{argmin}} \ l(D_i; w_i) + \frac{\rho}{2}\left\| w_i + \lambda_i^k - w_0^{k+1} \right\|_2^2, \forall i$$

$$\lambda_i^{k+1} := \lambda_i^k + w_i^{k+1} - w_0^{k+1}, \forall i$$

where $(\lambda_i, \forall i)$ are the scaled dual variables corresponding to the constraints (10b), the overline denotes the average (over $i = [n]$) of a vector (e.g., $\overline{w}^k = (1/n)\sum_i w_i^k$, $\overline{\lambda}^k = (1/n)\sum_i \lambda_i^k$), and k is the iteration number. From the above algorithm, we can derive the following synchronous protocol for collaborative learning: each user optimizes its local model in parallel based on its local dataset and the received global model from the server, and then sends the updated model to the server; in turn, the server updates the global model parameter by driving the local parameters into consensus, and then distributes the updated value back to the users, and the process re-iterates. *With this method, each user's raw dataset is kept locally and not uploaded to the server, preventing direct information leakage.* However, updates in the above distributed ADMM algorithm have to be synchronized in the sense that the server has to wait for all the users to finish their local updates before it can proceed to the next iteration. In practice, the users could have different delays because of heterogeneity in proceeding speed and training dataset, and in the above synchronous protocol the server needs to wait for the slowest user to finish its update before starting the next iteration, leading to the straggler problem [10]. Moreover, if some users get disconnected and drop out from the system temporarily, the synchronous algorithm has to stop immediately.

To speed up the convergence of the algorithm under heterogeneous environments and make it more robust to individual user failure, we need to allow asynchronous updates, where the server starts to perform its update immediately upon receiving updates from only a subset of users. Our asynchronous distributed algorithm is inspired by the recent development of asynchronous distributed algorithms [11] and works as follows.

Initialization. The server is responsible for updating the global model w_0, while each user i is responsible for updating its local model w_i and dual variable λ_i. The server keeps an iteration counter k which starts from 0 and is increased by 1 after each w_0 update. Similarly, each user keeps an iteration counter k_i which starts from 0 and is increased by 1 after each λ_i update. As the proposed algorithm is asynchronous, the counters k and $k_i, \forall i$ are updated independently. Denote by $w_i^{k_i}$ and $\lambda_i^{k_i}$ the values of w_i and λ_i when the i-th user is at iteration k_i, respectively, and w_0^k the value of w_0 when the server is at iteration k.

Step 1: w_0-update by the server. At iteration k, the server needs to wait for the users' (w_i, λ_i) updates before it can update the global model w_0. Two preconditions are imposed here for the server to update. The first condition is called *partial barrier*, where the server only needs to wait for a minimum of $s \in [1, n]$ updates from users. Let Ω_k be the set of users whose updates are received by the server and during iteration k. So this condition requires that $|\Omega_k| \geq s$. The second condition is called *bounded delay*, where update from every user has to be processed by the server at least once every $\tau \geq 1$ iterations. Let τ_i be the variable kept by the server to count the delay for each user i. For each iteration k, if $i \in \Omega_k$, τ_i is set to be zero. Otherwise, τ_i is increased by one. Therefore, the second condition is not satisfied as long as there exists a user $i \in [n] \setminus \Omega_k$ such that $\tau_i \geq \tau - 1$ at any iteration k. Here both s and τ are design parameters of our asynchronous algorithm. It is easy to see that the synchronous ADMM algorithm is a special case of our asynchronous algorithm with $s = n$ or $\tau = 1$.

When both conditions are met, the server updates w_0 based on the updated values $\{(\hat{w}_i, \hat{\lambda}_i)\}_{i\in\Omega_k}$ received from the users in the following way:

$$w_0^{k+1} := \underset{w_0}{\text{argmin}} \ \beta r(w_0) + \frac{n\rho}{2}\left\| w_0 - \overline{w}^k - \overline{\lambda}^k \right\|_2^2, \tag{11}$$

where w_i^k and λ_i^k are the most recent updates from user i at the server until iteration k. Specifically, for users $i \in \Omega_k$, $w_i^k = \hat{w}_i$ and $\lambda_i^k = \hat{\lambda}_i$. For the other users $i \in [n] \setminus \Omega_k$, no new updates are received in this iteration and hence $w_i^k = w_i^{k-1}$ and $\lambda_i^k = \lambda_i^{k-1}$. After that, the iteration counter of the server k is increased by 1, and the updated w_0^{k+1} are sent back to only the users in Ω_k. The server then waits until satisfying the two preconditions again to move to the next iteration.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 28,2021 at 15:26:30 UTC from IEEE Xplore. Restrictions apply.

Step 2: (w_i, λ_i)-update by the i-th user. At iteration k_i, based on the updated value $\hat{w}_0$ received from the server, the user i updates its local model parameter w_i and dual variable λ_i as

$$w_i^{k_i+1} := \underset{w_i}{\operatorname{argmin}}\ l(D_i; w_i) + \frac{\rho}{2}\left\|w_i + \lambda_i^{k_i} - \hat{w}_0\right\|_2^2, \quad (12)$$

$$\lambda_i^{k_i+1} := \lambda_i^{k_i} + w_i^{k_i+1} - \hat{w}_0. \quad (13)$$

The new local model $w_i^{k_i+1}$ and dual variable $\lambda_i^{k_i+1}$ are then sent to the server, and the iteration counter k_i is increased by 1. The user then waits for the next w_0-update from the server before updating again.

V. Differentially Private Parameter Sharing

With the protocol derived from the above asynchronous algorithm, the dataset D_i is kept locally at each user i without being shared with others. Therefore, it can prevent direct information leakage. However, the shared parameters w_i^k and λ_i^k sent to the server by the users $i \in \Omega_k$ at each iteration k are computed from its private dataset D_i and may lead to privacy leakage as demonstrated in model inversion attack [12]. Therefore, we need to preserve privacy of these shared parameters. The goal of this section is to achieve differential privacy of the shared parameters in the asynchronous distributed ADMM algorithm by sanitizing them before they leave the user. Note that the dual variables λ_i^k are derived from w_i^k based on (13) without further querying user datasets. According to Theorem 2, it is straightforward to preserve the differential privacy of λ_i^k by using the noisy $\widetilde{w}_i^{k_i+1}$ in (13) if we can guarantee differential privacy of w_i^k. Therefore, in the following, we will restrict our focus to w_i^k in a specific iteration k.

A straightforward approach to achieve differential privacy in our setting would be through randomized perturbation, where each user $i \in \Omega_k$ adds enough random noises into its shared parameters w_i^k directly at each iteration k before sending them out such that the server would not be able to learn much about its individual training samples in D_i from the received parameters. This approach is also called local differential privacy [7] in literature. However, since the server is not trusted in our setting, the noises added by each individual user must be large enough to satisfy differential privacy, which may accumulate too much noise in the final aggregate results.

To tackle this issue, we observe that the server only needs to know the average of the local models $\overline{w}^k$ in (11) for global model updating at each iteration. Furthermore, we have

$$\overline{w}^k = \sum_{i\in[n]} \frac{w_i^k}{n} = \sum_{i\in[n]} \frac{w_i^{k-1} + \Delta w_i^k}{n} = \overline{w}^{k-1} + \frac{\sum_{i\in\Omega_k} \Delta w_i^k}{n},$$

where $\Delta w_i^k := w_i^k - w_i^{k-1}$ represents the update of the local model of user $i \in \Omega_k$ during iteration k, while the local models of all other users $i \in [n] \setminus \Omega_k$ do not change during iteration k. Therefore, the server only needs to know the sum of the updates of the local model parameters $\sum_{i\in\Omega_k} \Delta w_i^k$ from all users in Ω_k to proceed at each iteration rather than individual values of $\Delta w_i^k, \forall i \in \Omega_k$. Based on this observation, if we can have an efficient secure aggregation protocol such that the server can only obtain the sum of the updates instead of the individual values, the noise added by each user can be much smaller, improving the utility of the aggregated results. In the following, we will design an efficient secure aggregation protocol based on secrete sharing and then combine this protocol with a distributed noise generation scheme to guarantee differential privacy of shared parameters while achieving high utility.

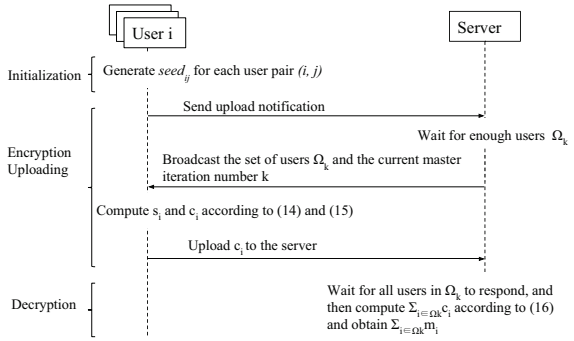

Fig. 2. Basic protocol for efficient secure aggregation in asyncrhonous setting.

In the remainder of this section, we first describe the secure aggregation protocol in our asynchronous setting such that the cloud server learns nothing but the noisy sum. Then we describe how each participating user should choose the noise distribution so that the differential privacy of an individual participating user is protected even when a subset of participants may drop out or collude with the cloud server.

A. Secure Aggregation in Asynchronous Setting

For our asynchronous setting, a secure aggregation protocol should be able to (1) hide individual user data, (2) recover the sum for an uncertain set of users in each round, and (3) incur low communication cost for participating users. Our basic idea to provide a secure aggregation protocol that satisfies the above requirements is to utilize secret sharing protocols in a communication-efficient way. Denote by m_i the plaintext message (i.e., model parameters) of user i that needs to be shared with the server. Our proposed protocol involves minimal interactions between the users and the server during each iteration k of the asynchronous algorithm, which is critical to improve the performance of the asyncrhonous setting, and consists of the following two main steps:

- *Encryption uploading:* Users in Ω_k upload their own encrypted message $c_i, i \in \Omega_k$ to the server.
- *Decryption:* The server decrypts the sum of the messages sent by users in Ω_k.

The basic idea of the protocol is to protect the message m_i of user i by hiding it with a random number s_i in the plaintext space, i.e., $c_i = m_i + s_i$. However, the challenge here is how to remove the random number s_i from the final ciphertext at the server part. To this end, we require that all the s_i will sum up to 0, i.e., $\sum_{i\in\Omega_k} s_i = 0$, which enables the cloud server to recover $\sum_{i\in\Omega_k} m_i$ but prevents it from recovering each individual message m_i. However, each user

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 28,2021 at 15:26:30 UTC from IEEE Xplore. Restrictions apply.

needs to communicate with each other in order to generate such secrets s_i to achieve this, which is quite inefficient in terms of communication overhead. To reduce the communication overhead, we introduce a pseudorandom function (PRF) F here. The PRF F takes a random seed $seed_{i,j}$ that both users agree on during initialization and the iteration number k, and outputs a different pseudorandom number $F(seed_{i,j}, k)$ at each iteration. User i could calculate the shared secret s_{ij} without interacting with user j in each iteration as long as they both use the same seed and iteration number, and thus each user could calculate s_i independently. This procedure greatly reduces the amortized communication overhead of our protocol over multiple iterations as in our algorithm.

The detailed protocol is depicted in Fig. 2. All user needs to go through an initialization step upon enrollment which involves pairwise communications with all other users (which can be facilitate by the server) to generate a random seed $seed_{ij}$. After this initialization step, all enrolled users could upload their messages through the encryption uploading step. In each round k, only a subset of users would upload their messages. Users send a notification signal to the server once they are ready to upload their updated shared parameters, and the server waits until receiving notifications from enough users. The server then broadcasts the information Ω_k to all users in Ω_k who will upload their messages in this iteration k. User $i \in \Omega_k$ would first compute its secret at the current iteration as follows:

$$s_i = \sum_{j \in \Omega_k \setminus \{i\}} (s_{ij} - s_{ji}), \tag{14}$$

where $s_{ij} = F(seed_{i,j}, k)$ is a secret known by both user i and j. User i could then generate the encrypted message for m_i as $c_i = m_i + s_i$. In the decryption step, the server receives c_i from users in Ω_k. The server could then recover the sum of plaintext messages from users in Ω_k as follows:

$$\begin{aligned} \sum_{i \in \Omega_k} c_i &= \sum_{i \in \Omega_k} m_i + \sum_{i \in \Omega_k} \sum_{j \in \Omega_k \setminus \{i\}} (s_{ij} - s_{ji}) \\ &= \sum_{i \in \Omega_k} m_i. \end{aligned} \tag{15}$$

Note that in the above protocol, we assume all active users in Ω_k have stable connection to the cloud server. In some cases, the users may not maintain connections during the protocol. We could modify our protocol to address user churns during the interaction process. The basic idea is to use (t, n)-threshold secret sharing $(t < n)$ [13] to improve the robustness of our protocol, so that even some of the users are offline, we could still recover the missing information. When users are uploading their encrypted message to the server, they also calculate secret shares of s_{ij} and encrypt each share of s_{ij} with the encryption key of a user in Ω_k. These encrypted shares are sent to the server together with the ciphertext c_i. When the server needs to recover c_i, it needs all the $s_{ij}, i, j \in \Omega_k$, and thus it requests the secret shares for the missing s_{ij} from remaining users. As long as there are at least t remaining users, they could help the server recover s_{ij} and decrypt $\sum_{i \in \Omega_k} m_i$. This improves robustness of our protocol, but brings additional communication cost. In practice we need to consider the probability for user dropout in order to decide whether it is beneficial to adopt the modified protocol.

B. Distributed Noise Generation

The cryptographic construction of Section V-A ensures that the cloud server learns nothing other than what it already knows and the sum of the shared parameters revealed at each iteration. However, individual privacy can still be violated indirectly as the aggregate sum may leak information about a user's private dataset. In this section, we show how to guarantee (ϵ, δ)-differential privacy of the aggregate sum in the collaborative learning process.

Note that $\sum_{i \in \Omega_k} \Delta w_i^k$ is a function of user datasets $\cup_{i \in \Omega_k} D_i$ through optimization problems (12). To add appropriate scales of noise, we need to estimate the sensitivities of the optimal solutions to these optimization problems. Specifically, we have the following lemma:

Lemma 1. *The ℓ_2-sensitivity of the optimal solution $w_i^{k_i+1}$ with respect to dataset D_i in* (12) *is $2/\rho$.*

Proof. Let D_i and D_i' be two datasets that differ in the value of the j-th sample. Moreover, we let $G(w_i) = l(D_i; w_i) + (\rho/2)\left\|w_i + \lambda_i^{k_i} - \hat{w}_0\right\|_2^2$, $g(w_i) = l(D_i'; w_i) - l(D_i; w_i)$, $w_i^* = \mathrm{argmin}_{w_i} G(w_i)$, and $(w_i')^* = \mathrm{argmin}_{w_i} G(w_i) + g(w_i)$. We observe that due to the convexity of $l(\cdot)$ and ρ-strong convexity of $(\rho/2)\left\|w_i + \lambda_i^{k_i} - \hat{w}_0\right\|_2^2$, $G(w_i)$ is ρ-strongly convex. Moreover, $G(w_i) + g(w_i)$ is also ρ-strongly convex. Note that both $G(w_i)$ and $g(w_i)$ are differentiable. According to the Lemma 7 from [4], we have

$$\|w_i^* - (w_i')^*\|_2 \leq \frac{1}{\rho} \max_{w_i} \|\nabla g(w_i)\|_2 \tag{16}$$

Since $\|\nabla g(w_i)\|_2 \leq 2$ according to our assumptions on the loss function $l(\cdot)$, the conclusion follows. □

From the above lemma, it is easy to find that the ℓ_2-sensitivity of $\sum_{i \in \Omega_k} \Delta w_i^k$ is also $2/\rho$. Therefore, based on this fact and Theorem 1, it is sufficient to preserve (ϵ, δ)-differential privacy of shared parameters $\sum_{i \in \Omega_k} \Delta w_i^k$ at iteration k by adding Gaussian noise $\mathcal{N}(0, \sigma^2)$ to the sum of local model updates with $\sigma \geq \sqrt{2\ln(1.25/\delta)}(2/\rho\epsilon)$. In our setting, the users do not fully trust the cloud server. Therefore, we cannot rely on the cloud server to add noise to the sum $\sum_{i \in \Omega_k} \Delta w_i^k$. Instead, we must add noise before the cloud server can decrypt the sum. It is also problematic to have a single user in Ω_k to add the noise because this designated user can learn the true sum, violating other users' privacy. Moreover, users may not trust each other and some subset of users may even collude with the cloud server in real-world settings.

We propose to let the users ensure the differential privacy of their intermediate results by themselves. Each user $i \in \Omega_k$ involved in the current iteration would add noise to its submitted intermediate result Δw_i^k before participating in

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 28,2021 at 15:26:30 UTC from IEEE Xplore. Restrictions apply.

the secure aggregation protocol as described in Section V-A. Assume there are at least γ percentage of honest users in Ω_k in each iteration. Then these honest users may add a small amount of noise as long as the total noise in the final sum is large enough to protect individual privacy. Specifically, to generate a total noise of $\mathcal{N}(0, \sigma^2)$ in the sum, each user should independently add a noise of $\mathcal{N}(0, \sigma^2/\gamma|\Omega_k|)$ to their individual results Δw_i^k before applying the secure aggregation protocol in Section V-A. However, since the number of the participating users $|\Omega_k|$ is not known beforehand, each user can add a noise of $\mathcal{N}(0, \sigma^2/\gamma s)$ since $|\Omega_k| \geq s$ as designed in the asynchronous ADMM. It is straightforward to see that the above distributed noise generation scheme can ensure that the final sum will contain sufficient noise from the honest users so as to guarantee differential privacy, while keeping the error of the sum to be small. Formally, we have

Theorem 4. *Our approach achieves (ϵ, δ)-differential privacy at each iteration k for all users in Ω_k when there is at least γ percentage of honest users following the protocol.*

As a comparison, we analyze the utility improvement of combining secure aggregation with differential privacy in our approach. Suppose we do not utilize the secure aggregation protocol so that each user's individual result is revealed to the cloud server in each iteration. Therefore, to guarantee (ϵ, δ)-differential privacy, each user needs to independently add a noise of $\mathcal{N}(0, \sigma_i^2)$ to its intermediate result, where $\sigma_i \geq \sqrt{2\ln(1.25/\delta)}(2/\rho\epsilon)$, according to Lemma 1. Therefore, the sum of the intermediate results from all users in Ω_k would have a noise of $\mathcal{N}(0, \sum_{i\in\Omega_k}\sigma_i^2)$, whose standard deviation is $\sqrt{|\Omega_k|}$ times larger than that of our proposed approach. Therefore, our approach can obtain significant utility improvement compared to the local differential privacy approach when the number of participating users is large.

The overall procedures of the server and the users are summarized in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 Procedure at the server

1: Initialization: choose an initial value of w_0 and broadcast it to all the users. Set $\overline{w}^0 \leftarrow 0$, $\overline{\lambda}^0 \leftarrow 0$, $k \leftarrow 1$ and $\tau_i \leftarrow 0, \forall i \in [n]$;
2: **repeat**
3: **repeat**
4: wait;
5: **until** receive encrypted updates from a set of users Ω_k such that $|\Omega_k| \geq s$ **and** $\max_{i\in[n]\setminus\Omega_k} \tau_i < \tau - 1$;
6: **for** user $i \in \Omega_k$ **do**
7: $\tau_i \leftarrow 0$;
8: **end for**
9: **for** user $i \in [n] \setminus \Omega_k$ **do**
10: $\tau_i \leftarrow \tau_i + 1$;
11: **end for**
12: decrypt $\sum_{i\in\Omega_k} \Delta w_i^k$ and $\sum_{i\in\Omega_k} \Delta\lambda_i^k$ through the secure aggregation protocol in Section V-A;
13: $\overline{w}^k \leftarrow \overline{w}^{k-1} + \frac{\sum_{i\in\Omega_k}\Delta w_i^k}{n}$;
14: $\overline{\lambda}^k \leftarrow \overline{\lambda}^{k-1} + \frac{\sum_{i\in\Omega_k}\Delta\lambda_i^k}{n}$;
15: update w_0^{k+1} by (11);
16: broadcast w_0^{k+1} to all the users in Ω_k;
17: $k \leftarrow k + 1$;
18: **until** stopping criteria is satisfied;

Algorithm 2 Procedure at the i-th user

1: Initialization: set $\lambda_i^0 \leftarrow 0$ and $k_i \leftarrow 0$;
2: **repeat**
3: **repeat**
4: wait;
5: **until** receive update $\hat{w}_0$ from the server;
6: update $w_i^{k_i+1}$ by (12) and obtain $\tilde{w}_i^{k_i+1} := w_i^{k_i+1} + \mathcal{N}(0, \sigma_k^2/\gamma s)$ by noise sampling;
7: compute $\Delta w_i^{k_i+1} \leftarrow \tilde{w}_i^{k_i+1} - \tilde{w}_i^{k_i}$;
8: compute $\Delta\lambda_i^{k_i+1} \leftarrow \tilde{w}_i^{k_i+1} - \hat{w}_0$;
9: send the encrypted $(\Delta w_i^{k_i+1}, \Delta\lambda_i^{k_i+1})$ back to the server through the secure aggregation protocol in Section V-A;
10: $k_i \leftarrow k_i + 1$;
11: **until** stopping criteria is satisfied;

VI. Evaluation Results

In this section, we evaluate the performance of our proposed approach on a real-world dataset for logistic regression analysis. All experiments are conducted in MATLAB on a computer with 2.5 GHz Intel Core i7 CPU and 16 GB RAM.

A. Experimental Setup

Dataset and learning task. The dataset we use is the *Adult* dataset from the UC Irvine Machine Learning Repository [14], which contains demographic information about $48,842$ individuals. The classification task is to determine whether the *annual income* of a person is below or above 50K based on 14 attributes, namely, *age*, *workclass*, *fnlwgt*, *education*, *education-num*, *marital-status*, *occupation*, *relationship*, *race*, *sex*, *capital-gain*, *capital-loss*, *hours-per-week*, and *native-country*. For data preprocessing, we first remove all entries with missing values in this dataset. Then for binary categorial attributes, we set them to be 0 or 1. For the remaining categorical attributes that have more than 2 possible values, we transform these attributes into binary vectors by one-hot encoding. We next normalize the other numeric attributes into the range of $[0, 1]$. Finally, each sample is normalized to ensure that its norm is at most 1. After these preprocessing, our dataset now has 40000 samples, each having a 105-dimensional feature vector and a label with values in $\{-1, 1\}$. Among these samples, 30000 samples will be used as the training dataset while the remaining 10000 samples will be used as the testing dataset. We focus on using logisitc regression as the learning task to predict the *annual income* based on the other attributes of a person using the above dataset. The loss function used

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 28,2021 at 15:26:30 UTC from IEEE Xplore. Restrictions apply.

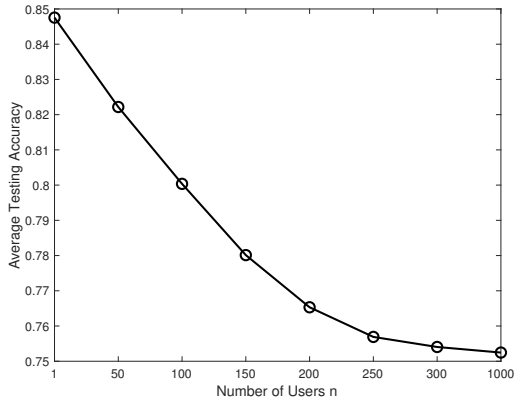

Fig. 3. Centralized vs. Local Approaches.

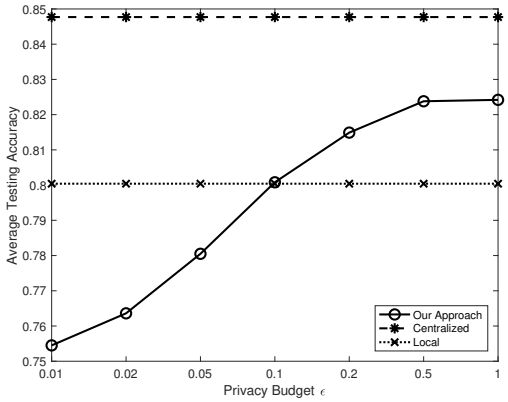

Fig. 4. Accuracy of our approach for different privacy budget with $n = 100$ and $\delta = 10^{-3}$. The accuracy of centralized and local approaches are plotted for comparison.

here is

$$l(h(x_{ij}; w), y_{ij}) = \log\big(1 + \exp\big(-y_{ij} x_{ij}^T w\big)\big). \qquad (17)$$

The regularizer used here is ℓ_2-norm, i.e., $r(w) = (\beta/2)\|w\|_2^2$. With a trained logistic model w, given an attribute vector x, we predict the label y to be 1 if $x^T w \geq 0$ and -1 otherwise. The accuracy of a logistic model is measured by its success rate, which is the fraction of testing samples that are correctly classified using the trained model w.

Parameter setting. In all experiments, the weight of the regularizer β is tuned to be the optimal value. We vary the number of users n between 1, 10, 50, 100, 150, 200, 300, 1000. The local training dataset of each user is randomly initialized and has the same size.

Baseline approaches. We compare our approach with the following two baseline approaches:

- *Centralized approach*: In this approach, all sensing datasets from users are collected at the server for analysis. Obviously, this approach will have the best performance of the model accuracy, but it does not protect user privacy and is privacy-oblivious.
- *Local approach*: Each individual user trains the model only based on its local dataset without collaboration in this approach. Although user privacy is protected in this approach, its model accuracy could be lower than that of the centralized approach.

B. Simulation Results

Collaborative learning benefits. In this part, we first evaluate the benefits of collaborative learning by comparing the centralized approach and local approach. As shown in Fig. 3, as we increases the number of users from 1 (corresponding to the centralized approach) to 1000, the size of the dataset at each user decreases and the average testing accuracy of the obtained model reduces. Therefore, collaborative learning can improve the accuracy of the learning model by training over a larger dataset.

Privacy-accuracy trade-off. Next, we evaluate the effects of different values of privacy parameters ϵ on the accuracy of the collaborative learned model when the number of users $n = 100$, algorithm parameters $s = 100$, total number of master iterations $T = 20$, $\gamma = 1$, and $\delta = 10^{-3}$. In practice, participants can choose the values for these meta-parameters by training on a calibration dataset, e.g., a public dataset that has no privacy implications. Given the randomized nature of our proposed differentially private algorithm, we run all simulations 100 times to compute the average results. Fig. 4 shows the trade-off between accuracy and privacy. The x-axis represents the privacy budget ϵ and the y-axis denotes the accuracy of the obtained global model on the testing dataset. As expected, a larger ϵ value results in higher accuracy while providing lower differential privacy guarantee. However, our proposed scheme can achieve almost the same results of the centralized approach and outperform the local approach when ϵ is reasonably large (e.g., 0.1 in our simulations). Hence our proposed approach can achieve high accuracy without sacrificing too much in privacy guarantee.

VII. Related Work

The privacy issue of crowdsensing has been investigated in the literature. A large number of existing solutions [15], [16], [17] focus on simple aggregate queries such as COUNT ans SUM in crowdsensing applications. However, none of these methods apply to our problem, as learning often involves solving a complex optimization problem, which is much more challenging than COUNT/SUM queries and their derivatives. A few work has considered privacy-preserving learning in crowdsensing. For instance, Liu et al. [18] consider privacy-preserving collaborative learning for the mobile setting and design a system based on sensing data perturbation to preserve the privacy of mobile users. Gong et al. [19] study the application of mobile health monitoring and propose a privacy-preserving scheme based on distributed optimization and secure multi-party computation. However, these solutions do not use the rigorous notation of differential privacy as their privacy-preserving goal. Perhaps the most related work to ours is [20], where the authors propose a general framework for machine

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 28,2021 at 15:26:30 UTC from IEEE Xplore. Restrictions apply.

learning with smart devices from crowdsensing data. The key idea of their work is to utilize stochastic gradient descent (SGD) to design a distributed learning algorithm and Laplace mechanism to achieve differential privacy during parameter sharing. Gradient and subgraident methods such as SGD, although computationally simple at each user, generally require too many iterations (and hence communications) to converge [21], which is not feasible in cases where communication is expensive for participating users such as mobile devices. In comparison, our distributed algorithm based on ADMM often converges to good accuracy within a few tens of iterations in statistical and machine learning problems and is more suitable for communication-constrained scenarios. Moreover, by using ADMM, the overall regularized ERM problem is decomposed into multiple smaller problems, which can then be solved by each individual user based on their own optimal solvers.

Our work is also related to the literature of privacy-preserving multi-party machine learning, which is attracting increasing attention in the big data era. Techniques from secure multi-party computation and homomorphic encryption have been developed for specific learning tasks such as linear regression [22], [23]. However, such techniques typically involve high computation and communication overhead and are not suitable for crowdsensing where user devices could be resource-constrained. ADMM-based approaches [24], [25] have also been developed in this setting, however they cannot protect the differential privacy of the learning results and only use synchronous ADMM. Differential privacy has also been applied to complex learning tasks such as deep learning [26], [27]. These problems or solutions are orthogonal to ours that focuses on ERM.

VIII. Conclusion

This paper focuses on privacy-preserving collaborative learning for crowdsensing. We have proposed a new distributed asynchronous learning scheme based on ADMM with rigorous privacy guarantee. Our methodology works for many popular machine learning models that fit into the regularized ERM framework and preserves the differential privacy of crowdsensing users' data while achieving high accuracy of the resulting model. With the proposed approach, users can enjoy the benefits of crowdsensing without giving up their privacy. In the future, we plan to extend the proposed methodology to more complex learning tasks such as deep learning and multi-task learning.

References

[1] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, 2011.

[2] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," Google Research Blog, April 2017. [Online]. Available: https://research.googleblog.com/2017/04/federated-learning-collaborative.html

[3] White House Report, "Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy," *Journal of Privacy and Confidentiality*, vol. 4, pp. 95–142, 2012.

[4] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *Journal of Machine Learning Research*, vol. 12, pp. 1069–1109, Mar. 2011.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

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

[7] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 2014, pp. 1054–1067.

[8] Apple's differential privacy is about collecting your data - but not your data. [Online]. Available: https://www.wired.com/2016/06/apples-differential-privacy-collecting-data/

[9] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.

[10] S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," in *Proceedings of the 20th international conference on World wide web*. ACM, 2011, pp. 607–614.

[11] R. Zhang and J. Kwok, "Asynchronous distributed admm for consensus optimization," in *ICML*, 2014, pp. 1701–1709.

[12] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1322–1333.

[13] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[14] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[15] H. Jin, L. Su, H. Xiao, and K. Nahrstedt, "Inception: incentivizing privacy-preserving data aggregation for mobile crowd sensing systems." in *MobiHoc*, 2016, pp. 341–350.

[16] X. Jin and Y. Zhang, "Privacy-preserving crowdsourced spectrum sensing," in *IEEE INFOCOM*, 2016.

[17] E. Shi, H. Chan, E. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in *NDSS*. Internet Society, 2011.

[18] B. Liu, Y. Jiang, F. Sha, and R. Govindan, "Cloud-enabled privacy-preserving collaborative learning for mobile sensing," in *SenSys*. ACM, 2012, pp. 57–70.

[19] Y. Gong, Y. Fang, and Y. Guo, "Private data analytics on biomedical sensing data via distributed computation," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 13, no. 3, pp. 431–444, 2016.

[20] J. Hamm, A. C. Champion, G. Chen, M. Belkin, and D. Xuan, "Crowd-ml: A privacy-preserving learning framework for a crowd of smart devices," in *ICDCS*, 2015, pp. 11–20.

[21] J. F. Mota, J. M. Xavier, P. M. Aguiar, and M. Puschel, "D-admm: A communication-efficient distributed algorithm for separable optimization," *IEEE Transactions on Signal Processing*, vol. 61, no. 10, pp. 2718–2723, 2013.

[22] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *IEEE Symposium on Security and Privacy*, 2013, pp. 334–348.

[23] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *IEEE Symposium on Security and Privacy*, 2017, pp. 19–38.

[24] K. Xu, H. Yue, L. Guo, Y. Guo, and Y. Fang, "Privacy-preserving machine learning algorithms for big data systems," in *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*. IEEE, 2015, pp. 318–327.

[25] K. Xu, Y. Guo, L. Guo, Y. Fang, and X. Li, "My privacy my decision: Control of photo sharing on online social networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 2, pp. 199–210, 2017.

[26] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *CCS*. ACM, 2015, pp. 1310–1321.

[27] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *CCS*. ACM, 2016, pp. 308–318.

Authorized licensed use limited to: University of Texas at San Antonio. Downloaded on June 28,2021 at 15:26:30 UTC from IEEE Xplore. Restrictions apply.