Privacy-Preserving K-Means Clustering over Vertically Partitioned Data

Jaideep Vaidya
Department of Computer Sciences
Purdue University
250 N University St
West Lafayette, IN 47907-2066
jsvaidya@cs.purdue.edu

Chris Clifton
Department of Computer Sciences
Purdue University
250 N University St
West Lafayette, IN 47907-2066
clifton@cs.purdue.edu

ABSTRACT

Privacy and security concerns can prevent sharing of data, derailing data mining projects. Distributed knowledge discovery, if done correctly, can alleviate this problem. The key is to obtain valid results, while providing guarantees on the (non)disclosure of data. We present a method for k-means clustering when different sites contain different attributes for a common set of entities. Each site learns the cluster of each entity, but learns nothing about the attributes at other sites.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; H.2.4 [**Database Management**]: Systems—*Distributed databases*

General Terms

Security

Keywords

Privacy

1. INTRODUCTION

Data mining and privacy are often perceived to be at odds, witness the recent U.S. Senate proposal of a "Data Mining Moratorium Act"[11]. Data mining *results* rarely violate privacy, as they generally reveal high-level knowledge rather than disclosing instances of data. However, the concern among privacy advocates is well founded, as bringing data together to support data mining makes misuse easier. The problem is not data mining, but the way data mining is done.

SIGKDD '03, August 24-27, 2003, Washington, DC, USA

Imagine the following scenario. A law enforcement agency wants to cluster individuals based on their financial transactions, and study the differences between the clusters and known money laundering operations. Knowing the differences and similarities between normal individuals and known money launderers would enable better direction of investigations. Currently, an individual's financial transactions may be divided between banks, credit card companies, tax collection agencies, etc. Each of these (presumably) has effective controls governing release of the information. These controls are not perfect, but violating them (either technologically or through insider misuse) reveals only a subset of an individual's financial records. The law enforcement agency could promise to provide effective controls, but overcoming these controls now gives access to an individual's entire financial history. This raises justifiable concerns among privacy advocates.

Privacy and data mining can coexist. The problem with the above scenario is not the data mining results, but how they are obtained. Current U.S. regulations require banks to report certain transactions (e.g., large cash deposits), but law enforcement does not have full access to accounts. If the results were obtained without sharing information, between the data sources, and the results could not be used to deduce private information, data mining would not reduce privacy. Using these results to devise more effective regulations on what transactions must be reported could actually improve both privacy and the ability to detect criminal activity.

While obtaining globally meaningful results without sharing information may seem impossible, it *can* be done. Algorithms have been developed to efficiently solve several types of distributed computations in a secure manner. This paper presents a method for k-means clustering in scenarios like the above, demonstrating how results from secure multiparty computation can be used to generate privacy-preserving data mining algorithms. We assume *vertically partitioned* data: The data for a single entity are split across multiple sites, and each site has information for all the entities for a specific subset of the attributes. We assume that the *existence* of an entity in a particular site's database may be revealed, it is the *values* associated with an entity that are private. The goal is to cluster the known set of common entities without revealing any of the values that the clustering is based on.

K-means clustering[9, 13] is a simple technique to group items into k clusters. The basic idea behind k-means clus-

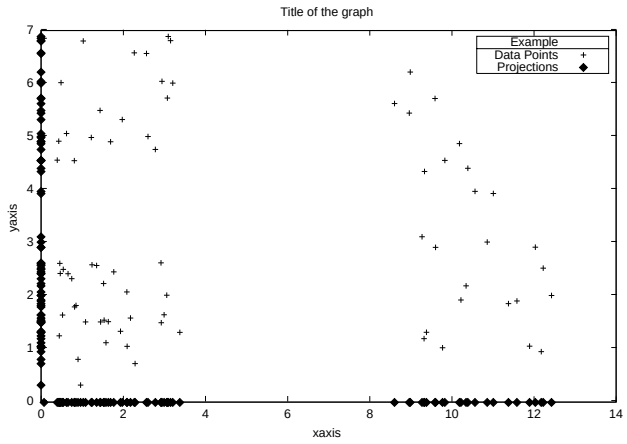

Figure 1: Two dimensional problem that cannot be decomposed into two one-dimensional problems.

tering is as follows:

Initialize the k means $\mu_1 \dots \mu_k$ to 0.
Arbitrarily select k starting points $\mu'_1 \dots \mu'_k$
repeat
  Assign $\mu'_1 \dots \mu'_k$ to $\mu_1 \dots \mu_k$ respectively
  for all points i **do**
    Assign point i to cluster j if distance $d(i, \mu_j)$ is the minimum over all j.
  end for
  Calculate new means $\mu'_1 \dots \mu'_k$.
until the difference between $\mu_1 \dots \mu_k$ and $\mu'_1 \dots \mu'_k$ is acceptably low.

Each item is placed in its closest cluster, and the cluster centers are then adjusted based on the data placement. This repeats until the positions stabilize.

The results come in two forms: Assignment of entities to clusters, and the cluster centers themselves. We assume that the cluster centers μ_i are semiprivate information, i.e., each site can learn only the components of μ that correspond to the attributes it holds. Thus, all information about a site's attributes (not just individual values) is kept private; if sharing the μ is desired, an evaluation of privacy/secrecy concerns can be performed after the values are known.

At first glance, this might appear simple – each site can simply run the k-means algorithm on its own data. This would preserve complete privacy. Figure 1 shows why this will not work. Assume we want to perform 2-means clustering on the data in the figure. From y's point of view (looking solely at the vertical axis), it appears that there are two clusters centered at about 2 and 5.5. However, in two dimensions it is clear that the difference in the horizontal axis dominates. The clusters are actually "left" and "right", with both having a mean in the y dimension of about 3. The problem is exacerbated by higher dimensionality.

Given a mapping of points to clusters, each site can independently compute the components of μ_i corresponding to its attributes. Assigning points to clusters, specifically computing which cluster gives the minimum $d(i, \mu_j)$, requires cooperation between the sites. We show how to privately compute this in Section 2.1. Briefly, the idea is that site A generates a (different) vector (of length k) for every site (including itself) such that the vector sum of all the site vectors is $\vec{0}$. Each site adds its local differences $|point - \mu_i|$ to its vector. At the same time, the vector is permuted in an order known only to A. Each site (except a single holdout) sends their permuted vector to site B. Site B sums the received vectors, then the holdout site and B perform a series of secure additions and comparisons to find the minimum i without learning distances. B now asks A the real index corresponding to i, giving the proper cluster for the point.

The second problem is knowing when to quit, i.e., when the difference between μ and μ' is small enough; we show how to privately compute this in Algorithm 2. This makes use of secure sum and secure comparison, described in Section 2.3. We will begin with details of the algorithm. We will introduce background work as necessary, particularly in the security discussion of Section 3. We discuss mitigating the risks from colluding parties in Section 4, and communication cost in Section 5. We conclude with a discussion of related work, as well as suggestions for future research.

2. PRIVACY PRESERVING *K*-MEANS ALGORITHM

We now formally define the problem. Let r be the number of parties, each having different attributes for the same set of entities. n is the number of the common entities. The parties wish to cluster their *joint* data using the k-means algorithm. Let k be the number of clusters required.

The final result of the k-means clustering algorithm is the value/position of the means of the k clusters, with each side only knowing the means corresponding to their own attributes, and the final assignment of entities to clusters. Let each cluster mean be represented as $\mu_i, i = 1, \dots, k$. Let μ_{ij} represent the projection of the mean of cluster i on party j. Thus, the final result for party j is

- the final value/position of $\mu_{ij}, i = 1 \dots k$
- cluster assignments: $clust_i$ for all points $(i = 1, \dots, n)$

The k-means algorithm also requires an initial assignment (approximation) for the values/positions of the k means. This is an important issue, as the choice of initial points determines the final solution. Research has led to mechanisms producing a good initial assignment [4]. Their technique uses classic k-means clustering done over multiple subsamples of the data, followed by clustering the results to get the initial points. For simplicity, we assume that the k means are selected arbitrarily. Since the underlying operations in [4] involve k-means clustering, it is quite easy to extend our algorithm to search for and start off with good initial means.

Thus, for $i = 1 \dots k$, every party selects its share μ'_{ij} of any given mean. This value is local to each party and is unknown to the other parties.

The basic algorithm directly follows the standard k-means algorithm. The approximations to the true means are iteratively refined until the improvement in one iteration is below a threshold. At each iteration, every point is assigned to the proper cluster, i.e., we securely find the cluster with the minimum distance for each point (this is described in Section 2.1.) Once these mappings are known, the local components of each cluster mean can be computed locally. We then use Algorithm 2 (*checkThreshold*) to test termination: was the improvement to the mean approximation in that iteration below a threshold? This is shown formally in Algorithm 1.

Algorithm 1 Privacy Preserving k-means clustering

Require: r parties, k clusters, n points.

1: **for all** sites $j = 1 \ldots r$ **do**
2: **for all** clusters $i = 1 \ldots k$ **do**
3: initialize μ'_{ij} arbitrarily
4: **end for**
5: **end for**
6: **repeat**
7: **for all** $j = 1 \ldots r$ **do**
8: **for** $i = 1 \ldots k$ **do**
9: $\mu_{ij} \leftarrow \mu'_{ij}$
10: $Cluster[i] = \emptyset$
11: **end for**
12: **end for**
13: **for** $g = 1 \ldots n$ **do**
14: **for all** $j = 1 \ldots r$ **do**
15: {Compute the distance vector $\vec{X}_j$ (to each cluster) for point g.}
16: **for** $i = 1 \ldots k$ **do**
17: $x_{ij} = |data_{gj} -_D \mu_{ij}|$
18: **end for**
19: **end for**
20: Each site puts g into $Cluster[closest\_cluster]$ {$closest\_cluster$ is Algorithm 3}
21: **end for**
22: **for all** $j = 1 \ldots r$ **do**
23: **for** $i = 1 \ldots k$ **do**
24: $\mu'_{ij} \leftarrow$ mean of j's attributes for points in $Cluster[i]$
25: **end for**
26: **end for**
27: **until** *checkThreshold* {Algorithm 2}

Algorithm 2 checkThreshold: Find out if the new means are sufficiently close to old means

Require: Th is a threshold for termination, Random number generator *rand* produces values uniformly distributed over $0..n-1$ spanning (at least) twice the domain of the distance function $-_D$.

1: **for all** $j = 1 \ldots r$ **do**
2: $d_j \leftarrow 0$
3: **for** $i = 1 \ldots k$ **do**
4: $d_j \leftarrow d_j + |\mu'_{ij} -_D \mu_{ij}|$
5: **end for**
6: **end for**
7: {Securely compute if $\sum d_j \leq Th$.}
8: At P_1: $m =$ rand()
9: **for** j=1 ... r-1 **do**
10: P_i sends $m + d_j \pmod n$ to P_{j+1}
11: **end for**
12: At P_r: $m = m + d_r$
13: At P_1: $Th' = Th + r$
14: P_1 and P_r return $secure\_add\_and\_compare(m - Th' \pmod n) > Th' - m \pmod n)$ {Secure comparison is described in Section 2.3.}

The *checkThreshold* algorithm (Algorithm 2) is straightforward, except that to maintain security (and practicality) all arithmetic must be mod$n$. This results in a non-obvious threshold evaluation at the end, consisting of a secure addition/comparison. *Intervals* are compared rather than the actual numbers. Since $Th < n/2$ and the domain of $-_D < n/2$, if the result of $m - Th'$ is positive, it will be less than $n/2$, and if it is negative, due to the modulo operation, it will be greater than $n/2$. Thus, $m - Th' > Th' - m (mod n)$ if and only if $m < Th'$, and the correct result is returned.

2.1 Securely Finding the Closest Cluster

This algorithm is used as a subroutine in the k-means clustering algorithm to privately find the cluster which is closest to the given point, i.e., which cluster should a point be assigned to. Thus, the algorithm is invoked for every single data point in each iteration. Each party has as its input the component of the distance corresponding to each of the k clusters. This is equivalent to having a matrix of distances of dimension $k \times r$. For common distance metrics; such as Euclidean, Manhattan, or Minkowski; this translates to finding the cluster where the sum of the local distances is the minimum among all the clusters.

The problem is formally defined as follows. Consider r parties $P_1, \ldots, P_r$, each with their own k-element vector $\vec{X}_i$:

$$P_1 \text{ has } \vec{X}_1 = \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{k1} \end{bmatrix}, P_2 \text{ has } \begin{bmatrix} x_{12} \\ x_{22} \\ \vdots \\ x_{k2} \end{bmatrix}, \ldots, P_r \text{ has } \begin{bmatrix} x_{1r} \\ x_{2r} \\ \vdots \\ x_{kr} \end{bmatrix}.$$

The goal is to compute the index l that represents the row with the minimum sum. Formally, find

$$\underset{i=1..k}{argmin}(\sum_{j=1..r} x_{ij})$$

For use in k-means clustering, $x_{ij} = |\mu_{ij} - point_j|$, or site P_j's component of the distance between a point and the cluster i with mean μ_i.

The security of the algorithm is based on three key ideas.

1. Disguise the site components of the distance with random values that cancel out when combined.
2. Compare distances so only the comparison result is learned; no party knows the distances being compared.
3. Permute the order of clusters so the real meaning of the comparison results is unknown.

The algorithm also requires three non-colluding sites. These parties may be among the parties holding data, but could be external as well. They need only know the number of sites r and the number of clusters k. Assuming they do not collude with each other, they learn nothing from the algorithm. For simplicity of presentation, we will assume the non-colluding sites are P_1, P_2, and P_r among the data holders. Using external sites, instead of participating sites P_1, P_2 and P_r, to be the non-colluding sites, is trivial.

The algorithm proceeds as follows. Site P_1 generates a length k random vector $\vec{V}_i$ for each site i, such that $\sum_{i=1}^{r} \vec{V}_i = \vec{0}$. P_1 also chooses a permutation π of $1..k$. P_1 then engages each site P_i in the permutation algorithm (see Section 2.2) to generate the sum of the vector $\vec{V}_i$ and P_i's distances $\vec{X}_i$.

The resulting vector is known only to P_i, and is permuted by π known only to P_1, i.e., P_i has $\pi(\vec{V_i}+\vec{X_i})$, but does not know π or $\vec{V_i}$. P_1 and $P_3 \ldots P_{r-1}$ send their vectors to P_r.

Sites P_2 and P_r now engage in a series of secure addition / comparisons to find the (permuted) index of the minimum distance. Specifically, they want to find if $\sum_{i=1}^{r} x_{li} + v_{li} < \sum_{i=1}^{r} x_{mi} + v_{mi}$. Since $\forall l, \sum_{i=1}^{r} v_{li} = 0$, the result is $\sum_{i=1}^{r} x_{li} < \sum_{i=1}^{r} x_{mi}$, showing which cluster (l or m) is closest to the point. P_r has all components of the sum except $\vec{X_2} + \vec{V_2}$. For each comparison, we use a secure circuit evaluation (see Section 2.3) that calculates $a_2 + a_r < b_2 + b_r$, without disclosing anything but the comparison result. After $k-1$ such comparisons, keeping the minimum each time, the minimum cluster is known.

P_2 and P_r now know the minimum cluster in the permutation π. They do not know the real cluster it corresponds to (or the cluster that corresponds to any of the others items in the comparisons.) For this, they send the minimum i back to site P_1. P_1 broadcasts the result $\pi^{-1}(i)$, the proper cluster for the point.

The full algorithm is given in Algorithm 3. Several optimizations are possible, we discuss these when analyzing the complexity of the algorithm in Section 5. We now describe the two key building blocks borrowed from the Secure Multiparty Computation literature. We first give the permutation algorithm. We then describe the secure addition comparison, which builds a circuit that has two inputs from each party, sums the first input of both parties and the second input of both parties, and returns the result of comparing the two sums. This (simple) circuit is evaluated securely using the generic algorithm described in Section 2.3. Following these, we will prove the security of the method. A graphical depiction of stages 1 and 2 is given in Figures 2 and 3.

Algorithm 3 *closest_cluster*: Find minimum distance cluster

Require: r parties, each with a length k vector $\vec{X}$ of distances. Three of these parties (trusted not to collude) are labeled P_1, P_2, and P_r.

{Stage 1: Between P_1 and all other parties}
P_1 generates r random vectors $\vec{V_i}$ summing to $\vec{0}$ (see Algorithm 4).
P_1 generates a random permutation π over k elements
for all $i = 2 \ldots r$ **do**
  $\vec{T_i}$ (at P_i) = *add_and_permute*$(\vec{V_i}, \pi$(at P_1)$, \vec{X_i}$(at P_i)) {This is the permutation algorithm described in Section 2.2}
end for
P_1 computes $\vec{T_1} = \pi(\vec{X_1} + \vec{V_1})$

{Stage 2: Between all but P_2 and P_r}
for all $i = 1, 3 \ldots r-1$ **do**
  P_i sends $\vec{T_i}$ to P_r
end for
P_r computes $\vec{Y} = \vec{T_1} + \sum_{i=3}^{r} \vec{T_i}$

{Stage 3: Involves only P_2 and P_r}
$minimal \leftarrow 1$
for j=2..k **do**
  if *secure_add_and_compare*$(Y_j + T_{2j} < Y_{minimal} + T_{2minimal})$ **then**
    $minimal \leftarrow j$
  end if
end for

{Stage 4: Between P_r (or P_2) and P_1}
Party P_r sends $minimal$ to P_1
P_1 broadcasts the result $\pi^{-1}(minimal)$

Algorithm 4 genRandom: Generates a (somewhat) random matrix $V_{k \times r}$

Require: Random number generator *rand* producing values uniformly distributed over $0..n-1$ spanning (at least) the domain of the distance function $-_D$.
Ensure: The sum of the resulting vectors is $\vec{0}$.
for all i = 1 ... k **do**
  $PartSum_i \leftarrow 0$
  for j = 2 ... r **do**
    $V_{ij} \leftarrow rand()$
    $PartSum_i \leftarrow PartSum_i + V_{ij} \pmod n$
  end for
  $V_{i1} \leftarrow -PartSum_i \pmod n$
end for

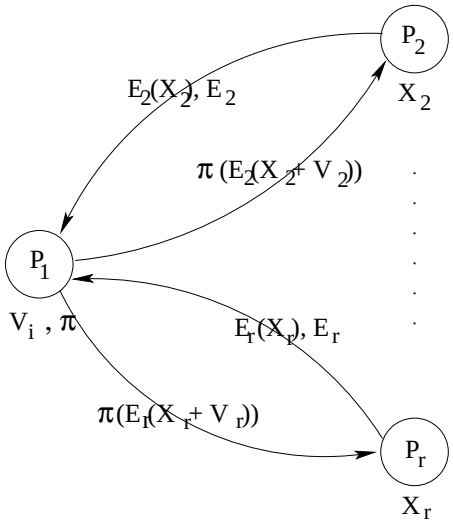

Figure 2: Closest Cluster - Stage 1

2.2 Permutation Algorithm

The secure permutation algorithm developed by Du and Atallah simultaneously computes a vector sum and permutes the order of the elements in the vector. We repeat the idea here for completeness, for more details see [7]. We do present a more formal proof of the security of the algorithm than that in [7], this is given as part of the overall security proof of our algorithm in Section 3.2.

The permutation problem is an asymmetric two party algorithm, formally defined as follows. There exist 2 parties, A and B. B has an n-dimensional vector $\vec{X} = (x_1, \ldots, x_n)$, and A has an n-dimensional vector $\vec{V} = (v_1, \ldots, v_n)$. A also has a permutation π of the n numbers. The goal is to give B the result $\pi(\vec{X} + \vec{V})$, without disclosing anything else. In particular, neither A nor B can learn the other's vector, and

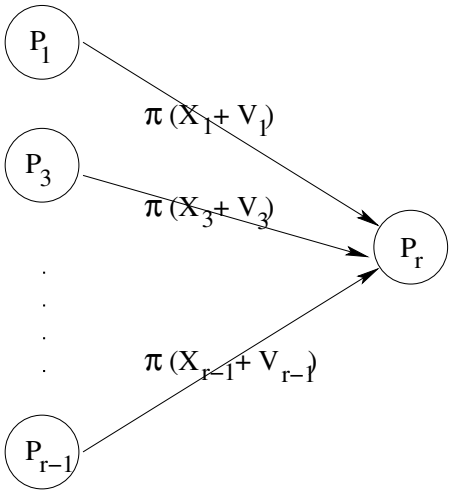

Figure 3: Closest Cluster - Stage 2

B does not learn π. For our purposes, the $\vec{V}$ is a vector of random numbers from a uniform random distribution, used to hide the permutation of the other vector.

The solution makes use of a tool known as *Homomorphic Encryption*. An encryption function $\mathcal{H} : \mathcal{R} \rightarrow \mathcal{S}$ is called *additively homomorphic* if there is an efficient algorithm *Plus* to compute $H(x + y)$ from $H(x)$ and $H(y)$ that does not reveal x or y. Many such systems exist; examples include systems by Benaloh[3], Naccache and Stern [24], Okamoto and Uchiyama[25], and Paillier [26]. This allows us to perform addition of encrypted data without decrypting it.

The permutation algorithm consists of the following steps:

1. B generates a public-private keypair (E_k, D_k) for a homomorphic encryption scheme.
2. B encrypts its vector $\vec{X}$ to generate the encrypted vector $\vec{X'} = (x'_1, \ldots, x'_n), x'_i = E_k(x_i)$.
3. B sends $\vec{X'}$ and the public key E_k to A.
4. A encrypts its vector $\vec{V}$ generating the encrypted vector $\vec{V'} = (v'_1, \ldots, v'_n), v'_i = E_k(v_i)$.
5. A now multiplies the components of the vectors $\vec{X'}$ and $\vec{V'}$ to get $\vec{T'} = (t'_1, \ldots, t'_n), t'_i = x'_i * v'_i$.

   Due to the homomorphic property of the encryption,

   $$x'_i * v'_i = E_k(x_i) * E_k(v_i) = E_k(x_i + v_i)$$

   so $\vec{T'} = (t'_1, \ldots, t'_n), t'_i = E_k(x_i + v_i)$.
6. A applies the permutation π to the vector $\vec{T'}$ to get $\vec{T'_p} = \pi(\vec{T'})$, and sends $\vec{T'_p}$ to B.
7. B decrypts the components of $\vec{T'_p}$ giving the final result $\vec{T_p} = (t_{p_1}, \ldots, t_{p_n}), t_{p_i} = x_{p_i} + v_{p_i}$.

2.3 General Secure Multiparty Computation / Secure Comparison

Secure two party computation was first investigated by Yao [29] and was later generalized to multiparty computation. The seminal paper by Goldreich proves that there exists a secure solution for *any* functionality[15]. The approach used is as follows: the function f to be computed is first represented as a combinatorial circuit, and then the parties run a short protocol for every gate in the circuit. Every participant gets (randomly selected) shares of the input wires and the output wires for every gate. Since determining which share goes to which party is done randomly, a party's own share tells it nothing. Upon completion, the parties exchange their shares, enabling each to compute the final result. This protocol can be proven to both give the desired result and to do so without disclosing anything other than the result. This approach, though appealing in its generality and simplicity, means that the size of the protocol depends on the size of the circuit, which depends on the size of the input.

This is impractical for large inputs and many parties, as in data mining. However, for a limited number of simple two-party operations, such as the *secure_add_and_compare* function used in Algorithms 2 and 3, the complexity is reasonable. For two parties, the message cost is $O(circuit\_size)$, and the number of rounds is constant. We can add and compare numbers with $O(m = \log(number\_of\_entities))$ bits using an $O(m)$ size circuit.

3. SECURITY DISCUSSION

We first need to define formally what we mean by secure. For this, we turn to the definitions of Secure Multiparty Computation.

3.1 Secure Multi-Party Computation

To prove that our k-means algorithm preserves privacy, we need to define privacy preservation. We use the framework defined in *Secure Multiparty Computation*.

Yao first postulated the two-party comparison problem (Yao's Millionaire Protocol) and developed a provably secure solution[29]. This was extended to multiparty computations by Goldreich et al.[15]. They developed a framework for secure multiparty computation, and in [14] proved that computing a function privately is equivalent to computing it securely.

We start with the definitions for security in the semi-honest model. A semi-honest party follows the rules of the protocol using its correct input, but is free to later use what it sees during execution of the protocol to compromise security. A formal definition of private two-party computation in the semi-honest model is given below.

Definition 1. (privacy w.r.t. semi-honest behavior):[14]
Let $f : \{0,1\}^* \times \{0,1\}^* \longmapsto \{0,1\}^* \times \{0,1\}^*$ be probabilistic, polynomial-time functionality, where $f_1(x, y)$ (respectively, $f_2(x, y)$) denotes the first (resp., second) element of $f(x, y)$); and let Π be two-party protocol for computing f.

Let the view of the first (resp., second) party during an execution of protocol Π on (x, y), denoted $view_1^{\Pi}(x, y)$ (resp., $view_2^{\Pi}(x, y)$), be $(x, r_1, m_1, \ldots, m_t)$ (resp., $(y, r_2, m_1, \ldots, m_t)$). r_1 represent the outcome of the first (resp., r_2 the second) party's internal coin tosses, and m_i represent the i^{th} message it has received.

The output of the first (resp., second) party during an execution of Π on (x, y) is denoted $output_1^{\Pi}(x, y)$ (resp., $output_2^{\Pi}(x, y)$) and is implicit in the party's view of the execution.

Π privately computes f if there exist probabilistic polynomial time algorithms, denoted S_1, S_2 such that

$$\{(S_1(x, f_1(x,y)), f_2(x,y))\}_{x,y\in\{0,1\}^*} \equiv^C \left\{\left(view_1^{\Pi}(x,y), output_2^{\Pi}(x,y)\right)\right\}_{x,y\in\{0,1\}^*}$$

$$\{(f_1(x,y), S_2(x, f_1(x,y)))\}_{x,y\in\{0,1\}^*} \equiv^C \left\{\left(output_1^{\Pi}(x,y), view_2^{\Pi}(x,y)\right)\right\}_{x,y\in\{0,1\}^*}$$

where $\equiv^C$ denotes computational indistinguishability.

The above definition states that a computation is secure if the view of each party during the execution of the protocol can be effectively simulated knowing only the input and the output of that party. This is not quite the same as saying that private information is protected. If information can be deduced from the final result, it is obviously not kept private under this definition. For example, if two entities map to different clusters, they must have some attribute values that are different. If one site has exactly the same values for those entities, it has learned the "private" information that those entities have different values in the attributes held by some other site. This cannot be helped, as this information can always be deduced from the result and the site's own input.

A key result we use is the composition theorem. We state it for the semi-honest model. A detailed discussion of this theorem, as well as the proof, can be found in [14].

THEOREM 1. *(Composition Theorem for the semi-honest model): Suppose that g is privately reducible to f and that there exists a protocol for privately computing f. Then there exists a protocol for privately computing g.*

PROOF. Refer to [14]. □

Our protocols are somewhat stronger than the semi-honest model. The proofs hold in any situation where the parties do not collude to discover information; a single malicious party may disrupt the results, but cannot learn private information that would not be revealed by the result. This will become apparent in the proofs below.

3.2 Permutation Algorithm

The permutation algorithm reveals nothing to A, so A's view must be simulated using only it's own input. B gets the result vector.

THEOREM 2. *The Permutation Algorithm (Section 2.2) privately computes a permuted vector sum of two vectors, where one party knows the permutation π and the other gets permuted sum $\pi(\vec{X}+\vec{V})$.*

PROOF.

$A's$ view:

A receives an encryption key E_k and a encrypted vector $\vec{X}'$ of size n. It can simulate the encryption key by generating a single random number from a uniform random distribution. Assuming security of encryption and since A knows the n, the vector $\vec{X}'$ can also be simulated simply by generating n randoms from an uniform distribution. Using its own vector $\vec{V}$ and the simulated input, the simulator for A can perform steps 4–6 to complete the simulation of A's view.

$B's$ view:

The simulator for B performs steps 1 and 2 to generate E_k and $\vec{X}'$. In step 6 B receives a size n vector $\vec{T}'_p$. To simulate $\vec{T}'_p$, B encrypts the components of the result $T_p = \pi(\vec{X}+\vec{V})$: $t'_{pi} = E_k(t_{pi})$.

The simulator for both runs in time linear in the size of the input vectors, meeting the requirement for a polynomial-time simulation. □

3.3 Closest Cluster Computation

Algorithm 3 returns the index of the closest cluster (i.e., the row with the minimum row sum). To prove this algorithm is privacy preserving, we must show that each party can construct a polynomial time simulator for the view that it sees, given only its own input and this closest cluster index.

THEOREM 3. *Algorithm 3 privately computes the index of the row with the minimum row sum, revealing only this result assuming parties do not collude to expose other information.*

PROOF. The simulator is constructed in stages, corresponding to the stages of the algorithm.

Stage 1. The only communication occurring in this stage occurs in the $r-1$ calls to the Permutation Algorithm. Thus, we simply need to apply the composition theorem stated in Theorem 1, with g being the closest cluster computation algorithm and f being the permutation algorithm. What remains is to show that we can simulate the result T_i. The simulator for P_1 is exactly the algorithm used by P_1, without sending any data. For the remaining sites, since the v_i are unknown and chosen from a uniform distribution on $(0..n-1)$, $v_i + x_i$ will also form a uniform distribution on $(0..n-1)$. Each $P_i, i = 2 \ldots r$ can simulate the vector $\vec{T}_i$ by selecting values randomly from a uniform distribution on $(0..n-1)$. This is indistinguishable from what it sees in the algorithm.

Stage 2. All the parties other than P_2 and P_r send their permuted result vectors to the receiver. Since only P_r sees new information, we need only concern ourselves with simulating what it sees. The received vectors can be simulated by P_r exactly as they were simulated by the P_i in Stage 1. The vector $\vec{Y}$ is equal to the actual distances minus $\vec{T}_2$. However, since $\vec{T}_2$ consists of values uniformly distributed over $(0..n-1)$, $\vec{Y}$ is effectively $distances - v$, and is thus also uniformly distributed over $(0..n-1)$. However, we cannot simulate it by generating random values, as we must preserve the relationship $\vec{Y} = \vec{T}_1 + \sum_{j=3}^{r} T_j \pmod n$. Fortunately, the sum of the simulator-generated $\vec{T}_i$ will give a vector $\vec{Y}$ that both meets this constraint and is uniformly distributed over $(0..n-1)$, giving a view that is indistinguishable from the real algorithm.

Stage 3. Here P_2 and P_r engage in a series of comparisons. Again, we use the composition theorem. Each comparison is secure, so we need only show that we can simulate the sequence of comparison results.

The simulator uniformly chooses a random ordering of the k clusters from the $k!$ possible orderings. We regard this as the distance-wise ordering of the clusters relative to the point. This ordering is used to choose the appropriate result, $\leq$ or $>$, for each comparison. Effectively, the simulator runs steps 17-21, but makes the comparisons locally based on the random ordering. The probability of any given

ordering is $1/k!$, the same as the probability of any given ordering achieved after the permutation π in the actual view. Therefore, the probability of any given sequence of comparison results is the same under the ordering as under the view seen in the actual algorithm.

Note that all of the possible $2^{(k-1)}$ sequences are not equally likely, e.g., the sequence of all >s corresponds to only *one* ordering, while the sequence of all $\leq$s corresponds to $(k-1)!$ orderings. However, selecting random total orderings generates sequences matching the (non-uniform) probability distribution of the actual sequences of comparisons.

Stage 4. P_r (or P_2) sends the index i to P_1. Since the true index i_t is the final result known to all the parties, and P_1 decides upon the permutation π, the simulator generates $\pi(i_t) = i$ as the message it receives.

The final result i_t is sent to all parties. Since this is the final result, obviously all the parties can simulate it.

Since this simulator is also linear in the size of the input, and we have proven the permutation algorithm to be secure, application of the composition theorem proves that Algorithm 3 preserves privacy. □

3.4 Stopping Criterion

Before analyzing the security of the entire k-means algorithm, we prove the security of the threshold checking Algorithm 2.

THEOREM 4. *Algorithm 2 determines if* $\sum |\mu'_{ij} -_D \mu_{ij}| < Th|$, *revealing nothing except the truth of this statement.*

PROOF. Steps 10 and 14 are the only steps of Algorithm 2 requiring communication, so the simulator runs the algorithm to this point. In step 10, party P_1 first sends $m + d_1 \pmod n$ where m is the random number known to P_1. Each of the parties $P_j, j = 2 \ldots r$ receive a message $m + \sum_{j=1}^{i} d_j$ from their left neighbor. Since m is chosen from a uniform distribution on $(0 \ldots n-1)$, and all arithmetic is $\bmod n$, this sum forms a uniform distribution on $(0 \ldots n-1)$ and can be simulated by generating a random number over that distribution:

$$\begin{aligned} Pr\left[VIEW_j^{\text{Algorithm 2 Step 10}} = x\right] &= Pr\left[m + \sum_{i=1}^{j} d_i = x\right] \\ &= Pr\left[m = x - \sum_{i=1}^{j} d_i\right] \\ &= \frac{1}{n} \\ &= Pr\left[Simulator_j = x\right] \end{aligned}$$

The *secure_add_and_compare* algorithm gives only the final result $m - Th' \pmod n > Th' - m \pmod n = \sum_{j=1}^{r} d_j \leq Th$. Step 14 is easily simulated knowing that result.

This simulator runs in the $O(k)$ time required by the Algorithm, and is thus polynomial. Applying the composition theorem with Algorithm 2 as f and *secure_add_and_compare* algorithm as g, along with the other facts given above, proves that Algorithm 2 is secure. □

3.5 Overall k-means algorithm

We now analyze the security of the entire k-means algorithm. In every iteration, the following things are revealed to the parties:

- Each party's local share of the k cluster means.
- The cluster assignment for every point.

These values are the desired result of the final iteration. Since it is impossible to know in advance the number of iterations required to halt, the number of iterations needs to be accepted as part of the final output. The results from the intermediate iterations *may* be used to infer information beyond this result. For example, if the cluster centers for site j do not change between iterations, and a point moves between two clusters, site j knows that those two clusters are both relatively close to the point across the sum of the other sites. However, since the location of the point in the other dimensions is not known, this information is of little use. In any iteration the final assignment of points to clusters is the same for every party. If this intermediate assignment should not be revealed, either a genuine third party will be required or else the algorithm will be quite inefficient. Allowing the intermediate results to be accepted as part of overall results allows an efficient algorithm with provable security properties. Forbidding knowledge of intermediate results would prevent each site from computing the next iteration locally, making the entire computation much more expensive.

We therefore state the proven overall security properties in the following theorem.

THEOREM 5. *Algorithm 1 is a private algorithm computing the k clusters of the combined data set, revealing at most the point assignment to clusters at each iteration and the number of iterations required to converge.*

PROOF. All of the communication in Algorithm 1 all occurs in the calls to Algorithms 3 and 2. The results of Algorithm 3 are point assignments to clusters, and can be simulated from the known result for that iteration. The results of Algorithm 2 are easily simulated; for all but the final iteration it returns false, in the final iteration it returns true. Applying the composition theorem shows that within the defined bounds the k-means algorithm is secure. □

4. HANDLING COLLUSION

Parties P_1, and P_r have more information than the others during the execution of the above algorithm. Specifically, P_1 knows

1. the permutation π, and
2. the values of the random splits (i.e., the random matrix $V_{k\times r}$).

P_r learns

1. the permuted result vectors of the permutation algorithm ($\vec{T_i}$) for all the parties other than P_2, and
2. the comparison results.

(Note that P_2 also learns the comparison results.) While we have proven that this information is meaningless *in isolation*, collusion between P_1 and P_r provides enough information to derive the distances between each point and each party's means. It is necessary to carefully select these two parties so that all parties are confident the two will not collude.

The assumption of non-collusion is often implicitly made in the real world. For example, take the case of lawyers

for parties on opposite sides in court. While no *technical* means prevent collusion, safeguards exist in the form of severe punishments for breaking this rule as well as the business penalty of lost reputation. Similar legal and reputation safeguards could be enforced for privacy-preserving data mining. In addition, if there were not at least two parties who did not want to share information, there would be no need for a secure algorithm. Since collusion between P_1 and P_r reveals P_1's information to P_r, P_1 would be unlikely to collude simply out of self-interest.

However, technical solutions are more satisfying. Let p, $1 \leq p \leq r-1$, be a user defined anti-collusion security parameter. We present a modification of the algorithm that guarantees that at least $p+1$ parties need to collude to disclose additional information. The problem is in Algorithm 3. The key idea is that stage 1 is run p times, each time selecting a new party to act as P_1. Thus, the permutation π and the random matrix $V_{k\times r}$ is different for every run, however the row sum of each V matrix is $\vec{0}$, so the total sum is still the actual distance. In stage 4, to get the true index from the permuted index, the p parties apply their inverse permutations in order. Thus, the true index is $\pi_1^{-1}(\pi_2^{-1}(\ldots(\pi_p^{-1}(i'))\ldots))$.

5. COMMUNICATION ANALYSIS

We give a bottom-up analysis of the communication cost of one iteration of the algorithm. The total cost is dependent on the number of iterations required to converge, which is dependent on the data. Assume r parties, n data elements, and that encrypted distances can be represented in m bits.

The permutation algorithm requires only two rounds of communication. For length-n vectors, the total bit cost is $2n * m + public\_key\_size = O(n)$ bits.

The *secure_add_and_compare* algorithm is a two party protocol, implemented using secure circuit evaluation. There are several general techniques for implementing circuit evaluation that optimize different parameters such as computation cost, communication cost (number of rounds or total number of bits), etc. The basic tool used, one out of two oblivious transfer, can also be implemented in several ways. Methods exist that require a constant number of rounds of communication (by parallelizing the oblivious transfers) with bit communication cost linear in the number of gates in the circuit. An excellent survey is given in[12]. The *secure_add_and_compare* algorithm requires two addition circuits and one comparison circuit, all of $m = \log n$ bits (where n is based on the resolution of the distance). Both addition and comparison require a number of gates that is linear in m. Therefore this step requires a constant number of rounds and $O(m)$ bits of communication.

In Algorithm 3, *closest_cluster*, communication occurs in several places. Steps $4-5$ make $r-1$ calls to the permutation algorithm with size k vectors. Steps $10-11$ require $r-2$ rounds of communication and $(r-2)*k*m$ bits. Steps $17-18$ use $k-1$ calls to the *secure_add_and_compare* algorithm. Steps $24-25$ require two rounds with $O(r \log k)$ bit cost. Thus the total cost is $2(r-1)+r-2+(k-1)*const \approx 3r+const*k = O(r+k)$ rounds and $2k*m*(r-1)+k*m*(r-2)+(k-1)*const*(\log n) \approx 3*m*kr+kc\log n = O(kr)$ bits.

The collusion resistant variant of Section 4 multiplies the cost of steps $4-5$ and step 24 by a factor of p. This gives $O(pr+k)$ rounds and $O(pkr)$ bits.

We now give a communication analysis of Algorithm 2. Step 10 involves $r-1$ rounds of communication, with bit cost $(r-1)*m$. Step 14 makes one call to *secure_add_and_compare*, for constant rounds and $O(m)$ bits. Thus, the total cost is $O(r)$ rounds and $O(rm)$ bits.

Finally, we come to the analysis of the entire algorithm. We do not count any setup needed to decide the ordering or role of the parties. One iteration of the k-means algorithm requires one call to the closest cluster computation for every, point and one call to the *checkThreshold* algorithm. Since all points can be processed in parallel, the total number of rounds required is $O(r+k)$. The bit communication cost is $O(nrk)$.

5.1 Optimizations

The cost of secure comparisons in Stage 3 of Algorithm 3 can be eliminated with a security compromise that would often be innocuous. First, the random vector generated in step 2 is generated so the rows sum to randomly chosen r instead of 0. In Stage 2, all parties (including P_2) send their permuted vectors to P_r. Now P_r can independently find the index of the row with the minimum row sum. Thus, the communication cost is $2(r-1)+r-1+2 \approx 3r = O(r)$ rounds and $2k*m*(r-1)+k*m*(r-1)+2(\log k) \approx 3krm$ bits.

The problem with this approach is that P_r learns the relative distance of a point to each cluster, i.e., it learns that p is 15 units farther from the second nearest cluster than from the cluster it belongs to. It does not know which cluster the second nearest is. Effectively, it gets k equations (one for each cluster) in $k+1$ unknowns. (The unknowns are the location of the point, the location of all clusters but the one it belongs in, and the distance to the closest cluster center.) Since the permutation of clusters is different for each point, as is the random R, combining information from multiples points still does not enable solving to find the exact location of a point or cluster. However, probabilistic estimates on the locations of points/clusters are possible. If the parties are willing to accept this loss of security in exchange for the communication efficiency, they can easily do so.

Let us now compare our communication cost with that of the general circuit evaluation method. For one iteration of the algorithm a circuit evaluation would be required for each point to evaluate the cluster to which the point is assigned. Even with an optimized circuit, the closest cluster computation requires at least $r-1$ addition blocks for each cluster. I.e., it requires approximately kr addition circuits, and $k-1$ comparison blocks. These blocks are all of width at least m bits. The best known general method still requires at least r^2 bits of communication for every circuit. Thus, a lower bound on the amount of bits transferred is $O(kmr^3)$ bits.

A simple upper bound on *non*-secure distributed k-means is obtained by having every party send its data to one site. This gives $O(n)$ bits in one round. Privacy is adding a factor of $O(r+k)$ rounds and $O(rk)$ bit communication cost. While this tradeoff may seem expensive, if the alternative is not to perform data mining at all, it seems quite reasonable.

6. CONCLUSIONS

There has been work in distributed clustering that does not consider privacy issues, e.g., [6, 19]. Generally, the goal of this work is to reduce communication cost. The idea is to find important data points or patterns locally and utilize

these to compute the global patterns. However, sharing local patterns inherently compromises privacy. Our work ensures reasonable privacy while limiting communication cost.

There recently been a surge in interest in privacy-preserving data mining. One approach is to add "noise" to the data before the data mining process, and using techniques that mitigate the impact of the noise from the data mining results[2, 1, 10, 27].

The approach of protecting privacy of distributed sources was first addressed for the construction of decision trees[22]. This work closely followed the secure multiparty computation approach discussed below, achieving "perfect" privacy, i.e., nothing is learned that could not be deduced from one's own data and the resulting tree. The key insight was to trade off computation and communication cost for accuracy, improving efficiency over the generic secure multiparty computation method. There has since been work to address association rules in horizontally partitioned data[17, 16], EM Clustering in Horizontally Partitioned Data[21], association rules in vertically partitioned data[28], and generalized approaches to reducing the number of "on-line" parties required for computation[18]. While some of this work makes trade-offs between efficiency and information disclosure, all maintain provable privacy of individual information and bounds on disclosure, and disclosure is limited to information that is unlikely to be of practical concern.

Clustering in the presence of differing scales, variability, correlation and/or outliers can lead to unintuitive results if an inappropriate space is used. Research has developed robust space transformations that permit good clustering in the face of such problems [20]. Such estimators need to be calculated over the entire data. An important extension to our work would be to allow privacy preserving computation of such estimators, giving higher confidence in clustering results. Similarly, extending this work to the more robust EM-clustering algorithm [5, 23] under the heterogeneous database model is a promising future direction. Another problem is to find the set of common entities without revealing the identity of entities that are not common to all parties.

We have made use of several primitives from the Secure Multiparty Computation literature. Recently, there has been a renewed interest in this field, a good discussion can be found in [8]. Currently, assembling these into efficient privacy-preserving data mining algorithms, and proving them secure, is a challenging task. Our paper has demonstrated how these can be combined to implement a standard data mining algorithm with provable privacy and information disclosure properties. Our hope is that as the library of primitives and known means for using them grow, standard methods will develop to ease the task of developing privacy-preserving data mining techniques.

7. ACKNOWLEDGMENTS

We thank Patricia Clifton for comments, corrections, and catching a flaw in an earlier proof of Stage 3. We also thank the anonymous reviewers for their detailed suggestions for improving the paper and acknowledge Murat Kantarcioglu for a synopsis of secure multiparty computation definitions and relationship to this work.

8. REFERENCES

[1] D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 247–255, Santa Barbara, California, USA, May 21-23 2001. ACM.

[2] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data*, pages 439–450, Dallas, TX, May 14-19 2000. ACM.

[3] J. Benaloh. Dense probabilistic encryption. In *Proceedings of the Workshop on Selected Areas of Cryptography*, pages 120–128, Kingston, Ontario, May 1994.

[4] P. S. Bradley and U. M. Fayyad. Refining initial points for K-Means clustering. In *Proc. 15th International Conf. on Machine Learning*, pages 91–99. Morgan Kaufmann, San Francisco, CA, 1998.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society*, B 39:1–38, 1977.

[6] I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Proceedings of Large-scale Parallel KDD Systems Workshop, ACM SIGKDD*, Aug. 15-18 1999. (Also published as Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence, Volume 1759, pp. 245-260, 2000).

[7] W. Du and M. J. Atallah. Privacy-preserving statistical analysis. In *Proceeding of the 17th Annual Computer Security Applications Conference*, New Orleans, Louisiana, USA, December 10-14 2001.

[8] W. Du and M. J. Atallah. Secure multi-party computation problems and their applications: A review and open problems. In *New Security Paradigms Workshop*, pages 11–20, Cloudcroft, New Mexico, USA, September 11-13 2001.

[9] R. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[10] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–228, Edmonton, Alberta, Canada, July 23-26 2002.

[11] M. Feingold, M. Corzine, M. Wyden, and M. Nelson. Data-mining moratorium act of 2003. U.S. Senate Bill (proposed), Jan. 16 2003.

[12] M. Franklin and M. Yung. Varieties of secure distributed computing. In *Proc. Sequences II, Methods in Communications, Security and Computer Science*, pages 392–417, Positano, Italy, June 1991.

[13] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, San Diego, CA, 1990.

[14] O. Goldreich. Secure multi-party computation, Sept. 1998. (working draft).

[15] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game - a completeness theorem for protocols with honest majority. In *19th ACM*

Symposium on the Theory of Computing, pages 218–229, 1987.

[16] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *The ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02)*, pages 24–31, Madison, Wisconsin, June 2 2002.

[17] M. Kantarcıoğlu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE-TKDE*, submitted.

[18] M. Kantarcioglu and J. Vaidya. An architecture for privacy-preserving mining of client information. In C. Clifton and V. Estivill-Castro, editors, *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, volume 14, pages 37–42, Maebashi City, Japan, Dec. 9 2002. Australian Computer Society.

[19] H. Kargupta, W. Huang, K. Sivakumar, and E. Johnson. Distributed clustering using collective principal component analysis. *Knowledge and Information Systems*, 3(4):405–421, Nov. 2001.

[20] E. M. Knorr, R. T. Ng, and R. H. Zamar. Robust space transformations for distance-based operations. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 126–135, San Francisco, California, 2001. ACM Press.

[21] X. Lin and C. Clifton. Privacy preserving clustering with distributed EM mixture modeling. *Knowledge and Information Systems*, Submitted.

[22] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO 2000*, pages 36–54. Springer-Verlag, Aug. 20-24 2000.

[23] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Wiley & Sons, 1997.

[24] D. Naccache and J. Stern. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM conference on Computer and communications security*, pages 59–66, San Francisco, California, United States, 1998. ACM Press.

[25] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology - Eurocrypt '98, LNCS 1403*, pages 308–318. Springer-Verlag, 1998.

[26] P. Paillier. Public key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - Eurocrypt '99 Proceedings, LNCS 1592*, pages 223–238. Springer-Verlag, 1999.

[27] S. J. Rizvi and J. R. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 682–693, Hong Kong, Aug. 20-23 2002. VLDB.

[28] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644, Edmonton, Alberta, Canada, July 23-26 2002.

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