Privacy-Preserving Distributed k-Means Clustering over Arbitrarily Partitioned Data*

Geetha Jagannathan
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ, 07030, USA
gjaganna@cs.stevens.edu

Rebecca N. Wright
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ, 07030, USA
rwright@cs.stevens.edu

ABSTRACT

Advances in computer networking and database technologies have enabled the collection and storage of vast quantities of data. Data mining can extract valuable knowledge from this data, and organizations have realized that they can often obtain better results by pooling their data together. However, the collected data may contain sensitive or private information about the organizations or their customers, and privacy concerns are exacerbated if data is shared between multiple organizations.

Distributed data mining is concerned with the computation of models from data that is distributed among multiple participants. Privacy-preserving distributed data mining seeks to allow for the cooperative computation of such models without the cooperating parties revealing any of their individual data items. Our paper makes two contributions in privacy-preserving data mining. First, we introduce the concept of arbitrarily partitioned data, which is a generalization of both horizontally and vertically partitioned data. Second, we provide an efficient privacy-preserving protocol for k-means clustering in the setting of arbitrarily partitioned data.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining

General Terms

Algorithms; Security.

1. INTRODUCTION

Many organizations collect large amounts of data about their clients and customers. Such data was initially used only for record keeping. There was soon a realization that these large collections of data could be "mined" for knowledge that could improve the performance of the organization. Well known data-mining tasks include clustering, prediction, association rule mining and outlier detection [13]. Data mining has been used extensively, for purposes including biomedical and DNA data analysis [8], financial data analysis [3], identification of unusual patterns, and analysis of telecommunications data [12].

While much data mining occurs on data within an organization, it is quite common to use data from multiple sources in order to yield more precise or useful knowledge. However, privacy and secrecy considerations can prohibit organizations from being willing or able to share their data with each other. Privacy-preserving data mining, introduced by Agarwal and Srikant [1] and Lindell and Pinkas [9], arose as a solution to this problem by allowing parties to cooperate in the extraction of knowledge without any of the cooperating parties having to reveal their individual data items to each other or any other parties.

Techniques from secure multiparty computation [7] form one approach to privacy-preserving data mining. Yao's general protocol for secure circuit evaluation [18] can be used, in theory, to solve any two-party privacy-preserving distributed data mining problem. However, since data mining usually involves millions or billions of data items, the communication cost of this protocol renders it impractical for these purposes. This has led to the search for problem-specific protocols that are efficient in terms of communication complexity. In many cases (including our solution described in this paper), the more efficient solutions still make use of a general solution such as Yao's, but only to carry out a much smaller portion of the computation. The rest of the computation uses other methods to ensure the privacy of the data. A complementary approach to privacy-preserving data mining uses randomization techniques. Although such solutions tend to be more efficient than cryptographic solutions, they are generally less private or less accurate.

Privacy-preserving data mining solutions have been presented both with respect to horizontally and vertically partitioned databases, in which either different data objects with the same attributes are owned by each party, or different attributes for the same data objects are owned by each party, respectively. We introduce the notion of arbitrarily partitioned data, which generalizes both horizontally and vertically partitioned data. In arbitrarily partitioned data, different attributes for different items can be owned by either party. Although extremely "patchworked" data is

*This work was supported by the National Science Foundation under Grant No. CCR-0331584.

unlikely in practice, one advantage of considering arbitrarily partitioned data is that protocols in this model apply both to horizontally and vertically partitioned data, as well as to hybrid that are mostly, but not completely, vertically or horizontally partitioned. (These concepts are defined more precisely in Section 2.)

In this paper, we provide a privacy-preserving solution to an important data mining problem, that of clustering data. Privacy-preserving clustering has been previously addressed [14, 16]. Oliviera and Zaiane [14] addressed the privacy-preserving data clustering problem using randomization techniques. Vaidya and Clifton [16] presented a privacy-preserving k-means clustering protocol on vertically partitioned data using cryptographic techniques.

Informally, a clustering algorithm partitions a set of objects into clusters so that all objects within any cluster are "similar" or "close", while objects in different clusters are "dissimilar". Clustering has been successfully applied in many domains. In image processing, clustering algorithms are used for image segmentation, which is the problem of distinguishing objects from the background [17]. In bioinformatics, clustering has been used to group genes with similar expression profiles [15]. Astronomers have used clustering to create catalogs of objects in the sky [4].

The *k-means clustering algorithm* (also known as *Lloyd's algorithm*) [5, 10, 11] is a well known iterative algorithm that successively refines potential clusters in an attempt to minimize the k-means *objective function*, which measures the "goodness" of a given clustering. We present a privacy-preserving protocol for k-means clustering in the setting of arbitrarily partitioned data distributed between two parties. Our protocol is efficient and provides cryptographic privacy protection. In particular, our protocol provides the first privacy-preserving solution to k-means clustering for horizontally partitioned data. We also provide an analysis of the performance and privacy of our solution.

In Section 2, we review the k-means clustering algorithm and provide preliminary definitions that are used in the rest of the paper. Section 3 describes our privacy-preserving protocol for k-means clustering when the data is arbitrarily partitioned between two parties. The analysis of the communication complexity and privacy of our protocol is given in Section 4.

2. DEFINITIONS

In this section, we briefly review clustering and the k-means clustering algorithm, discuss the privacy properties we seek to achieve, and describe the concept of arbitrarily partitioned data.

2.1 Clustering and the k-Means Algorithm

We briefly review the k-means clustering algorithm. (For a more detailed description, see [13].) The k-means clustering algorithm partitions the set D of objects into the specified number k of disjoint subsets, called *clusters*. The clustering depends on a well defined notion of the distance between a given pair of objects. In our case, we assume data objects are elements of $\mathbb{R}^d$, and we define the distance between two objects as their Euclidean distance. Each cluster is represented by its *center*, which is the attribute-wise average of all objects in the cluster.

The k-means algorithm proceeds iteratively, successively refining k potential clusters until a specified termination condition is reached. Initially, the algorithm arbitrarily selects k of the objects in D as k initial candidate cluster centers. Each object in D is assigned to the cluster whose center is closest to it. Subsequently, the clusters centers and corresponding cluster assignments are modified with the intention of reducing the average distance of each object to its closest candidate cluster center. In each iteration, each object in the database D is reassigned to the cluster whose candidate center is closest. The current center of each cluster is then recomputed on the basis of the modified set of the objects in the cluster. This iterative process continues until a specified termination condition is met. A commonly used condition terminates the process when the change in the average distance of each object to its closest cluster center is small. Another commonly used condition is the absence of any change in cluster composition. In this paper, we use a commonly used termination condition, terminating when the change in the centers of all clusters falls below a prespecified threshold value. This version is illustrated in Figure 1.

Input: Database D, integer k
Output: Cluster centers $\mu_1 \dots \mu_k$

1. Arbitrarily select k objects from D as initial cluster centers $\mu'_1 \dots \mu'_k$.
2. Repeat
   (a) $(\mu_1 \dots \mu_k) = (\mu'_1 \dots \mu'_k)$
   (b) Assign each object $d_i \in D$ to the cluster whose center is closest.
   (c) Recompute the centers of the k clusters as $\mu'_1 \dots \mu'_k$.

   Until $(\mu_1 \dots \mu_k)$ is close to $(\mu'_1 \dots \mu'_k)$

Figure 1: k-means clustering algorithm

2.2 Arbitrarily Partitioned Data

In the two-party distributed data setting, two parties (call them Alice and Bob) hold data forming a (virtual) database consisting of their joint data. More specifically, the virtual database $D = \{d_1, d_2, \dots, d_n\}$ consists of n *objects*, or *records*. Each object d_i is described by the values of ℓ numeric attributes. We denote the values of the ℓ attributes of object d_i by $(x_{i,1}, x_{i,2}, \dots, x_{i,\ell})$. When convenient, we sometimes abuse notation and refer to this vector of values as the set $d_i = \{x_{i,1}, x_{i,2}, \dots, x_{i,\ell}\}$.

We introduce in this paper the concept of *arbitrarily partitioned data*, illustrated in Figure 2. In arbitrarily partitioned data, there is not necessarily a simple pattern of how data is shared between the parties. For each d_i, Alice knows the values for a subset of the attributes, and Bob knows the values for the remaining attributes. That is, each d_i is partitioned into disjoint subsets d_i^A and d_i^B such that Alice knows d_i^A and Bob knows d_i^B. We emphasize that the set of attributes whose values are known to Alice for some object d_i does not have to equal the set of attributes whose values are known to Alice for some other object d_j $(i \neq j)$. In particular, it is possible that $d_i^A = \emptyset$ or $d_i^A = d_i$. That is, a given object may be "completely owned" by Bob or by Alice. If an object is completely owned by Alice (respectively, Bob), its existence is unknown to Bob (respectively, Alice).

Both Alice and Bob know the names of all attributes. We assume that both parties either know the total number n of objects or they know the range of values for all of the attributes. Some data values may be known to both parties; we consider such values to be owned by one party, who will be responsible for handling the data value.

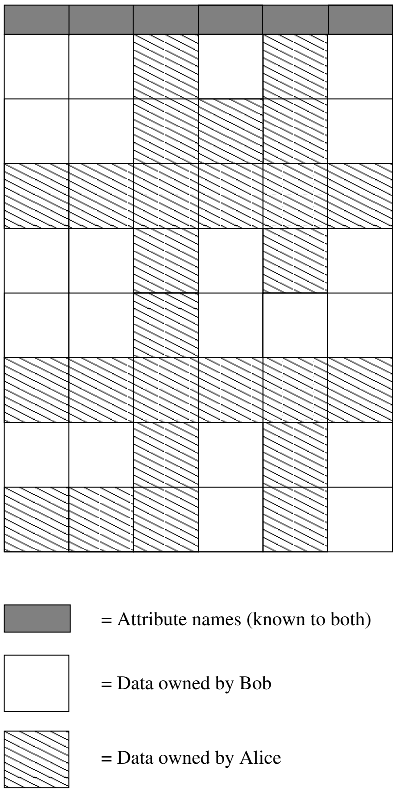

Figure 2: Arbitrarily partitioned data

Clearly, both horizontally partitioned data and vertically partitioned data can be viewed as specific cases of arbitrarily partitioned data. Specifically, in horizontally partitioned data, each object in the virtual database is completely owned by Alice or completely owned by Bob. In vertically partitioned data, each attribute is "completely owned" by either Alice or Bob except perhaps for some common "data handle". As previously mentioned, although extremely "patchworked" data is unlikely in practice, the generality of this model can make it better suited to practical settings in which data may be mostly, but not completely, vertically or horizontally partitioned.

2.3 Privacy Properties

Our desired outcome is that clusters are computed on Alice and Bob's joint data. Alice should learn the cluster number for each data object she owns completely, and Bob should learn the cluster number for each data object that he owns completely. For objects that both have attributes for, they should both learn the cluster number. In addition, the parties should either learn random shares (defined below) of the final cluster centers, or, depending on the desired outcome for the particular application, one or both party should learn the actual final cluster centers.

In an ideal world, Alice and Bob would have access to a trusted third party who could perform the necessary calculations. Alice and Bob would securely send their data to this trusted party, which would then compute the cluster to which each object is assigned using the appropriate clustering algorithm, and send the desired information to the party that owns the object. However, trusted third parties are hard to find in the real world, and even then, such trust may be misplaced. In this work, we do not rely on a third party. Instead, we provide algorithms by which Alice and Bob can carry out the functionality that would be provided by the trusted third party *without* actually requiring such a party or requiring the parties to send their data to each other.

In this paper, we assume that both Alice and Bob are *semi-honest*. That is, both parties faithfully follow their specified protocols, but we assume they record intermediate messages in an attempt to infer as much information about the other party's data as possible. Additional cryptographic techniques, such as zero knowledge proofs, can be used to provide privacy even against malicious behavior, but typically at a significant performance cost.

As we discuss further in Section 4, our solution leaks some additional information beyond just the cluster numbers, in that it reveals some information about the progress of the computation as it is carried out. This information does not appear to be useful except perhaps in degenerate cases, and some protection can be provided by having the initial choice of candidate cluster centers be chosen randomly.

2.4 Cryptographic Primitives

We describe the various cryptographic primitives used in this paper.

RANDOM SHARES. According to our privacy requirements, both parties should get their final output, but all the values computed in the intermediate steps of the algorithm should be unknown to both parties. So in our paper all the intermediate values that are supposed to be unknown to the parties, such as the candidate cluster centers at the end of each iteration, are computed throughout the algorithm are shared as uniformly distributed random values between the two parties. Their sum is the actual intermediate value (i.e., the ones that would be computed using the centralized k-means algorithm on the union of the data).

More formally, we say that *Alice and Bob have random shares of a value x drawn from a field F of size N* (usually abbreviated to *Alice and Bob have random shares of x*) to mean that Alice knows a value $a \in F$ and Bob knows a value $b \in F$ such that

$$(a + b) \bmod N = x,$$

where a and b are uniformly random in field F. Note that the requirement that $x \in F$ implies that $(a + b) \bmod N$ is actually equal to x, not simply that $a+b$ and x are congruent modulo N.

Throughout the paper, we assume that a finite field F of suitable size N is chosen such that all computations can be done in that field, and all computations throughout the remainder of the paper take place in F.

SECURE SCALAR PRODUCT PROTOCOL. Alice has a vector $X = (x_1, ..., x_n)$ and Bob has a vector $Y = (y_1, ..., y_n)$. They wish to securely compute the scalar product as $s_A + s_B = X.Y$ where s_A and s_B are the random shares of Alice and Bob respectively. (Recall that all computations in the paper are carried out modulo N.) We use the private homomorphic scalar product protocol given by Goethals et al. [6]. This protocol uses a (semantically secure) homomorphic encryption scheme where Alice generates a private and public key pair and sends the public key to Bob. Alice computes the encryptions of $x_1, ..., x_n$ and sends them to Bob. Bob chooses a random value s_B and uses the properties of the homomorphic encryption to compute the encryption of $X.Y - s_B$, and sends to Alice. Alice decrypts and obtains $s_A = X.Y - s_B$. This protocol is secure when the parties are semi-honest.

YAO'S CIRCUIT EVALUATION PROTOCOL. When Alice has a vector $X = (x_1, ..., x_n)$ and Bob has $Y = (y_1, ..., y_n)$, we use Yao's circuit evaluation protocol [18] to securely compute the index j such that $x_j + y_j$ is minimum. We also use Yao's circuit evaluation protocol to compute $\frac{a+b}{m+n}$ where Alice knows a and m and Bob knows b and n. Alternatively the Bar-Ilan and Beaver's protocol [2] can also be used to compute $(n+m)^{-1}$ and then a secure multiplication protocol can be used to compute the product.

3. PRIVACY-PRESERVING k-MEANS CLUSTERING PROTOCOL

We now describe our privacy-preserving k-means clustering protocol. Alice and Bob share a data set in an arbitrary partition, as explained in Section 2.2. They wish to learn the k-means clustering of their joint data, without revealing their data to each other or anyone else. The final output of the algorithm should be an assignment of a cluster number between 1 and k to each object. If an object is shared, then both parties learn the assignment; otherwise, only the party who owns the object gets the assignment. The algorithm can terminate with the final mean of each cluster randomly shared by both parties, or if desired, one or both parties can learn the actual final centers.

As described in Section 2.1, the k-means clustering algorithm is an iterative algorithm that successively refines candidate cluster centers. Initially, k candidate cluster centers are chosen from the data points in D. This can be done in one of several ways: for example, they can be taken as a pre-specified set of points, such as the first k data points, they can be chosen randomly by one or the other party, or they can be chosen randomly with randomness introduced by both parties. As we discuss in Section 4, the latter method provides the most robustness against certain kinds of potential privacy leaks.

In each iteration of the algorithm, each object is assigned to the closest candidate cluster center. Then, new candidate cluster centers are determined based on the actual centers of the points as they have been assigned. This procedure is repeated until a specified termination condition is reached. Our privacy-preserving distributed version follows the same iterative structure, but protects the data and intermediate values from being learned by the parties. This is done using interaction, privacy-preserving subprotocols and random sharings as we now describe.

In each iteration of the privacy-preserving protocol, the candidate cluster centers are calculated as a random sharing between the two parties. Let ℓ denote the number of attributes and $d_i = (x_{i,1}, \ldots, x_{i,\ell})$ for $1 \leq i \leq n$ denote the i^{th} object in the database. Let μ_j^A for $1 \leq j \leq k$ denote Alice's share of the j^{th} mean, μ_j^B for $1 \leq j \leq k$ denote Bob's share. The candidate cluster centers are given by $\mu_j^A + \mu_j^B$ for $1 \leq j \leq k$. For each object d_i, Alice and Bob securely compute the distances $\mathsf{dist}(d_i, \mu_j)$ for $1 \leq j \leq k$, between the object and each of the k cluster centers. The result of the distance calculation is learned as random shares between Alice and Bob. Using these random shares, they then securely compute the closest cluster for each object in the database.

At the end of each iteration, Alice and Bob learn the cluster assignment for the objects, as follows. If an object d_i is shared by both Alice and Bob, then both of them learn the cluster to which the object belongs. If not, it is revealed only to the party who completely owns d_i. The iterative process is repeated until the change in the centers of all clusters falls below a certain specified threshold value.

When the termination condition is reached, the parties may wish to send each other their cluster center shares so that they can learn the actual cluster centers of the joint data. Whether this is desirable or not depends on the particular application for which the clustering results are intended to be used.

Figure 3 illustrates the overall privacy-preserving k-means clustering protocol. In the remainder of this section, we describe in detail each of the subprotocols used in the privacy-preserving k-means clustering algorithm. In Section 3.1, we present a secure protocol that computes the cluster assignment for each object. In Sections 3.2 and 3.3, we present secure protocols for recomputing the cluster centers and for iteration termination, respectively.

Input: Database D consisting of n objects, integer k denoting the number of clusters
Output: Assignment of the cluster numbers to the objects

1. Randomly select k objects from D as initial cluster centers $\mu'_1 \ldots \mu'_k$.
2. Randomly share the cluster centers between Alice and Bob:
$$\text{Alice's share} = (\alpha_1^A \ldots \alpha_k^A)$$
$$\text{Bob's share} = (\alpha_1^B \ldots \alpha_k^B)$$
3. Repeat
   (a)
   $$(\mu_1^A \ldots \mu_k^A) = (\alpha_1^A \ldots \alpha_k^A)$$
   $$(\mu_1^B \ldots \mu_k^B) = (\alpha_1^B \ldots \alpha_k^B)$$
   (b) For each d_i in D
      i. Run the secure closest cluster protocol
      ii. Assign to d_i the closest cluster
   (c) Alice and Bob securely recompute random shares of the centers of the k clusters as $(\alpha_1^A \ldots \alpha_k^A)$ and $(\alpha_1^B \ldots \alpha_k^B)$ respectively.

   Until $(\mu_1^A + \mu_1^B \ldots \mu_k^A + \mu_k^B)$ is close enough to $(\alpha_1^A + \alpha_1^B \ldots \alpha_k^A + \alpha_k^B)$

Figure 3: Privacy-preserving k-means clustering protocol over arbitrarily partitioned data

3.1 Secure Protocol to Compute Closest Cluster

This subprotocol takes an arbitrarily partitioned object and k randomly shared candidate cluster centers as its input, and outputs to the appropriate party or parties the closest cluster to which the object belongs. It reveals no other additional information.

Consider an object $d_i = (x_{i,1}, \ldots, x_{i,\ell})$. This object can be owned by Alice or Bob or shared by both. Suppose that the object is shared by both Alice and Bob. Without loss of generality, assume $x_{i,p_1}, \ldots, x_{i,p_s}$ belong to Alice and the rest of the $\ell - s$ attributes belong to Bob. To compute the closest cluster the protocol first computes the distance between the object d_i to each of the k clusters. For each cluster $1 \le j \le k$, we compute the distance $\mathsf{dist}(d_i, \mu_j)$ between the object d_i to the jth-cluster mean μ_j.

$(\mathsf{dist}(d_i, \mu_j))^2 = (x_{i,1}-\mu_{j,1})^2+(x_{i,2}-\mu_{j,2})^2+\cdots+(x_{i,\ell}-\mu_{j,\ell})^2$.

Since $\mu_{j,t} = \mu^A_{j,t} + \mu^B_{j,t}$, where $\mu^A_{j,t}$ is Alice's share of the mean and $\mu^B_{j,t}$ is Bob's share of the mean,

$$(\mathsf{dist}(d_i, \mu_j))^2 = (x_{i,1}-(\mu^A_{j,1}+\mu^B_{j,1}))^2+\cdots+(x_{i,l}-(\mu^A_{j,\ell}+\mu^B_{j,\ell}))^2$$

$$= \sum_{m=1}^{\ell} x^2_{i,m} + \sum_{m=1}^{\ell} (\mu^A_{j,m})^2 + \sum_{m=1}^{\ell} (\mu^B_{j,m})^2 + 2\sum_{m=1}^{\ell} \mu^A_{j,m}\mu^B_{j,m}$$

$$-2\sum_{m=1}^{\ell} \mu^A_{j,m} x_{i,m} - 2\sum_{m=1}^{\ell} x_{i,m}\mu^B_{j,m}$$

To privately compute the term $\sum_{m=1}^{\ell} x^2_{i,m}$ over the shared data, Alice computes the sum that involves the components $x_{i,p_1}, \ldots, x_{i,p_s}$, while Bob computes the sum of the rest of the components. The second term $\sum_{m=1}^{\ell} (\mu^A_{j,m})^2$ is completely computed by Alice and similarly the third term is computed by Bob. They then use a secure scalar product protocol, such as the one described in [6], to compute random shares of the last three terms. Since a sum involving random shares results in random shares, we have

$$(\mathsf{dist}(d_i, \mu_j))^2 = \alpha_{i,j} + \beta_{i,j}$$

where $\alpha_{i,j}$ is the random share known to Alice and $\beta_{i,j}$ is known to Bob where

$$\alpha_{i,j} = \sum_{m=1}^{s} x^2_{i,p_m} + \sum_{m=1}^{\ell} (\mu^A_{j,m})^2 + a_j + c_j + e_j$$

$$\beta_{i,j} = \sum_{m=1}^{\ell-s} x^2_{i,q_m} + \sum_{m=1}^{\ell} (\mu^B_{j,m})^2 + b_j + d_j + f_j$$

$$a_j + b_j = 2\sum_{m=1}^{\ell} \mu^A_{j,m}\mu^B_{j,m} \mod N$$

$$c_j + d_j = -2\sum_{m=1}^{\ell} \mu^A_{j,m} x_{i,m} \mod N$$

$$e_j + f_j = -2\sum_{m=1}^{\ell} \mu^B_{j,m} x_{i,m} \mod N$$

and N is the size of the chosen finite field.

Alice has a k-length vector $A = (\alpha_{i,1}, \ldots, \alpha_{i,k})$ and Bob has $B = (\beta_{i,1}, \ldots, \beta_{i,k})$. They securely compute the index j such that $\alpha_{i,j} + \beta_{i,j}$ is minimum using Yao's circuit evaluation [18]. Since k is typically quite small, particularly in comparison to the number of data items, the overhead this requires should be sufficiently small. The object d_i is assigned to cluster j.

Communication and Computational Complexity. For each object d_i, this protocol computes the closest cluster. Each object has ℓ components. Let us assume that each component is represented by c bits. Communication is involved when the two parties engage in the secure scalar product computation to compute the distance between an object d_i and each of the cluster centers. For each distance computation the scalar product protocol is invoked three times between vectors of length ℓ. The communication complexity of each scalar product protocol is $O(c\ell)$. Hence it requires a communication of $O(kc\ell)$ bits to compute the distance of the object d_i from all the k centers.

Yao's circuit evaluation [18] requires communication of $O(c)$ bits per comparison and hence $O(\ell c)$ bits for computing the minimum component. The total communication complexity for one invocation of the closest cluster protocol is $O(\ell ck))$.

The estimation of the computational complexity involves counting the number of encryptions, decryptions and multiplications for Alice and Bob. To compute the closest cluster for each object the scalar product protocol is executed for each cluster three times between vectors of length ℓ. For each execution of the scalar product protocol Alice performs ℓ encryptions and one decryption. Bob performs ℓ exponentiations and one encryption. Alice performs at most 2ℓ multiplications and Bob performs at most 3ℓ multiplications. Since there are k clusters the total computational complexity for computing the closest cluster for an object is $O(k\ell)$ encryptions and at most $O(k\ell)$ multiplications for Alice and $O(k\ell)$ exponentiations and at most $O(k\ell)$ multiplications for Bob. The use of the secure scalar product protocol described in [6] increases the computational complexity. But the current hardware and optimization tricks makes the computational complexity for both Alice and Bob tolerable (see discussion in [6]).

Privacy. The above protocol securely computes the index of the closest cluster for every object. The only information revealed is the output, which is the cluster index. The distance between each object d_i to each of the k candidate centers is available only as a random share between Alice and Bob. Hence each cannot obtain any information about the other party's data. For each distance computation, the communication between Alice and Bob occurs through the secure scalar product protocol. The scalar product protocol leaks no information and returns only a random share to both Alice and Bob. Thus the distance function is computed as a random sharing between Alice and Bob without leaking any information. Finally, Yao's circuit evaluation protocol securely computes the closest cluster. The output of the protocol is an assignment of the closest cluster of a given point. This is the only information available to the parties at the end of the protocol when a trusted third party is used.

3.2 Secure Protocol for Recomputing the Mean

At the end of each iteration Alice and Bob learns his or her share of the cluster centers. The next iteration requires recomputing each of the k-cluster centers. Let us assume that Alice has objects $d^A_{i_1}, \ldots, d^A_{i_p}$ and Bob has objects $d^B_{i_1}, \ldots, d^B_{i_q}$, for each cluster $1 \le i \le k$. Each of the d^A_i and d^B_i is an ℓ-tuple, where $d^A_{i,j}$ and $d^B_{i,j}$ each denote the jth coordinate of the corresponding ℓ-tuple.

Alice calculates the shares s_j and n_j for $1 \le j \le \ell$, where $s_j = \sum_{r=1}^{p} d^A_{i_r,j}$ and n_j denotes the number of objects in $d^A_{i_1}, \ldots, d^A_{i_p}$ for which Alice has the values for attribute A_j. If Alice does not have the value for attribute j for some object d^A_i, she treats it as zero.

Similarly, Bob calculates the shares t_j and m_j for $j, 1 \le j \le \ell$, where $t_j = \sum_{r=1}^{q} d^B_{i_r,j}$ and m_j denotes the number of objects in $d^B_{i_1}, \ldots, d^B_{i_q}$ for which Bob has the values for attribute A_j. Again, if Bob does not have the value for attribute j for some object d_i, he treats it as zero in the computation of the sum. The jth component of the ith cluster center is given by $\mu_{i,j} = (s_j + t_j)/(n_j + m_j)$.

Since this involves only four values, it can be computed efficiently using Yao's [18] circuit evaluation protocol. We can also use Bar-Ilan and Beaver's protocol [2] to compute the inverse of $(n_j + m_j)^{-1}$ as $u + v$ where Alice learns u and Bob learns v. The secure multiplication protocol can then be used to compute $a + b = (s_j + t_j)(u + v)$. Alice learns a and Bob learns b.

Communication and Computational Complexity. Each component of the mean requires communication of $O(c)$ bits. Hence the communication complexity to recalculate the mean is $O(k\ell c)$. The computational complexity is $O(k\ell c)$.

Privacy. The protocols we use to recalculate the centers are secure and they do not leak any information. Each party learns only a random share of each mean and thus cannot infer any information about the centers of the clusters.

3.3 Secure Protocol for Termination of Iterations

The k-means clustering algorithm is an iterative algorithm. At the end of every iteration, the cluster centers are recalculated. The iterative process is terminated when there are no substantial improvement in the approximations. In our case, the algorithm terminates when the Euclidean distance between the cluster centers between two consecutive iterations is less than a specified value ϵ. Denote Alice's share of cluster centers at the end of the ith iteration by $\mu_1^{A,i}, \ldots, \mu_k^{A,i}$ and Bob's share by $\mu_1^{B,i}, \ldots, \mu_k^{B,i}$. Similarly, denote Alice's share of cluster centers at the end of the $(i+1)$st iteration by $\mu_1^{A,i+1}, \ldots, \mu_k^{A,i+1}$ and Bob's share by $\mu_1^{B,i+1}, \ldots, \mu_k^{B,i+1}$. Each μ_i is an ℓ-tuple and is denoted by $(\mu_{i,1}, ..., \mu_{i,\ell})$. For $1 \le j \le k$, we securely compute the square of the distances:

$$
\begin{aligned}
&(\mathsf{dist}(\mu_j^{A,i+1} + \mu_j^{B,i+1}, \mu_j^{A,i} + \mu_j^{B,i}))^2 \\
&= (\mathsf{dist}((\mu_{j,1}^{A,i+1} + \mu_{j,1}^{B,i+1}, ..., \mu_{j,\ell}^{A,i+1} + \mu_{j,\ell}^{B,i+1}), \\
&\quad (\mu_{j,1}^{A,i} + \mu_{j,1}^{B,i}, ..., \mu_{j,\ell}^{A,i} + \mu_{j,\ell}^{B,i})))^2 \\
&= \sum_{m=1}^{\ell} (\mu_{j,m}^{A,i} + \mu_{j,m}^{B,i})^2 + \sum_{m=1}^{\ell} (\mu_{j,m}^{A,i+1} + \mu_{j,m}^{B,i+1})^2 - \\
&\quad 2 \sum_{m=1}^{\ell} ((\mu_{j,m}^{A,i+1} + \mu_{j,m}^{B,i+1})(\mu_{j,m}^{A,i} + \mu_{j,m}^{B,i})) \\
&= \alpha_j + \beta_j,
\end{aligned}
$$

where α_j is Alice's share of the distance and β_j is Bob's share. These are random shares. Alice and Bob have a k-length vector $(\alpha_1, \ldots, \alpha_k)$ and $(\beta_1, \ldots, \beta_k)$ respectively. They communicate with each other and securely check if $\alpha_j + \beta_j < \epsilon$ for $1 \le j \le k$. As before, since the amount of input involved is small, this can be securely and efficiently implemented using Yao's protocol.

Communication and Computational Complexity. To check if the algorithm terminates, Alice and Bob has to check that the Euclidean distance between two consecutive iterations is less than ϵ for all the k-cluster centers. This involves executing the scalar product protocol four times to obtain the random shares of the distance vectors and then applying the Yao's protocol. As explained in Section 3.1, it requires a communication of $O(kc\ell)$ bits to compute the distance between two consecutive iterations of all the k centers. Yao's circuit evaluation involves $O(kc\ell)$ bits of communication to check if all the k distances are less than ϵ. The total computational complexity is $O(k\ell)$ encryptions and at most $O(k\ell)$ multiplications for Alice and $O(k\ell)$ exponentiations and at most $O(k\ell)$ multiplications for Bob. The discussion is similar to the one described in Section 3.1.

Privacy. The scalar product protocol and Yao's circuit evaluation protocol are secure and they leak no information. Therefore, the only information that Alice and Bob can obtain at the end of the execution of the protocol is the output whether the iterations can be terminated or not.

4. PERFORMANCE ANALYSIS

Putting all the subprotocols together as shown in Figure 3 yields our complete privacy-preserving k-means clustering protocol over arbitrarily partitioned data. We have discussed the complexity and privacy of each component; we now discuss the overall complexity and privacy.

Communication and Computational Complexity. As described in Section 3.1, the two parties need to communicate $O(ck\ell)$ bits to determine the closest cluster for each point. Here c is the maximum number of bits needed to represent each component of an object or its encryption and ℓ denotes the number of attributes. Hence the total communication complexity to assign cluster numbers in each iteration is $O(nck\ell)$ bits. At the end of each iteration, the protocol recomputes the center of each of the k clusters, each of which is a ℓ-length vector. It takes $O(c)$ bits to compute each component of a single center, so the communication complexity to compute all k centers is $O(c\ell k)$. Additionally, it takes $O(c\ell k)$ bits of communication for Alice and Bob to securely check the termination criterion at the end of each iteration. Thus, the overall communication complexity for the privacy-preserving k-means clustering protocol is $O(n\ell ck)$ per iteration. It has been shown that for typical data and suitably chosen k, the number of iterations required by the k-means algorithm is typically small.

For each iteration the total computational complexity for computing the cluster centers is $O(nk\ell)$ encryptions and at most $O(nk\ell)$ multiplications for Alice and $O(nk\ell)$ exponentiations and at most $O(nk\ell)$ multiplications for Bob.

PRIVACY. As a privacy ideal, we compare the privacy of our k-means clustering algorithm with one that makes use of a trusted third party (TTP) who receives all the data, locally runs the standard k-means algorithm, and returns the appropriate cluster numbers (and possibly cluster centers) to the parties. If one party completely owns an object, then only that party gets the cluster number assigned to that object. If both parties share an object then both parties get the assignment.

In contrast, the privacy-preserving k-means clustering protocol is an interactive protocol. Although secret sharing is used to protect certain values, Alice and Bob learn some intermediate results. At the end of each iteration, they learn the assignment of the cluster number to the objects. If one party holds the object completely, then only that party gets the assignment. Otherwise, both parties get the assignment.

This intermediate information can sometimes reveal information about the parties' data. For example, consider a database consisting of n objects, say $d_1, \ldots, d_n$, and suppose that each object has two attributes x and y. Alice has the values corresponding to the attribute x and Bob has values corresponding to the attribute y. Suppose $d_1[x] = d_2[x] = \cdots = d_n[x] = 0, d_n[y] = 4$, and $d_1[y]$ through $d_{n-1}[y]$ ranges from 0 to 3, and suppose d_1, d_3 and d_n are chosen as the initial choices for the centers ($k = 3$). If one of the clusters in the next iteration contains only d_n, then Alice can guess that $d_n[y]$ is close to 4. But the result of the final iteration may not have a cluster having $d_n[y]$ as a single point. The likelihood of this kind of leakage occurring can be reduced by choosing the initial centers at random.

We note that our protocol, when restricted to vertically partitioned data, is similar to the one given by Vaidya and Clifton [16]. The communication complexity and the privacy of our protocol is essentially the same as theirs.

5. CONCLUSION

This paper has two main contributions. First, we introduce the idea of arbitrarily partitioned data, which is a generalization of both horizontally and vertically partitioned data. Protocols in this model can be applied to both horizontally and vertically partitioned data, as well as to data anywhere in between. Second, we provide a privacy-preserving k-means clustering algorithm over arbitrarily partitioned data.

As previously discussed, our algorithm potentially leaks some information through the intermediate cluster assignments, even though the intermediate cluster centers themselves are not revealed. It is not clear except for some rare scenarios such as the one described in Section 4 that these values can yield anything useful, but obviously it would be desirable to find an efficient algorithm that did not leak the intermediate cluster assignments.

A more general direction for further work is to obtain privacy-preserving data mining protocols for other data mining algorithms over arbitrarily partitioned data.

6. REFERENCES

[1] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 439–450. ACM Press, May 2000.

[2] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proc. Eighth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–209. ACM Press, 1989.

[3] S. Benninga and B. Czaczkes. *Financial Modelling.* MIT Press, 1997.

[4] R. R. de Carvalho, S. G. Djorgovski, N. Weir, U. Fayyad, J. Roden, A. Gray, and K. Cherkauer. Applications of clustering analysis and unsupervised classification algorithms to digitized POSS-II. *Bulletin of the American Astronomical Society*, 26:1372, December 1994.

[5] E. Forgey. Cluster analysis of multivariate data: Efficiency vs. interpretability of classification. *Biometrics*, 21:768, 1965.

[6] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen. On secure scalar product computation for privacy-preserving data mining. In *The 7th Annual International Conf. in Information Security and Cryptology*, 2004.

[7] O. Goldreich. *Foundations of Cryptography, Vol II.* Cambridge University Press, 2004.

[8] D. Gusfield. *Algorithms on Strings Trees and Strings.* Cambridge University Press, 1997.

[9] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Lecture Notes in Computer Science*, 1880, 2000.

[10] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[11] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–296, 1967.

[12] R. Mattison. *Data Warehousing and Data Mining for Telecommunication.* Artech Press, 1997.

[13] T. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[14] S. Oliveira and O. R. Zaïane. Privacy preserving clustering by data transformation. In *Proc. 18th Brazilian Symposium on Databases*, pages 304–318, 2003.

[15] Frank De Smet, Janick Mathys, Kathleen Marchal, Gert Thijs, Bart De Moor, and Yves Moreau. Adaptive quality-based clustering of gene expression profiles. *Bioinformatics*, 18(5):735–746, 2002.

[16] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *Proc. 9th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining.* ACM Press, 2003.

[17] O. Veksler. Image segmentation by nested cuts. In *Proc. of IEEE Computer Vision and Pattern Recognition*, pages 339–344, 2000.

[18] A. C.-C. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symp. on Foundations of Computer Science*, pages 162–167, 1986.