Communication-Efficient Privacy-Preserving Clustering

Geetha Jagannathan*, Krishnan Pillaipakkamnatt, Rebecca N. Wright*, Daryl Umano****

*Department of Computer Science, Rutgers University, New Brunswick, NJ, USA.

**Department of Computer Science, Hofstra University, Hempstead, NY,USA.

E-mail: geetha@cs.rutgers.edu, csckzp@hofstra.edu,
Rebecca.Wright@rutgers.edu, dumano33@hotmail.com

Abstract. The ability to store vast quantities of data and the emergence of high speed networking have led to intense interest in distributed data mining. However, privacy concerns, as well as regulations, often prevent the sharing of data between multiple parties. Privacy-preserving distributed data mining allows the cooperative computation of data mining algorithms without requiring the participating organizations to reveal their individual data items to each other.

This paper makes several contributions. First, we present a simple, deterministic, I/O-efficient k-clustering algorithm that was designed with the goal of enabling an efficient privacy-preserving version of the algorithm. Our algorithm examines each item in the database only once and uses only sequential access to the data. Our experiments show that this algorithm produces cluster centers that are, on average, more accurate than the ones produced by the well known iterative k-means algorithm, and compares well against BIRCH.

Second, we present a distributed privacy-preserving protocol for k-clustering based on our new clustering algorithm. The protocol applies to databases that are horizontally partitioned between two parties. The participants of the protocol learn only the final cluster centers on completion of the protocol. Unlike most of the earlier results in privacy-preserving clustering, our protocol does not reveal intermediate candidate cluster centers. The protocol is also efficient in terms of communication and does not depend on the size of the database. Although there have been other clustering algorithms that improve on the k-means algorithm, ours is the first for which a communication efficient cryptographic privacy-preserving protocol has been demonstrated.

Keywords. Secure computation, distributed data mining, privacy preservation, clustering.

1 Introduction

The rapid growth of storage capacity, coupled with the development of tools for data reporting and analysis led to the development of data warehousing [26]. A data warehouse

*Preliminary versions of parts of this work were published in [28] and [27].

allows for the analysis of operational or transactional data, such as from retail transactions or web server logs, for organizations. Decision support systems allow for data to be summarized and viewed along various dimensions by managers of such organizations. With advances in artificial intelligence and statistics, it then became possible to "mine" these data sources for hidden patterns and knowledge. Data mining has been successfully applied in a wide range of domains extending from DNA data analysis [22] to financial data analysis [5] to fraud detection [37]. Of the many issues that data mining raises, we consider two in this paper:

- **Private distributed data mining:** Data may be distributed among multiple databases owned by different entities. Distributed data mining is the acquisition of knowledge from multiple databases, each of which may be owned by a different organization. While distributed data mining has the potential to yield more precise knowledge or knowledge that has wider applicability than the use of data owned by a single entity, legal requirements of privacy (for example, in the case of medical records) or corporate secrecy can prevent such collaborative data mining. The field of privacy-preserving data mining [3, 35] concerns itself with algorithms and protocols that allow data owners to collaborate in such endeavors without any of the collaborators being required to reveal any items from their databases. The two goals to achieve in this setting are privacy (no "leakage" of original data) and communication efficiency (minimizing the amount of data to be exchanged between the participants).

- **Large databases and data streams:** Algorithms that work on large databases and data streams must be efficient in terms of I/O access. Since secondary-storage access can be significantly more expensive than main-memory access, an algorithm should make a small number of passes over the data, ideally no more than one pass. Also, since main memory capacity typically lags significantly behind secondary storage, these algorithms should have sublinear space complexity. Although the notion of data streams has been formalized relatively recently, a surprisingly large body of results already exists for stream data mining [10, 1, 2, 8, 13, 18].

Most of the privacy-preserving protocols available in the literature convert existing data mining algorithms or distributed data mining algorithms into privacy-preserving protocols. The resulting protocols can sometimes leak additional information. For example, in the privacy-preserving clustering protocols of [43, 31], the two collaborating parties learn the candidate cluster centers at the end of each iteration. As explained in [29], this intermediate information can sometimes reveal some of the parties' original data, thus breaching privacy. While it is possible to modify these protocols to avoid revealing the candidate cluster centers [7], the resulting protocols have very high communication complexity. In this paper, we describe a k-clustering algorithm that we specifically designed to be converted into a communication-efficient protocol for private clustering. The resulting protocol does not leak any information about the original data. Our algorithm is also I/O-efficient in that each data item is examined only once, and it only uses sequential access to the data. We also present a modified form of our k-clustering algorithm that works for data streams. We first overview related work and then describe our contribution in more detail.

1.1 Related Work

PRIVACY-PRESERVING DATA MINING Privacy-preserving data mining was introduced by Agarwal and Srikant [3] and Lindell and Pinkas [35]. Both papers were for the se-

cure construction of decision trees, albeit using completely different approaches. The former uses randomness to "perturb" the original data. Reconstruction algorithms are then used to extract summary information for creating decision trees. The latter paper uses techniques from cryptography and secure multiparty computation (SMC) [17], as do the results presented here.

Theoretically, Yao's general purpose secure circuit-evaluation protocol [47] solves any distributed two-party privacy-preserving data mining problem. As a practical matter, however, the circuits for even megabyte-sized databases would be intractably large. The alternative has been to find secure special-purpose protocols for specific data mining problems. These protocols typically use Yao's protocol for evaluating very small circuits in intermediate stages, but use other, more efficient, methods when examining the actual data.

Most results in privacy-preserving data mining assume that the data is either horizontally partitioned (that is, each party to the protocol has some subset of the rows of an imaginary "global database"), or vertically partitioned (that is, each party has some subset of the columns of the "global database") (e.g.,[12, 44, 32, 46]).

Privacy-preserving clustering has been previously addressed by Oliviera and Zaïane [41], Vaidya and Clifton [43], Jagannathan and Wright [29], Jha, Kruger, and McDaniel [31] and Bunn and Ostrovsky [7]. Oliviera and Zaïane's work [41] uses data transformation in conjunction with partition-based and hierarchical clustering algorithms, while the others use cryptographic techniques to give privacy-preserving versions of the k-means clustering algorithm. Vaidya and Clifton's result [43] addresses privacy-preserving k-means clustering for vertically partitioned data, Jha, Kruger, and McDaniel's [31] addresses horizontally partitioned data, and Jagannathan and Wright [29] and Bunn and Ostrovsky [7] address arbitrarily-partitioned data.

This paper presents a protocol for privacy-preserving clustering, based on a new algorithm for k-clustering. The new privacy-preserving protocol leaks no information, has better error in some cases, and is more input/output-efficient.

CLUSTERING Clustering is a well-studied combinatorial problem [24, 30, 23], and there have been a large number of algorithms developed to solve the various formulations of the problem. The task is to group similar items in a given data set into *clusters*. The goal is to obtain a clustering that minimizes an *objective function* mapping a given clustering into a numerical value.

The error-sum-of-squares (ESS) objective function is defined as the sum of the squares of the distances between points in the database to their nearest cluster centers. The k-clustering problem requires the partitioning of the data into k clusters with the objective of minimizing the ESS. Lloyd's algorithm [36] (more popularly known as the k-means algorithm) is a popular clustering tool for the problem. This algorithm converges to a local minimum for the ESS in a finite number of iterations. Ward's algorithm [45] for hierarchical agglomerative clustering also makes use of the notion of ESS. Although Ward's algorithm has been observed to work well in practice, it is rather slow ($O(n^3)$) and does not scale well to large databases.

STREAM DATA MINING ALGORITHMS Recently there have been a number of data mining algorithms designed for scalability. A key constraint on these algorithms is that the input is too large to fit in its entirety in main memory. Further, since secondary storage access is very expensive in terms of time, it is preferable that the number of scans of the database be limited. These constraints were formalized as the notion of a data stream by Henzinger,

Raghavan and Rajagopalan [25], although prior results such as [4, 38] had been published in this model. The well known clustering algorithm BIRCH [48], although not originally defined as a data stream algorithm, fits this definition of data stream clustering.

Among the earliest results for data mining in the data stream model are the works by Guha et al. [19, 18] for clustering. Another early work is by Domingoes and Hulten [10] for decision trees. Charikar, O'Callaghan and Panigrahy have presented a constant factor approximation algorithm for the k-medians problem for data streams [8]. Aggarwal et al. have given algorithms for clustering data streams in which the clusters drift over time [1] and for clustering in high dimensional data [2]. Drifting clusters have also been studied by Fan [13].

1.2 Our Contributions

In this paper, we present a simple deterministic algorithm called ReCluster for I/O-efficient k-clustering. This algorithm examines each data item only once and uses only sequential access to the data. For fixed k, the time complexity of the algorithm is linear in the number n of items in the database and the space complexity is $O(\log n)$. We present results from the application of the algorithm to synthetic data and realistic data, and compare our results with the well-known iterative k-means clustering algorithm and BIRCH. Our results show that ReCluster, on average, is more accurate in identifying cluster centers than the k-means clustering algorithm and compares well against BIRCH. We note that although there have been other clustering algorithms that improve on the k-means clustering algorithm, this is the first for which an efficient cryptographic privacy-preserving version has been demonstrated. ReCluster works only on numeric data. We also present a modified form of the ReCluster algorithm called StreamCluster which is intended to work on data stream inputs.

We also demonstrate that the ReCluster algorithm lends itself well to privacy-preserving computation. Specifically, we present a privacy-preserving version of the ReCluster algorithm, for two-party horizontally-partitioned databases. This protocol is communication-efficient and it reveals only the final cluster centers (or the cluster assignments to data) to both parties at the end of the protocol. This protocol does not reveal the intermediate candidate cluster centers or intermediate cluster assignments.

Although an interesting clustering algorithm in its own right, ReCluster was explicitly designed to be converted into a privacy-preserving protocol. This was motivated by the fact that most of the existing efficient protocols for private k-clustering are based on Lloyd's k-means algorithm, with the same resulting clusters. All of them except [7] leak intermediate information that could potentially result in a breach of privacy. These protocols can be made more secure but only at the cost of higher communication complexity. In [7], Bunn and Ostrovsky describe a 2-party k-means clustering protocol that guarantees full privacy. However, the communication complexity of this protocol is linear in the size of the database, which is high when the size of the database is large.

An alternate way of solving this problem would be to develop privacy-preserving version of other k-clustering algorithms. Many of them, however, have their own disadvantages. Other well known k-clustering algorithms include CLARANS [40], BIRCH [48], and STREAMLS [18]. CLARANS is an in-memory algorithm, and hence does not scale well to large databases. Prasad and Rangan [34] presented a privacy-preserving protocol for BIRCH on arbitrarily-partitioned databases. This algorithm, when reduced to the case of horizontally-partitioned databases, results in a communication complexity of O(n), where n is the size of the database. This can be rather high for large databases. Also, the amount

of memory used for BIRCH is much higher than ReCluster (see Table 3). The STREAMLS algorithm depends on repeatedly solving a facility-location problem using a local search algorithm, and then finally applying a linear programming based clustering algorithm. Each of these subroutines (local search and linear programming) can be complicated to perform privately. In comparison, our privacy-preserving version of ReCluster is simple and communication-efficient, and produces good clusters.

We start in Section 2 by introducing some cryptographic primitives we use in our privacy preserving protocols. We present our k-clustering algorithm in Section 3. In Section 4, we present experimental results comparing our algorithm with the k-means clustering algorithm and BIRCH. We present the privacy-preserving clustering protocol with performance analysis and comparison to other private clustering algorithms in Section 5.

2 Cryptographic Primitives

We now introduce some cryptographic primitives used in later sections.

Random Shares According to our privacy requirements, both parties should receive the cluster centers at the end of the protocol, but all of the values computed in the intermediate steps of the algorithm should be unknown to either party. In our protocol, each computed intermediate value in the communication phase, such as a candidate cluster center, is shared as two uniformly distributed random values, with each party holding one of these two values. Their sum is the actual intermediate value.

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.

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.

Homomorphic Encryption Homomorphic encryption schemes [14] allow certain computations on encrypted values. In particular, an encryption scheme is *additively homomorphic* if there is some operation $\otimes$ on encryptions such that for all cleartext values a and b, $E(a) \otimes E(b) = E(a + b)$. Our solutions make use of a semantically secure additively homomorphic encryption scheme. Examples, under suitable cryptographic hardness assumptions, include [42, 9]. Other homomorphic encryption schemes include Boneh-Goh-Nissim cryptosystem [6] that supports arbitrary addition and one multiplication and the fully homomorphic encryption scheme [15] that supports evaluation of arbitrary circuits over encrypted data without having to decrypt.

Secure Scalar Product Protocol One of the steps in the private clustering protocol requires the secure computation of the scalar product of vector $X = (x_1, \ldots, x_n)$, held by Alice, and $Y = (y_1, \ldots, y_n)$, held by Bob. They need to securely compute the scalar product as $s_A + s_B = X \cdot Y$, where s_A and s_B are the random shares of Alice and Bob. Recall that all computations in the paper are carried out modulo N. Our solution makes use of a secure scalar product protocol that computes such shares, such as [16].

Algorithm 1 ReCluster

Input: Database D, Integer k
Output: Cluster centers S

1. $S' = \mathsf{RecursiveCluster}(D, 2k)$ // Produce $2k$ clusters

2. $S = \mathsf{MergeCenters}(S', k)$ // Compress to k clusters

3. Output S

Algorithm 2 RecursiveCluster

Input: Database D, Integer k
Output: Cluster centers S

If ($|D| \leq k$) then $S = D$
Else

1. D_1 = First half of D

2. D_2 = Second half of D

3. $S_1 = \mathsf{RecursiveCluster}(D_1, k)$

4. $S_2 = \mathsf{RecursiveCluster}(D_2, k)$

5. $S = \mathsf{MergeCenters}(S_1 \cup S_2, k)$

End If
Output S

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 protocol [47] to securely compute the index j such that $x_j + y_j$ is minimum.

3 The k-Clustering Algorithm

In this section we present our new algorithm, ReCluster, for k-clustering. We also present an extension of the algorithm to data streams and a distributed (non-private) version of ReCluster. Experimental results are given in Section 4.

3.1 The Basic ReCluster Algorithm

ReCluster makes the assumption that the size of the data set is known before running the algorithm. Our algorithm runs in the typical "divide, conquer and combine" fashion used in algorithms such as MergeSort. Strictly speaking, this strategy would require us to divide the database into two equal halves, recursively produce k cluster centers from each of the halves, and then "merge" these $2k$ centers into the k final centers. However, we take a slightly different approach, similar to that used by Guha et al. in [19]. Instead of generating k centers from each recursive call, we produce $2k$ cluster centers from each

recursive call, and then merge the total of $4k$ centers thus received into $2k$ centers[1]. (That is, the RecursiveCluster algorithm is initially invoked with the second input parameter being set to $2k$, where k is the desired number of clusters.) A final post-processing step uses the same merge technique to produce the k final centers from the $2k$ clusters returned from the top-most level of the recursion tree. The top level of ReCluster is presented in Algorithms 1 and 2. The subroutine for merging clusters is described in Algorithm 3. This latter algorithm depends on a notion of merge error which is described in Section 3.1.1. For a data set of n elements, the overall time complexity of the algorithm is $O(nk^2)$.

3.1.1 Error Measure.

The key step in ReCluster is the merging of $4k$ centers into $2k$ centers after the two recursive calls. For this step, we use a variation of the notion of error defined in Ward's algorithm for bottom-up hierarchical clustering [45]. Let C_1 and C_2 be two clusters being considered for a merge at some stage. Associated with each cluster center C is a *weight* (denoted $C.\mathsf{weight}$), which is the number of objects that are assigned to that cluster. In Ward's algorithm the error of $C_1 \cup C_2$ is defined as

$$\mathsf{error}_w(C_1 \cup C_2) = \frac{C_1.\mathsf{weight} * C_2.\mathsf{weight} * \mathsf{dist}^2(C_1, C_2)}{C_1.\mathsf{weight} + C_2.\mathsf{weight}},$$

where $\mathsf{dist}(C_1, C_2)$ is defined as the Euclidean distance between the center of C_1 and the center of C_2. ReCluster defines the error of the union as

$$\mathsf{error}_r(C_1 \cup C_2) = C_1.\mathsf{weight} * C_2.\mathsf{weight} * \mathsf{dist}^2(C_1, C_2).$$

In most situations, the pair of clusters that is chosen for merging is the same whether we define error of the union as error_w or as error_r. However, there are conditions where the choices made using the two error measures are different. Consider two cluster pairs (C_p, C_q) and (C_s, C_t) such that $\mathsf{dist}(C_p, C_q) \approx \mathsf{dist}(C_s, C_t)$ and $C_p.\mathsf{weight} * C_q.\mathsf{weight} \approx C_s.\mathsf{weight} * C_t.\mathsf{weight}$, but $C_p.\mathsf{weight} \gg C_q.\mathsf{weight}$ and $C_s.\mathsf{weight} \approx C_t.\mathsf{weight}$. That is, the distance between the cluster centers of the first pair is the same as the distance between the cluster centers of the second pair, but cluster C_p has many more objects in it compared to C_q, while clusters C_s and C_t have about the same number of objects.

If we use error_w to decide which of the two pairs gets merged, we would merge the pair (C_p, C_q), while error_r would favor the merger of (C_s, C_t). See Figure 1. Our experiments indicate that using error_r in ReCluster makes the algorithm less susceptible to noise, because noise objects (which do not clearly belong to any cluster) resemble low-weight clusters.

Given this definition of error, the compression of a set S of $4k$ clusters into $2k$ clusters is obtained by repeatedly:

1. choosing a pair of clusters C_i and C_j such that the error of their union is minimum among all such pairs,
2. deleting from S the clusters C_i and C_j, and
3. inserting into S the new cluster $C_i \cup C_j$.

The weight of this new cluster is $C_i.\mathsf{weight} + C_j.\mathsf{weight}$. See Algorithm 3.

[1] When the number of items in the database cannot be expressed in the form $k2^x$ for some integer x, the base case of the recursive function may end up producing clusters with fewer than k centers. This has been experimentally seen to sometimes produce poor clusterings.

Algorithm 3 MergeCenters

Input: Cluster centers S, Integer k
Output: Cluster centers S, such that $|S| = k$

1. While ($|S| > k$)
   (a) Compute the merge error for all pairs of centers in S.
   (b) Remove from S the pair with the lowest merge error, and insert the center of the merged cluster, with its weight as the sum of the weights of the pair.

   End While
2. Output S

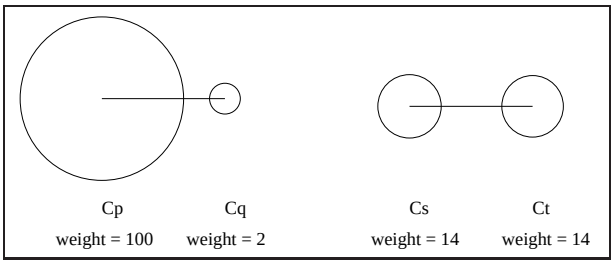


Figure 1: The effect of choosing error_r over error_w. ReCluster prefers to merge C_s and C_t, Ward's algorithm would prefer to merge C_p and C_q.

3.2 Extension to Data Streams

In this section we modify the ReCluster algorithm to work on data streams. The main idea is to eliminate recursion since the number of data items to be clustered is not known ahead of time.

The algorithm (see Algorithm 4) can be said to be "incrementally agglomerative." That is, the algorithm merges intermediate clusters without waiting for all the data to be available. At the beginning of each iteration, the algorithm holds a number of intermediate k-clusterings. We associate with each such k-clustering a *level*, a concept similar to the one in [18]. When two k-clusterings at level-i are "merged" to form a new k-clustering, this new clustering is said to be at level $(i + 1)$. We also associate with each cluster a *weight*, which is the number of data points in that cluster. Each cluster is represented by its center, which is the mean of all points in it.

In each iteration, the algorithm adds the next k data points as a level-0 intermediate k-clustering. If the algorithm detects two k-clusterings at level i in the list, these are "merged" (using the MergeCenters routine, see Algorithm 3) into a k-clustering at level $(i + 1)$.

When a k-clustering needs to be output after some points have been read from the data stream, all intermediate k-clusterings at all levels are "merged" using MergeCenters into a single k-clustering.

For a data stream of n elements, the overall time complexity of the algorithm is $O(nk^2)$. The amount of memory required to store the $O(\log \frac{n}{k})$ intermediate k-clusterings is $O(k \log \frac{n}{k})$.

Algorithm 4 StreamCluster

Input: Database D, Integer k
Output: Cluster centers S
Repeat as long as data remains:

1. Read the next k items from D as a level-0 k-clustering.
2. Add the level 0 clustering to the vector V of intermediate clusterings.
3. While $V[last].level = V[last-1].level$
   (a) $V[last-1] = \mathsf{MergeCenters}(V[last-1] \cup V[last], k)$
   (b) $V[last-1].level = V[last-1].level + 1$
   (c) Remove $V[last]$ from V.
4. If an output is needed, union all intermediate clusterings in V and return output from MergeCenters.

3.3 Distributed k-Clustering Protocol for Two Parties

Following convention, we designate the two parties who wish to jointly compute the cluster centers using their data as Alice and Bob. This distributed protocol forms the basis for the privacy-preserving clustering protocol to be presented in Section 5. We will assume that Alice holds the data set D_1 and Bob holds D_2. The protocol runs as follows:

1. Alice uses the ReCluster algorithm on the data set D_1 to compute $2k$ intermediate cluster centers.
2. Bob computes $2k$ cluster centers using his data set D_2.
3. Alice sends her intermediate cluster centers to Bob.
4. Bob takes the union of all the intermediate clusters and uses the MergeCenters routine to obtain the k final cluster centers.

4 Experimental Results

In this section we present our experimental results that show the ability of ReCluster and its extensions to accurately identify clusters, as measured by the error sum of squares. We ran our experiments on a large number of synthetic and realistic data sets. All algorithms were implemented in C++ and experiments were run on a Windows-based personal computer with an AMD 3500+ CPU and 3 GB of main memory. We first describe in Sections 4.1 and 4.2 the datasets that we use. Results are given in Section 4.3 for the basic algorithm and in Section 4.4 for the distributed algorithm.

4.1 Synthetic Data Generator

To test ReCluster's ability to accurately identify clusters, we generated three types of synthetic data sets, namely Random Centers, Grid data sets, and Offset Grid data sets. We describe these data sets below. These data sets are similar to the ones used in [48, 20, 21, 39].

Parameter	Value
Number of clusters, k	5
Min. number of points per cluster, n_{l}	900
Max. number of points per cluster, n_{u}	1100
Min. major radius of cluster, r_{al}	30
Max. major radius of cluster, r_{au}	50
Min. minor radius of cluster, r_{bl}	30
Max. minor radius of cluster, r_{bu}	50
Max. X-axis value, s_{w}	500
Max. Y-axis value, s_{h}	500
Add noise	y

Table 1: Sample parameters for our synthetic data set generator while generating a Random Centers data set

Each data set consists of points distributed in 2D space. Our synthetic data set generator takes a number of parameters:

- The number k of clusters.
- The range $[n_{\mathrm{l}}, n_{\mathrm{u}}]$ of the desired number of points in each cluster.
- The parameters s_{w} and s_{h} which denote the width and the height, respectively, of the space in which data points are generated.

In one collection of data sets that we call Random Centers the clusters are chosen as ellipses with major radius r_{a} chosen in the defined range $[r_{\mathrm{al}}, r_{\mathrm{au}}]$ and minor radius r_{b} chosen in the defined range $[r_{\mathrm{bl}}, r_{\mathrm{bu}}]$. (When $r_{\mathrm{a}} = r_{\mathrm{b}}$ we get circular clusters.) The cluster centers are positioned randomly in the space within which points are generated. In grid data sets and in offset grid data sets, circular clusters are chosen with radius r, with all clusters having the same radius. The cluster centers are placed on a $\sqrt{k} \times \sqrt{k}$ grid. In the case of the offset grid, the cluster centers are then randomly perturbed with small values in the x and y directions. After the center and radius of each cluster is finalized, the synthetic data generator then generates $n_{\mathrm{c}} \in [n_{\mathrm{l}}, n_{\mathrm{u}}]$ points at random within the cluster. In all cases, in addition to the points generated by the procedure above, we add 5% noise in the form of data points uniformly distributed throughout the data set in order to explore how the algorithm performs with noisy data. Table 1 illustrates a set of sample parameters for these data sets. For each data set we ran the k-means algorithm 10 times, each run with a different randomly chosen set of initial cluster centers. We compared the performance of ReCluster against the average performance of the k-means algorithm.

Each synthetic data set used either uniform distributions over some intervals or normal distributions for generating clusters. Our figures do not include any of the normally distributed data since the results were essentially the same as for uniform distributions. Our experiments show that data ordering has a low effect on the output of our algorithm. This is possibly because the MergeCenters routine acts as a global optimizer.

4.2 Realistic Data

We compared the error sum of squares (ESS) of ReCluster against BIRCH for the network intrusion data set used in the 1999 KDD Cup [33]. This data set contains information about

True centers $\times 10^7$	Recluster centers $\times 10^7$	Average for k-means centers $\times 10^7$
5.52	5.71	7.25
8.14	8.87	11.99
8.77	9.31	14.1
5.68	5.71	7.42
6.13	6.34	7.68
6.59	6.85	8.88
7.48	7.52	13.35
10.04	10.69	14.84
5.65	5.64	11.91
6.55	7.02	8.60

Table 2: The error sum of squares for a subset of the Random Centers data sets. In each case the number of clusters was 5, with approximately 10,000 points in each cluster. Noise was added at the rate of 5%.

five million network connections. The vast majority of these data points are about benign network connections. A small fraction of these connections represent four different kinds of network intrusion attacks. We selected the 10% subset to run our experiments. Each connection record is described in terms of 41 attributes, of which 34 are continuous. We projected out the 7 symbolic attributes for our experiments. Also, we eliminated one outlier record. We set k to 5 in our runs of ReCluster and BIRCH.

4.3 Results for ReCluster

We present below the results of our experiments on synthetic data sets and on the network intrusion data set. In order to show the ability of ReCluster to accurately identify clusters we compute the error sum of squares for the centers detected by ReCluster. For some of the data sets we also visually present the clusters and the detected centers as a demonstration of the clustering algorithm.

In Figure 2, we present the results from a typical experiment on a Random Centers data set, which is a synthetically generated data set with randomly positioned cluster centers. We created 50 such data sets. In 25 of the 50 cases we distributed data points using a uniform distribution within each cluster, and the remaining data sets used Gaussian distributions. Figure 2 shows a data set in which the points were distributed using the uniform distribution. ReCluster does very well in identifying cluster centers, even in the presence of noise. However, in four out of these 25 data sets ReCluster placed the center of a cluster outside the cluster, albeit close to the cluster itself. See Figure 3 for such a data set. Also shown in Figure 3 are the centers identified in the best run of k-means algorithm. On this particular data set the k-means algorithm failed to identify all cluster centers on four of the 10 runs. In Table 2 we present the error sum of squares for ReCluster and k-means on 10 of these Random Centers data sets.

The average running time for ReCluster over all the uniform distribution data sets, which had about 51,000 points on average, was 1.06 seconds, and the average running time for the k-means algorithm was 8.09 seconds.

In Figure 4, we present the results from experiments on a Grid data set, which is a synthetically generated data set with cluster centers positioned on a grid. We ran our experiments

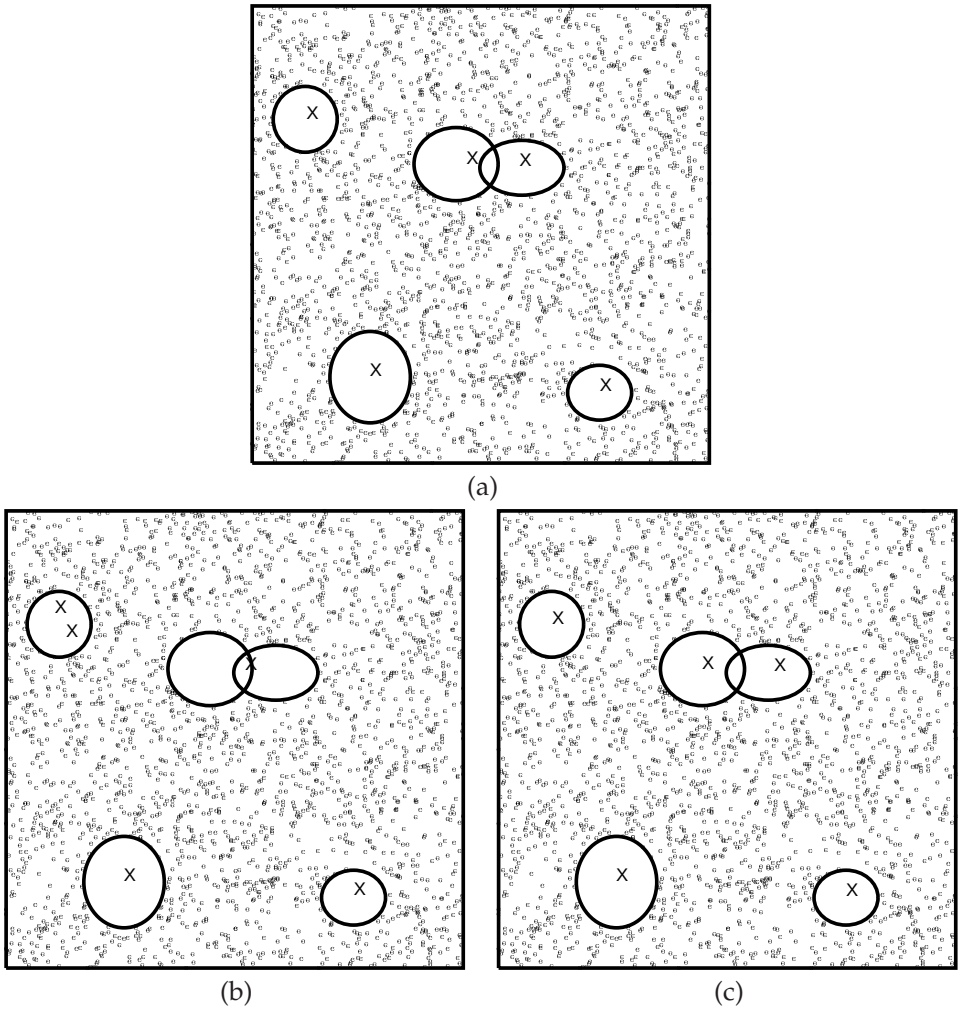


Figure 2: The data set consists of about 10,000 points concentrated around each cluster center plus 5% "noise" points occurring elsewhere. The points concentrated at each cluster center are illustrated by a large open ellipse around each center. The noise points are shown as points. Cluster centers as identified by (a) ReCluster (ESS = 5.71×10^7) , (b) the worst run of k-means algorithm (ESS = 8.49×10^7) and (c) the best run of k-means algorithm (ESS = 5.48×10^7).

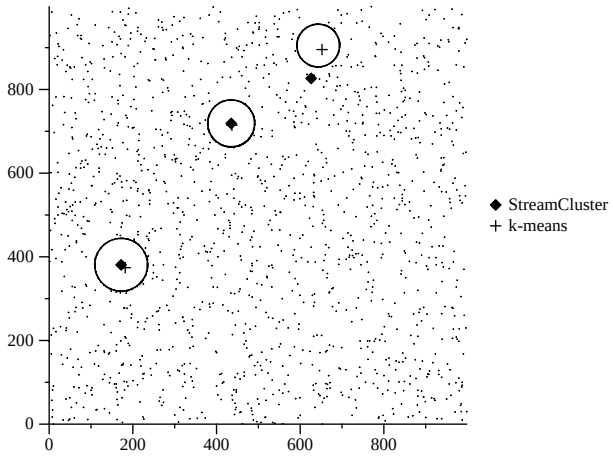


Figure 3: An example data set in which ReCluster did not accurately identify all cluster centers. The data set consists of about 10,000 points concentrated around each cluster center plus 5% "noise" points occurring elsewhere. The points concentrated at each cluster center are illustrated by a large open ellipse around each center. The noise points are shown as points. Also shown are the centers identified by the best run of the k-means algorithm.

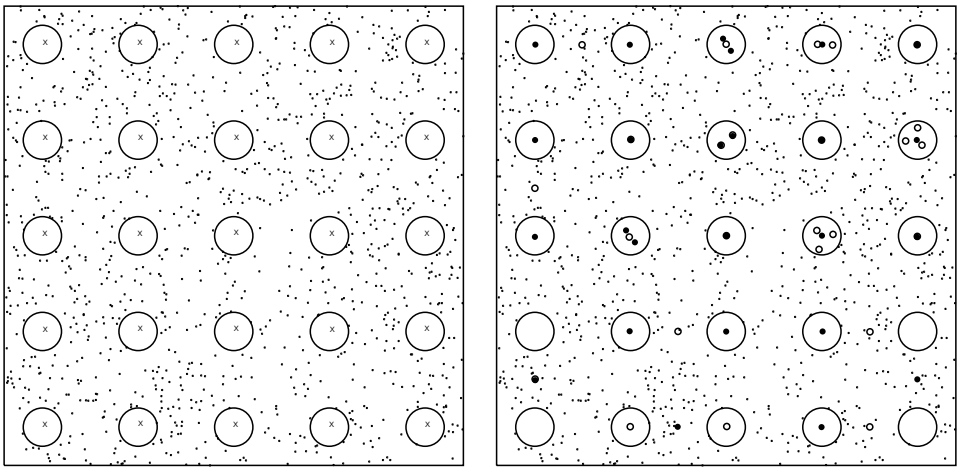
on two such data sets and Figure 4 represents one of them. As can be seen, ReCluster accurately identifies the centers of all clusters. On the other hand, for this data set not even the best run of the k-means algorithm identified all 25 cluster centers. The worst run of k-means did not identify even half of the cluster centers. In Figure 5 we present the results on an Offset Grid data set. ReCluster was quite successful in identifying all cluster centers, which k-means did not even on its best run.

In Table 3 we present the result of our experiment on the Network Intrusion data. As our results indicate, ReCluster does extremely well relative to BIRCH in this data set. BIRCH needs more than 32 times as much memory as used by ReCluster, to find five clusters. Even when given 128 times as much memory as ReCluster, BIRCH does not perform nearly as well as. Perhaps with much more memory BIRCH could perform better.

4.4 Results for Distributed ReCluster

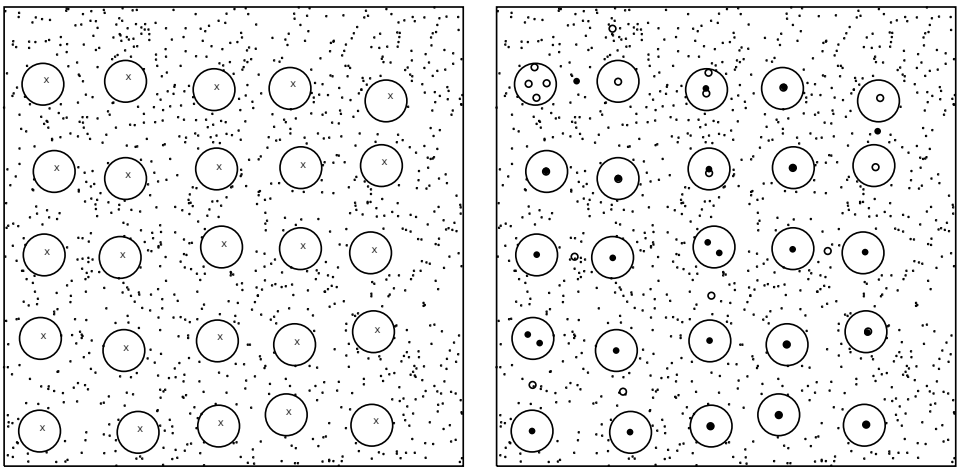
The synthetic data set generator can also generate data sets for the two party distributed k-clustering problem. In addition to the parameters mentioned in Section 4.1, the user can input how the clusters are to be apportioned between the two parties, and how much data about one party's clusters are also known to the other party.

Although similar in its essentials to ReCluster, the distributed clustering protocol in Section 3.3 is not identical to ReCluster. The ReCluster algorithm splits the data set into two equal halves, then recursively clusters each half, following which it merges the clusters from both halves. The distributed algorithm makes no assumptions about the relative sizes of the data sets held by the two parties and one party could have far more data than the other. Given this distinction, it is not immediately obvious that the distributed clustering algorithm would have properties similar to that of ReCluster. To test the effectiveness of this distributed algorithm, we ran experiments where the parties had different proportions



Recluster ESS = 3.27×10^7 Average k-means ESS = 1.69×10^8

Figure 4: Cluster centers identified in a grid data set. The data set consists of about 1000 points concentrated around each grid point plus 5% "noise" points occurring elsewhere. The points concentrated at each grid point are illustrated by a large open circle around each grid point. The noise points are shown as points. On the left are cluster centers identified by ReCluster. On the right are the best (identified by small filled circles) and worst (identified by small open circles) runs of the k-means algorithm.



Recluster ESS = 3.37×10^7 Average k-means ESS = 1.01×10^8

Figure 5: Cluster centers identified in an offset grid data set. The data set consists of about 1000 points concentrated around each grid point plus 5% "noise" points occurring elsewhere. The points concentrated at each grid point are illustrated by a large open circle around each grid point. The noise points are shown as points. On the left are cluster centers identified by ReCluster. On the right are the best (identified by small filled circles) and worst (identified by small open circles) runs of the k-means algorithm.

Algorithm	**Mem. Allowed** ($\times$ 24000 bytes)	**ESS**
ReCluster	1	4.11259E14
BIRCH	1	*
BIRCH	2	*
BIRCH	4	*
BIRCH	32	*
BIRCH	64	4.83018E17
BIRCH	128	4.8299E17

Table 3: Comparison of the error sum of squares for the network intrusion 10% data set. An asterisk indicates that BIRCH failed to find five clusters due to insufficient memory.

of the total data. In one set of experiments, the first party had twice as many data points per cluster as the second. In the second set, the first party had five times as many data points as the second. Within each set, we also varied the "separation" between the two parties so that in some cases, the two parties had knowledge of no clusters in common. That is, some clusters were entirely "owned" by the first party and the others were entirely "owned" by the second. In other cases, each party had some fraction of the data points that belong to clusters that were primarily owned by the other party. In all cases, we compared the results to the k-means clustering algorithm on the combined data. Some typical runs are presented in Figures 6 and 7. As can be seen, the distributed clustering protocol has performance close to that of k-means.

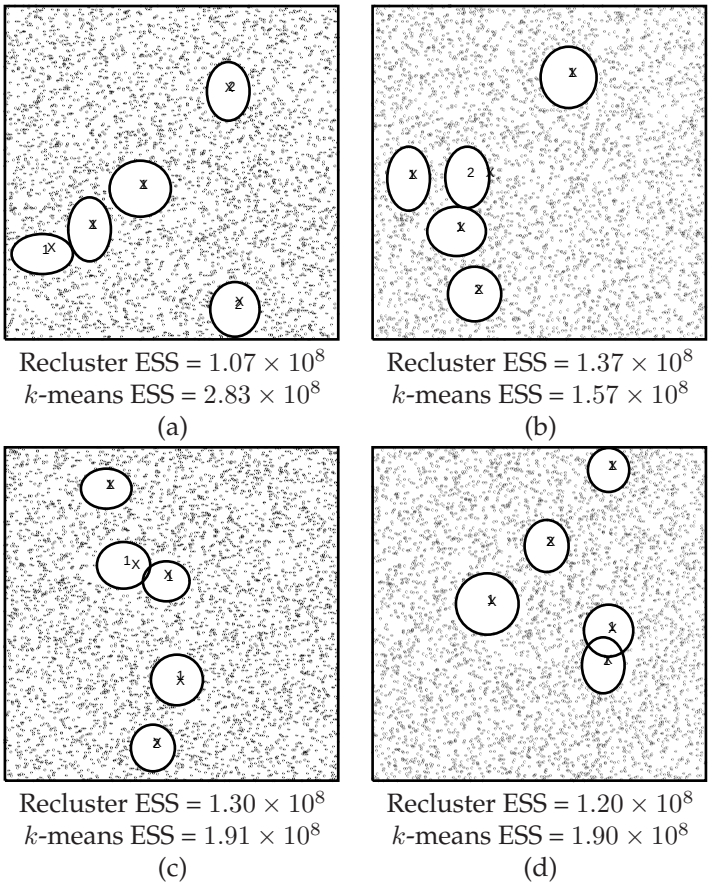
5 Privacy-Preserving k-Clustering Protocol

We now describe our privacy-preserving protocol for k-clustering based on the distributed ReCluster algorithm presented in Section 3. We begin with a description of the privacy model under which our protocol operates.

5.1 Our Model

Following convention, we designate the two parties who wish to collaborate as Alice and Bob who own databases $D_1 = \{d_1, \ldots, d_m\}$ and $D_2 = \{d_{m+1}, \ldots, d_n\}$, respectively. They wish to jointly compute a k-clustering of $D_1 \cup D_2$. At the end of the protocol both parties learn the final k cluster centers. The parties learn nothing else.

Suppose there were a trusted third party to whom Alice and Bob could send their data. The trusted third party would then compute the clustering and send the cluster centers to Alice and Bob. However, in many settings, there is no such trusted party. *Secure multiparty computation* seeks protocols that can carry out the required computation without requiring a trusted third party and without any party learning anything about the data other than their output (and what is implied by it). In this paper, we assume that Alice and Bob are *semi-honest*, meaning that they follow their protocol as specified, but may try to use the information they have learned (such as messages they receive in order to learn more than they are supposed to).



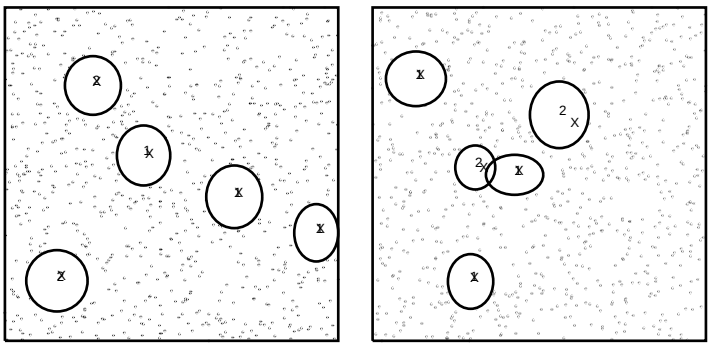
Recluster ESS = 1.07×10^8
k-means ESS = 2.83×10^8
(a)

Recluster ESS = 1.37×10^8
k-means ESS = 1.57×10^8
(b)

Recluster ESS = 1.30×10^8
k-means ESS = 1.91×10^8
(c)

Recluster ESS = 1.20×10^8
k-means ESS = 1.90×10^8
(d)

Figure 6: Some typical runs of the distributed clustering algorithm. The data sets consists of about 20,000 points concentrated around each cluster center owned by the first party, 10,000 points concentrated around each cluster center owned by the second party, plus 5% "noise" points occurring elsewhere. The points concentrated at each cluster center are illustrated by a large open ellipse around each center. The noise points are shown as points. Figures (a) and (b) are example data sets in which the first party "owns" the data for three of the clusters and the second party has data for the two remaining clusters. Figures (c) and (d) are example data sets in which the first party has data for four of the clusters and the second party has data for only one cluster. In Figures (a) and (c) each party has 25% of the data from the other party's clusters. In Figures (c) and (d) each party has no information regarding the clusters of the other party. In all cases the letter X marks the centers detected by the distributed clustering protocol. Also shown are the error sum of squares value for distributed ReCluster and the average error sum of squares for k-means.



Recluster ESS = 2.30×10^7
k-means ESS = 3.97×10^7

Recluster ESS = 2.54×10^7
k-means ESS = 2.56×10^7

Figure 7: Typical runs of the distributed clustering algorithm on more "skewed" data. Clusters owned by the first party have five times as many points as clusters owned by the second party (5000 points on average versus 1000 points on average). Each party has no data about the clusters owned by the other party. In all cases the letter X marks the centers detected by the distributed ReCluster protocol. Also shown are the error sum of squares value for distributed ReCluster and the average error sum of squares for k-means.

5.2 Overview of the Private Clustering Protocol

We now describe the private protocol. Alice and Bob separately use the ReCluster algorithm to compute $2k$ clusters each from their own shares of the data. Next, they randomly share their cluster centers with each other using the permute_share protocol described in detail in Section 5.3. At this point, Alice and Bob have random shares of $4k$ cluster centers. They again use the permute_share protocol to prevent each party keeping track of the cluster centers across iterations.

The two parties use the secure merge_clusters protocol, described in detail in Section 5.4, to iteratively merge the $4k$ clusters into k clusters. If the input to the merge_clusters protocol is j cluster centers, then at the end of the protocol, Alice and Bob obtain as output a random sharing of $j-1$ cluster centers. At the end of $3k$ joint executions of the merge_clusters protocol, they obtain a random sharing of k cluster centers. The parties may send each other their cluster center shares so that they obtain the actual cluster centers of the joint data.[2] This privacy-preserving ReCluster protocol does not leak any intermediate information.

Algorithm 5 illustrates the overall privacy-preserving ReCluster protocol. In the rest of this section, we describe in detail the subprotocols used. In Section 5.3, we describe the protocol that securely permutes and shares data between two parties. In Section 5.4, we present a secure protocol that merges clusters.

5.3 Secure Protocol to Share Data

When Alice has a vector of clusters C of the form $((c_1, w_1), \ldots, (c_k, w_k))$, this protocol helps Bob to obtain a permuted random share of the vector C. (Here each c_i is a cluster center,

[2] If they wish to learn the assignment of the cluster numbers to the objects they own, they can securely compute the cluster numbers using the shared cluster centers without revealing the cluster centers.

Algorithm 5 Privacy-preserving k-clustering protocol

Protocol Private_K_Clustering

Input: Database D consisting of n objects, where Alice owns $D_1 = (d_1, \ldots, d_m)$ and Bob owns $D_2 = (d_{m+1}, \ldots, d_n)$, and an integer k denoting the number of clusters.
Output: Assignment of cluster numbers to objects

1. Alice computes using the ReCluster algorithm $2k$ cluster centers from her data objects $\{d_1, \ldots, d_m\}$ and Bob computes using the ReCluster algorithm $2k$ cluster centers from his data objects $\{d_{m+1}, \ldots, d_n\}$.

2. Alice and Bob jointly share the cluster centers computed in Step 1 with each other.

   (a) Let $(c_1, w_1), \ldots, (c_{2k}, w_{2k})$ denote Alice's cluster centers. Here c_i denote the ith center and w_i denotes its corresponding weight. Similarly let $(c_{2k+1}, w_{2k+1}), \ldots, (c_{4k}, w_{4k})$ denote Bob's cluster centers.

   (b) Bob chooses a random permutation ϕ_1 and random values $r_1, \ldots, r_{2k}$, $s_1, \ldots, s_{2k}$. They engage in a secure permute_share protocol and obtain random shares of Alice's $2k$ cluster centers.

   (c) Alice chooses a random permutation ϕ_2 and random values $p_1, \ldots, p_{2k}$, $q_1, \ldots, q_{2k}$. They engage in a secure permute_share protocol and obtain random shares of Bob's $2k$ cluster centers.

   (d) Now Alice has $(c_1^A, w_1^A), \ldots, (c_{4k}^A, w_{4k}^A)$ and Bob has $(c_1^B, w_1^B), \ldots, (c_{4k}^B, w_{4k}^B)$. Bob chooses a random permutation ϕ_3 and random values $\alpha_1, \ldots, \alpha_{4k}$, $\beta_1, \ldots, \beta_{4k}$. They engage in a secure permute_share protocol and obtain random shares of Alice's data.

   (e) Bob adds the shares he obtains from Step 2d to his data. Alice chooses a random permutation ϕ_4 and random values $\gamma_1, \ldots, \gamma_{4k}$, $\delta_1, \ldots, \delta_{4k}$. They engage in a secure permute_share protocol and obtain random shares of Bob's data. Alice adds the shares she obtains from Step 2e to her data.

   At the end of Step 2, denote Alice's share of $4k$ cluster centers by $(c_1^A, w_1^A), \ldots, (c_{4k}^A, w_{4k}^A)$ and Bob's share by $(c_1^B, w_1^B), \ldots, (c_{4k}^B, w_{4k}^B)$.

3. Repeat the protocol merge_clusters $3k$ times to merge $4k$ clusters into k clusters.

and w_i is its weight.) At the beginning of the protocol, Alice and Bob agree on a homomorphic encryption scheme. Bob chooses a random permutation ϕ and a random vector $R = ((r_1, s_1), \ldots, (r_k, s_k))$. At the end of the protocol Bob holds $\phi(R)$ and Alice receives $\phi(C + R)$. This protocol is similar to the permutation protocol introduced by Du and Attalah [11]. We briefly describe the protocol in Algorithm 6.

COMMUNICATION AND COMPUTATIONAL COMPLEXITY Alice and Bob send k encrypted pairs to each other. If we assume that it takes c bits to represent each encryption then the total communication complexity is $O(kc)$. The estimation of the computational complexity involves the counting the number of encryptions, decryptions and exponentiations. Alice performs $O(k)$ encryptions and $O(k)$ decryptions, while Bob performs $O(k)$ encryptions. The number of exponentiations required by both Alice and Bob is $O(k)$.

PRIVACY If Alice and Bob were to use a trusted third party, then at the end of the protocol, Alice learns $\phi(C + R)$ and Bob learns $\phi(R)$. In our privacy-preserving protocol, Alice and Bob communicate using a semantically secure encryption scheme. Thus, Alice learns only her output and nothing else.

Algorithm 6 Secure protocol to share data

Protocol permute_share

Input: Alice has a vector of cluster centers C of the form $((c_1, w_1), \ldots, (c_k, w_k))$,
Bob has a random permutation ϕ and a random vector $R = ((r_1, s_1), \ldots, (r_k, s_k))$.
Output: Alice obtains $\phi(C + R)$ as output.

1. Alice chooses a key pair (pk, sk) and sends the public key pk to Bob.
2. Alice computes the encryption of the vector C as $((E(c_1), E(w_1)), \ldots, (E(c_k), E(w_k)))$ and sends to Bob.
3. Bob uses the property of the homomorphic encryption scheme to compute $\phi((E(c_1 + r_1), E(w_1 + s_1)), \ldots, (E(c_k + r_k), E(w_k + s_k)))$ and sends to Alice.
4. Alice decrypts to obtain her share $\phi((c_1 + r_1, w_1 + s_1), \ldots, (c_k + r_k, w_k + s_k))$ and Bob's share is $\phi((-r_1, -s_1), \ldots, (-r_k, -s_k))$.

5.4 Secure Protocol to Merge Clusters.

We now describe a protocol that securely merges m clusters into $m - 1$ clusters where the cluster centers are shared between Alice and Bob. Let $\{(c_1^A, w_1^A), \ldots, (c_m^A, w_m^A)\}$ denote Alice's share of the cluster centers and $\{(c_1^B, w_1^B), \ldots, (c_m^B, w_m^B)\}$ denote Bob's share of the cluster centers. Here c's denote the cluster centers and w's denote the weight of the corresponding clusters, where $c_i^A = (a_{i1}^A, \ldots, a_{i\ell}^A)$, $c_i^B = (a_{i1}^B, \ldots, a_{i\ell}^B)$ for $1 \leq i \leq m$ and ℓ denotes the number of attributes.

Alice and Bob jointly compute the merge error for all pairs of clusters. For two clusters C_i and C_j, for $1 \leq i < j \leq m$, the merge error is given by

$$\mathsf{error}(C_i \cup C_j) = C_i.\mathsf{weight} * C_j.\mathsf{weight} * (\mathsf{dist}(C_i, C_j))^2$$

$$= (w_i^A + w_i^B) * (w_j^A + w_j^B) * (\mathsf{dist}(C_i, C_j))^2$$
$$= (w_i^A * w_j^A + w_i^B * w_j^B + (w_i^B * w_j^A + w_i^A * w_j^B)) + (\mathsf{dist}(C_i, C_j))^2$$

where

$$(\mathsf{dist}(C_i, C_j))^2 =$$
$$(\mathsf{dist}((a_{i1}^A + a_{i1}^B, \ldots, a_{i\ell}^A + a_{i\ell}^B), (a_{j1}^A + a_{j1}^B, \ldots, a_{j\ell}^A + a_{j\ell}^B))^2$$
$$= \sum_{k=1}^{\ell}((a_{ik}^A + a_{ik}^B) - (a_{jk}^A + a_{jk}^B))^2$$
$$= \sum_{k=1}^{\ell}(a_{ik}^A - a_{jk}^A)^2 + \sum_{k=1}^{\ell}(a_{ik}^B - a_{jk}^B)^2 + 2\sum_{k=1}^{\ell}(a_{ik}^A - a_{jk}^A)(a_{ik}^B - a_{jk}^B)$$

The first term of the distance function is computed by Alice and the second term is computed by Bob. Alice and Bob use a secure scalar product protocol to compute the random shares of the third term. Similarly the scalar product protocol can be used to compute the random shares of $(w_i^B * w_j^A + w_i^A * w_j^B)$. We have

$$\mathsf{error}(C_i \cup C_j) = e_{ij}^A + e_{ij}^B.$$

where e_{ij}^A is the random share of the error known to Alice and e_{ij}^B is the random share of the error known to Bob, and

$$e_{ij}^A = (w_i^A * w_j^A) + \alpha_{ij} + \sum_{k=1}^{\ell}(a_{ik}^A - a_{jk}^A)^2 + \gamma_{ij}$$

$$e_{ij}^B = (w_i^B * w_j^B) + \beta_{ij} + \sum_{k=1}^{\ell}(a_{ik}^B - a_{jk}^B)^2 + \delta_{ij}$$

$$\alpha_{ij} + \beta_{ij} = (w_i^B * w_j^A + w_i^A * w_j^B))$$

$$\gamma_{ij} + \delta_{ij} = 2\sum_{k=1}^{\ell}(a_{ik}^A - a_{jk}^A)(a_{ik}^B - a_{jk}^B).$$

All of these computations are done in a finite field of size N.

Alice and Bob have $\frac{m(m-1)}{2}$-length vectors (e_{ij}^A) and (e_{ij}^B), respectively, where $1 \leq i < j \leq m$, with m being the number of clusters. They securely compute the indices i and j such that $(e_{ij}^A) + (e_{ij}^B)$ is minimum using Yao's circuit evaluation protocol [47]. Both Alice and Bob learns the indices i and j. Since m^2 is typically quite small, particularly in comparison to the number of data items, the overhead this requires should be fairly small.

The next step is to eliminate the ith and the jth cluster centers and jointly compute the new cluster center and the corresponding weight as random shares. The pth coordinate of the new cluster center is given by

$$\frac{w_i * a_{ip} + w_j * a_{jp}}{w_i + w_j}$$
$$= \frac{(w_i^A + w_i^B) * (a_{ip}^A + a_{ip}^B) + (w_j^A + w_j^B) * (a_{jp}^A + a_{jp}^B)}{w_i^A + w_i^B + w_j^A + w_j^B}$$

This can be computed as random shares using a secure scalar protocol [16] and a weighted-mean protocol [31].

COMMUNICATION AND COMPUTATIONAL COMPLEXITY For each pair of clusters this protocol computes the merge error, finds the indices for which the error is minimum and recomputes the new cluster center and weight for the merged pair of clusters. Each object has ℓ components. Communication is involved when the two parties engage in the secure scalar product protocol to compute the merge error of each pair of clusters. The scalar product protocol is invoked once between two vectors of length ℓ. The communication complexity for one invocation of the scalar product protocol is $O(c\ell)$. Hence, $O(m^2)$ pairs of error computations involve a communication complexity of $O(m^2c\ell)$, where c is the maximum number of bits required to represent each encryption.

Yao's circuit evaluation protocol [47] is invoked once per comparison and it takes $O(d)$ bits of communication, where d is the number of bits required to represent each component. The total communication for finding the minimum of m^2 elements is $O(m^2cd)$. Recomputing the cluster centers and the corresponding weights for the merged clusters involve constant bits of communication. The total communication complexity for one invocation of secure merge_clusters protocol is $O(m^2cd) + O(m^2c\ell) = O(m^2c(d + \ell))$.

For the merge_clusters protocol, a secure scalar product protocol between vectors of length ℓ is executed for each pair of clusters. For each execution of the scalar product, Alice performs ℓ encryptions and one decryption and Bob performs ℓ exponentiations and one encryption. So, the overall computational complexity for an invocation of the merge_clusters protocol is $O(m^2\ell)$ encryptions and $O(m^2)$ decryptions for Alice, and $O(m^2\ell)$ exponentiations and $O(m^2)$ encryptions for Bob.

PRIVACY The merge_clusters protocol securely merges m clusters into $m-1$ clusters without revealing any information. At the beginning of the protocol, the initial cluster centers and the weights are randomly shared between Alice and Bob. The merge error of the cluster pairs are calculated as random shares between Alice and Bob using the scalar product protocol. The scalar product protocol leaks no information and returns only random shares. Yao's circuit evaluation protocol securely computes the cluster pairs that gives a minimum error. The output of the protocol is a random share of the merged $m - 1$ cluster centers.

5.5 Overall Privacy and Performance of the Private ReCluster Protocol

PRIVACY For privacy requirements, we compare this protocol with the one that uses the trusted third party (TTP). When a TTP is used the two parties send the data objects owned by them to the TTP. The TTP computes the cluster centers and send them to both parties. Both parties do not receive any other additional information other than the k cluster centers.

In our privacy-preserving clustering protocol, both parties compute $2k$ clusters each independently from the data objects owned by them. They communicate with each other when they merge the $4k$ clusters into k clusters. The merge_clusters protocol does the merging using secure permute_share protocol, the scalar product protocol, and the Yao's protocol. These protocols are secure. They do not leak any information. The cluster centers at the end of the merge_clusters protocol are obtained as random shares between the two parties. When the parties need to compute the cluster centers they exchange their shares to each other. All the intermediate results are also available as random shares between the two parties. Neither party can learn any information from the encrypted messages that are communicated since the encryption scheme chosen is semantically secure. Hence the privacy-preserving ReCluster protocol is secure and does not leak any information.

COMMUNICATION AND COMPUTATIONAL COMPLEXITY Alice and Bob initially compute $2k$ clusters each from their own data. They invoke the permute_share protocol four times to obtain a share of each others data. This requires a communication of $O(kc)$ bits where c is the maximum number of bits required to represent each encryption. Alice and Bob jointly invoke the merge_clusters protocol $3k$ times to compute k cluster centers from $4k$ cluster centers. Each invocation of the merge_clusters protocol involves a communication of $O(k^2c(d+\ell))$ bits, where ℓ is the number of attributes in each row of the table, and d is the maximum number of bits required to represent a database entry. Hence the overall communication complexity is $O(k^3c(d+\ell))$ bits.

In Step 1 of Algorithm 5, Alice and Bob independently compute $2k$ clusters from their own data. This involves a time complexity of $O(nk^2)$. The sharing of the data between Alice and Bob in Step 2 requires $O(k)$ encryptions and $O(k)$ decryptions for Alice, and $O(k)$ encryptions and $O(k)$ decryptions for Bob. They also perform $O(k)$ exponentiations. The merge_clusters protocol is invoked $3k$ times. The computational complexity is $O(k^3\ell)$ encryptions, $O(k^3)$ decryptions for Alice, and $O(k^3\ell)$ exponentiations and $O(k^3)$ encryptions for Bob.

We next compare the performance of our new privacy-preserving protocol with earlier protocols for k-clustering [43, 31, 29, 7, 34]. All of these results, except [34], were based on the distributed k-means clustering algorithm. The k-means algorithm computes candidate cluster centers in an iterative fashion. The first three of these protocols reveal the candidate cluster to which each object has been assigned. As explained in [29], this intermediate information can sometimes reveal some of the parties' original data, thus breaching privacy. This leak can be prevented by having the cluster assignments shared between the two parties. Bunn and Ostrovsky give a secure protocol that does not leak this information [7]. However, this protocol has a large communication complexity of $O((n+k)t)$ for horizontally partitioned data, where t is the number of iterations of the k-means algorithm. In the expected setting, where n is very large, this has a substantial impact. Prasad and Rangan [34] presented a privacy-preserving protocol for BIRCH on arbitrarily-partitioned databases. This algorithm, when reduced to the case of horizontally-partitioned databases, results in a communication complexity of O(n), where n is the size of the database. This is can be rather high for large databases.

Our protocol suffers from neither of these drawbacks. Specifically, the communication complexity does not depend on n. Although, the complexity is cubic in k, for large n and small k, our protocol is not significantly slower than the k-means protocol for horizontally partitioned data [31]. Further, it does not leak any intermediate information. We note that hierarchical agglomerative clustering algorithms, such as Ward's, can be made private using the same techniques used in this paper. However, the resulting private protocol would have a communication complexity of $\Theta(n^3)$.

6 Conclusions

This paper makes two major contributions. First, we present a simple deterministic algorithm for I/O-efficient k-clustering. This algorithm examines each data item only once and uses only sequential access to data. We present extensions of the algorithm to data streams. Second, we present a privacy-preserving protocol for k-clustering based on our clustering algorithm. This protocol is communication efficient and does not reveal the intermediate candidate centers or cluster assignments.

Acknowledgements

Geetha Jagannathan and Rebecca N. Wright were funded under NSF grant CCR-0331584. We would like to thank the anonymous referees for their invaluable comments.

References

[1] C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proc. 2003 Int. Conf. on Very Large Data Bases (VLDB'03)*, pages 81–92, 2003.

[2] C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. In *Proc. 2004 Int. Conf. on Very Large Data Bases (VLDB'04)*, pages 852–863, 2004.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *ACM SIGMOD*, pages 439–450, May 2000.

[4] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29, New York, NY, USA, 1996. ACM Press.

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

[6] D. Boneh, E. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Second Theory of Cryptography Conference, TCC*, volume 3378 of *LNCS*, pages 325–341, 2005.

[7] P. Bunn and R. Ostrovsky. Secure two-party k-means clustering. In *ACM Conference on Computer and Communications Security*, pages 486–497, 2007.

[8] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 30–39, 2003.

[9] I. Dåmgard and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *PKC '01: Proceedings of the Fourth International Workshop on Practice and Theory in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, London, UK, 2001. Springer-Verlag.

[10] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Association for Computing Machinery Sixth International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[11] W. Du and M. Atallah. Privacy-preserving cooperative statistical analysis. In *17th ACSAC*, pages 102–112, 2001.

[12] W. Du and Z. Zhan. Building decision tree classifier on private data. In *Proceedings of the IEEE International Conference on Privacy, Security and Data Mining*, pages 1–8. Australian Computer Society, Inc., 2002.

[13] W. Fan. Streamminer: A classifier ensemble-based engine to mine concept-drifting data streams. In *Proc. 2004 Int. Conf. on Very Large Data Bases (VLDB'04)*, pages 1257–1260, 2004.

[14] C. Fontaine and F. Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP J. Inf. Secur.*, 2007:1–15, 2007.

[15] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[16] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen. On secure scalar product computation for privacy-preserving data mining. In *7th ICISC*, 2004.

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

[18] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE TKDE*, 15(3):515–528, 2003.

[19] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *IEEE FOCS*, pages 359–366, 2000.

[20] S. Guha, R. Rastogi, and K. Shim. Cure: an efficient clustering algorithm for large databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 73–84, New York, NY, USA, 1998. ACM Press.

[21] C. Gupta and R. L. Grossman. Genic: A single-pass generalized incremental algorithm for clustering. In *SDM*, 2004.

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

[23] J. Han and M. Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann Publishers Inc., 2000.

[24] J. A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, Inc., 1975.

[25] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. Technical report, Digital Systems Research Center, Palo Alto, CA, 1998.

[26] W. H. Inmon. *Building the Data Warehouse,3rd Edition*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[27] G. Jagannathan, K. Pillaipakkamnatt, and D. Umano. A secure clustering algorithm for distributed data streams. In *Workshops Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*, pages 705–710, 2007.

[28] G. Jagannathan, K. Pillaipakkamnatt, and R. Wright. A new privacy-preserving distributed k-clustering algorithm. In *Proc. of the 6th SIAM International Conference on Data Mining*, pages 492–496, 2006.

[29] G. Jagannathan and R. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *11th KDD*, pages 593–599, 2005.

[30] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[31] S. Jha, L. Kruger, and P. McDaniel. Privacy preserving clustering. In *ESORICS*, pages 397–417, 2005.

[32] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *Proc. ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02)*, pages 24–31, June 2002.

[33] KDD. The third international knowledge discovery and data mining tools competition dataset (KDD99 cup). http://kdd.ics.uci.edu/databases/kddcup99.html, 1999.

[34] P. Krishna Prasad and C. Pandu Rangan. Privacy preserving birch algorithm for clustering over arbitrarily partitioned databases. In *ADMA '07: Proceedings of the 3rd international conference on Advanced Data Mining and Applications*, pages 146–157, Berlin, Heidelberg, 2007. Springer-Verlag.

[35] Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.

[36] S. P. Lloyd. Least squares quantization in PCM. *IEEE Trans. on Info. Theory*, 28:129–137, 1982.

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

[38] J. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, pages 315–323, 1980.

[39] H. Nagesh, S. Goil, and A. Choudhary. Adaptive grids for clustering massive data sets. In *In 1st SIAM International Conference Proceedings on Data Mining*, 2001.

[40] R. T. Ng and J. Han. CLARANS: A method for clustering objects for spatial data mining. *IEEE TKDE*, 14(5):1003–1016, 2002.

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

[42] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *EUROCRYPT*

99, volume 1592 of *LNCS*, pages 223–238. Springer-Verlag, 1999.

[43] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *9th KDD*, 2003.

[44] J. Vaidya and C. Clifton. Privacy-preserving decision trees over vertically partitioned data. In *10th KDD*. ACM Press, 2004.

[45] J. H. Ward. Hierarchical grouping to optimize an objective function. *J. Amer. Stat. Assoc.*, 58(2):236–244, 1963.

[46] Z. Yang and R. N. Wright. Privacy-preserving computation of bayesian networks on vertically partitioned data. *IEEE Trans. on Knowledge and Data Engineering*, 18(9):1253–1264, 2006.

[47] A. C. Yao. How to generate and exchange secrets. In *27th FOCS*, pages 162–167, 1986.

[48] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: A new data clustering algorithm and its applications. *DMKD*, 1(2):141–182, 1997.