Distributed Privacy Preserving k-Means Clustering with Additive Secret Sharing[*]

Mahir Can Doganay
mahircan@su.sabanciuniv.edu

Thomas B. Pedersen
pedersen@sabanciuniv.edu

Yücel Saygın
ysaygin@sabanciuniv.edu

Erkay Savaş
erkays@sabanciuniv.edu

Albert Levi
levi@sabanciuniv.edu

Faculty of Engineering and Natural Sciences
Sabanci University, Istanbul, Turkey

ABSTRACT
Recent concerns about privacy issues motivated data mining researchers to develop methods for performing data mining while preserving the privacy of individuals. However, the current techniques for privacy preserving data mining suffer from high communication and computation overheads which are prohibitive considering even a modest database size. Furthermore, the proposed techniques have strict assumptions on the involved parties which need to be relaxed in order to reflect the real-world requirements. In this paper we concentrate on a distributed scenario where the data is partitioned vertically over multiple sites and the involved sites would like to perform clustering without revealing their local databases. For this setting, we propose a new protocol for privacy preserving k-means clustering based on additive secret sharing. We show that the new protocol is more secure than the state of the art. Experiments conducted on real and synthetic data sets show that, in realistic scenarios, the communication and computation cost of our protocol is considerably less than the state of the art which is crucial for data mining applications.

1. INTRODUCTION
Massive amounts of data are collected for various reasons by many organizations with the hope that data mining technology will extract useful knowledge from the collected data and turn it into something beneficial for the organization. In fact, data mining technology proved its success in numerous areas such as business intelligence, life-sciences, and security. On the other hand, the popularity of data mining was about to pave the way to its demise. Part of the reason for that is the launch of large scale projects related to homeland security. Some projects were actually stopped since they failed to meet privacy concerns. According to a very recent article in Computer World by Jaikumar Vijayan "The chairman of the House Committee on Homeland Security, has asked Department of Homeland Security Secretary Michael Chertoff to provide a detailed listing of all IT programs that have been canceled, discontinued or modified because of privacy concerns"[13]. In addition to that, the Chairman also asked for information about the measures being taken to address privacy issues[13]. As a result of increased privacy concerns, data mining researchers focused on developing techniques that would enable data mining while preserving the privacy of individuals and started a popular branch of research named "privacy preserving data mining"[1]. Protocols based on statistics and cryptography were proposed for privacy preserving classification, clustering, and pattern mining in centralized and distributed environments. However, privacy preserving data management in general, is still an ongoing research topic, and efficient, as well as provably secure, methods without strong assumptions are yet to be proposed.

In this paper, we consider a distributed scenario in which the data is vertically partitioned (different attributes for the same entity can be stored at different sites). In this case each site has a different projection of the database. We choose the popular k-means clustering algorithm and propose a new protocol for distributed privacy preserving k-means clustering. Instead of using computationally costly public key encryption schemes, we utilize additive secret sharing as a cryptographic primitive to implement a secure multiparty computation protocol in order to do privacy preserving clustering. State-of-the art privacy preserving clustering algorithms were not implemented and tested before, so our initial contribution is to implement an existing technique as a benchmark. We then show that our protocol outperforms the state-of-the art in terms of computation cost. The communication cost of our protocol is better than the state-of-the art up to 64 parties which is a realistic number for most applications. We also show that our protocol provides more security with less assumptions on the involved parties.

[*]This work was partially funded by the Information Society Technologies Programme of the European Commission, Future and Emerging Technologies under IST-014915 GeoPKDD project.

The rest of this paper is organized as follows: Section 2 outlines related work. In Section 3 we explain the central tools used in our scheme and in [12]. In Section 4 we state the problem and present our algorithm. In Section 5 we discuss how our method preserves the privacy of the data records of individuals. In Section 6, we analyze the computation and communication overhead of our algorithm and compare to [12]. Simulation results that support the analysis of this section are given in Section 7. Finally in Section 8, we give our conclusions.

2. RELATED WORK

Privacy preserving data mining algorithms fall in two categories: (1) Random perturbation-based and (2) secure multi-party computation based.

Perturbation techniques mix additive or multiplicative noise with the data so that actual values in the data set are not learned, yet the data mining results gathered from the perturbed data will not deviate significantly from the results gathered from the original data. The work of Aggrawal and Srikant[1] falls into this category. One recent significant work by Kargupta et al.[9] shows that random projection based multiplicative data perturbation is a very efficient way to perform privacy preserving distributed data mining. The results obtained from perturbed data have below 5% error rate as compared to the results obtained from the original data. Algorithms in that category present a very practical and efficient way of performing privacy preserving data mining; however, they are based on trading accuracy of the data mining results with privacy of individuals. Furthermore, algorithms in this category do not preserve privacy in any formal cryptological sense, i.e. one cannot easily calculate how much effort and resources are needed to filter out the noise and breach privacy.

Algorithms based on secure multiparty computation (SMC) do not have to trade accuracy with privacy and they preserve the security and privacy of the data of individuals in a formal way. It is well known from the SMC literature that any (reasonable) function can be computed among distributed parties without revealing the private inputs[15, 2]. Therefore, theoretically it is possible to perform distributed data mining without revealing data at individual sites. Clifton et al.[3] proposed that data mining algorithms should make use of a relatively small set of primitive functions and therefore generalized SMC algorithms for these primitive functions can be used to solve data mining problems. However, generalized solutions to SMC problems suffer from high levels of communication and computation cost. Therefore, these solutions should be used with caution in data mining problems where the size of the data is measured in gigabytes. One of the algorithms in this category of privacy preserving data mining algorithms is Clifton and Kantarcioglu's protocol[5] for association rule mining. They make use of the commutative encryption property of RSA encryption and this at the worst case needs a number of public key encryptions exponential in the number of data holders. Since public key encryptions, such as RSA, consists of expensive operations, these protocols suffer from high computation cost. However, not all algorithms in this category are inefficient. The proposed protocol of Clifton and Vaidya[12] for k-means clustering over vertically partitioned data brings a communication overhead linear in the size of the data.

The use of secret sharing to perform secure multiparty data mining gained some momentum in recent years. One of the most notable examples is the work of Laur, Lipmaa and Mielikainen[8] in which they use secret sharing for private support vector classification. One other notable use of secret sharing in a privacy preserving data mining algorithm is the work of Wright and Yang[14] to compute Bayesian networks over vertically partitioned data. Similar to the work of Clifton and Vaidya[12], we address privacy preserving k-means clustering problem over vertically partitioned data, where each involved party has a subset of attributes of all the entities in the dataset.

3. PRELIMINARIES

As Clifton and Vaidya[12], we assume that some of the parties in the protocol are non-colluding. In [12] they assume the existence of 3 parties who are pairwise non-colluding. In this paper we assume the existence of 4 such pairwise non-colluding parties. Since the security of our protocol relies on secret sharing, our protocol is secure even if the parties have unlimited computing power. In contrast, Clifton and Vaidya assume that all parties are computationally bounded. As always in an information theoretical scenario, we assume the existence of authentic and confidential channels. Such channels can always be implemented with a combination of symmetric and public key cryptography. We will not address this issue further.

Both our scheme and the scheme of Clifton and Vaidya[12] are based on *homomorphic* cryptographic methods. The improvement of our scheme over that of [12] comes from the use of homomorphic secret sharing instead of homomorphic encryption.

A *public key encryption scheme* is a set of three functions G, E, and D. The function G is a *key generation* function and when G is called with a random argument it generates a *key-pair*: $(pk, sk) = G(r)$, where pk is called the *public key*, and sk is called the *secret key*. The two keys satisfy the following *decryption* condition: $D(sk, c) = m$, where $c = E(pk, m, r)$ is called the *ciphertext*, m is called the *message* or *plaintext*, and r is a random number. Furthermore it is *computationally infeasible* to compute the message m when given only pk and $E(pk, m, r)$. An encryption scheme is said to be *additively homomorphic* if $E(pk, m_0, r)E(pk, m_1, r') = E(pk, m_0 + m_1, r'')$, for some value r''. Additively homomorphic encryption schemes are the central tool of [12]. The well-known Paillier homomorphic encryption scheme take messages of size n bits, and creates $2n$-bit ciphertexts. In practical applications of [12] messages are of 32 bits, so a ciphertext is approximately 64 times larger than the message.

A (t, n) *secret sharing scheme* is a set of two functions S and R. The function S is a *sharing function* and takes a *secret* s as input and creates n *secret shares*: $S(s) = (s_1, \ldots, s_n)$. The two functions satisfy that for any set $I \subseteq \{1, \ldots, n\}$ of t indices $R(I, s_{I_1}, \ldots, s_{I_t}) = s$. Furthermore we require that it is *impossible* to recover s from a set of $t-1$ secret shares. A secret sharing scheme is *additively homomorphic* if $R(I, s_{I_1} + s'_{I_1}, \ldots, s_{I_t} + s'_{I_t}) = s + s'$.

A very simple (n, n) secret sharing scheme which is additively homomorphic is $S(s) = (r_1, \ldots, r_{n-1}, r)$, where r_i is random for $i \in \{1, \ldots, n-1\}$, and $r = s - \sum_{i=1}^{n-1} r_i$. To recover s all secret shares are added: $s = r + \sum_{i=1}^{n-1} r_i$. If even one secret share is missing nothing is known about s. We use this simple additive secret sharing scheme in this paper. Notice that the communication cost when secret sharing a number between n parties is exactly $n-1$ (one random number send to each of the other $n-1$ parties).

The two homomorphic schemes described in this section are useful tools in computing simple functions over distributed data. As a simple example, suppose we want to compute the sum of n numbers $a_1, \ldots, a_n$, where each number is a secret known to only one party. With homomorphic encryption, the first party can compute $(pk, sk) = G(r)$, and send pk to all the other $n-1$ parties. He then sends $E(pk, a_1, r_1)$ to the second party, who computes $E(pk, a_1 + a_2, r_2) = E(pk, a_1, r_1)E(pk, a_2, r_2')$. The second party forwards this ciphertext to the third party, and so on, until the last party sends $E(pk, \sum_i a_i, r_n)$ back to the first party, who can then decrypt the result. Since it is computationally infeasible for any party, except for the first, to compute the intermediate results, no one will be able to learn anything else than the final sum. A similar procedure can be implemented with homomorphic secret sharing.

4. OUR ALGORITHM

Our privacy-preserving clustering algorithm is an improvement of the one proposed by Clifton and Vaidya in [12]. The central difference between our algorithm and the algorithm of [12] is the search for the cluster which is closest to a given entity. In [12] homomorphic encryption is used to do a secure computation of the closest cluster, whereas we use secret sharing. The power of secret sharing in this setting is that communication overheads are considerably lower. To make our presentation clear, we simplify some parts of the algorithms, but apply the same simplifications to [12] when we compare the efficiency of the two algorithms in Sec. 7.

Our algorithm performs distributed k-means clustering with r parties. The data is vertically partitioned such that each of the r parties has some of the attributes of the dataset. The number of entities in the dataset is n. The goal of the r parties is to perform k-means clustering on their *aggregated* data without revealing the values of the attributes they own to the other parties. The algorithm will divide the entries into k clusters and each party learns the cluster means corresponding to their own attributes, and the index of the cluster into which each entity is assigned. Ideally, no party should learn anything else than this.

Let μ_c, $c \in \{1, ..., k\}$, represent the cluster means of the result. Let μ_{ci} be the projection of cluster mean c onto the attributes of party i ($\mu_c = (\mu_{c1}, \ldots, \mu_{cr})$). As output of privacy preserving k-means clustering party i gets:

- The final mean μ_{ci} for each cluster $c \in \{1, \ldots, k\}$.
- The cluster index for each entity $j \in \{1, ..., n\}$.

Privacy preserving k-means clustering over vertically partitioned data is studied by Clifton and Vaidya in [12]. In our work we follow the same approach as Clifton and Vaidya. However, Clifton and Vaidya make use of additive homomorphic encryption which requires public key encryptions. The public key encryptions are the bottleneck of the method described in [12]. In contrast, we use additive secret sharing to achieve privacy, which gives us lower computation and communication overhead than [12].

Like the work of Clifton and Vaidya, our algorithm follows the standard k-means clustering method. Firstly, initial cluster means are selected, and all entities in the dataset are assigned to the closest initial clusters. After the initial assignment of clusters, the cluster means are recalculated and each entity is reassigned to the cluster with the closest cluster mean. The process continues until a termination criterion is met. Algorithm 1 shows the pseudo code of the privacy preserving k-means clustering algorithm, which is common to our work and [12].

Algorithm 1 Privacy Preserving k-means algorithm

```
do in parallel for each party $i \in \{1, \ldots, r\}$
    for each cluster $c \leftarrow 1, \ldots, k$ do
        initialize $\mu_{ci}$ randomly
    end for
end parallel
repeat
    for each entity $j \leftarrow 1, \ldots, n$ do
        Cluster[j] ← SecurelyComputeClosestCluster(j)
    end for
    do in parallel for each party $i \in \{1, \ldots, r\}$
        for each cluster $c \leftarrow 1, \ldots, k$ do
            $\mu_{ci} \leftarrow$ mean of party $i$'s attributes in cluster $c$
        end for
    end parallel
until termination criteria met
```

The algorithm that we present in this paper terminates when there is no change in the assignment of clusters. The algorithm presented in [12] terminates when the change in cluster means between two iterations is less than a given threshold. Our termination criterion corresponds to setting the threshold to 0. To give a fair comparison with the original algorithm presented in [12], we have used a threshold of 0 in our implementation of their algorithm.

The final clustering result depends on how we choose the initial cluster means. It is, however, a standard approach to choose the initial cluster means randomly, which is the case in [12] as well. Another factor to be considered is the distance measure used. For simplicity we use Euclidean distance. Whenever we compare our work to the work of [12], we use Euclidean distance in their algorithm as well.

Since the data is vertically partitioned, each party can compute part of the distances between each of the n entities in the dataset and the cluster means. Since we use Euclidean distance the square of the total distance between an entity and a cluster mean is the sum of the squares of the subdistances computed at the subspaces of each party:

$$\|x_i - \mu_c\|^2 = \sum_{p=1}^{r} \|x_{ip} - \mu_{cp}\|^2. \quad (1)$$

However, the parties cannot reveal their sub-distances in order to compute the sum of them, since the local sub-distances may contain private information. We therefore need to compute and compare the distances securely without revealing the individual sub-distances. This is done in the "SecurelyComputeClosestCluster" algorithm, which we will describe in the next subsection. It is in the secure computation of closest clusters that our algorithm varies from that of [12].

4.1 Secure Closest Cluster Computation

To compute the closest cluster mean for a given entity in the database we have to securely sum the sub-distances computed by each party and compare the results so that nothing other than the comparison result is learned. With n entities, the algorithm has to be invoked nt times, where t is the number of iterations in the standard k-means algorithm. In this section, we describe the algorithm for securely finding the cluster mean which is closest to a given entity.

Similar to [12], the security of our algorithm relies on three ideas:

- Each party sends secret shares its sub-distance to all the other parties, and the sum of the sub-distances is computed on the secret shares (where [12] adds random blinding values in a protocol which applies encryption).
- The comparison of distances is performed on secret shares so that only the comparison result is learned. The actual values of the distances are not learned.
- The ordering of clusters is permuted so that for each entity in the database, only the index of the closest cluster is learned. Relative orderings of the entity's distances to each cluster mean μ_c cannot be learned. This is very simple when we work with secret shares, but in [12] this is the step that requires the highest amount of communication and computation since they rely heavily on public key encryptions.

The most important difference between our work and the work of Clifton and Vaidya is the secure computation of the closest cluster. In [12] the first party selects "disguising values" for each pair of cluster and party such that the sum of all disguising values corresponding to one cluster is zero. Then, the first party together with all other parties compute the encryption of the sub-distances plus the corresponding disguising value. Afterwards, the encrypted distances are permuted by the second party. Finally, the first party decrypts the distances, finds the minimal distance, and reveals the identity of the closest cluster with the help from party 2. The closest cluster corresponds to the new cluster assignment of the given entity. In contrast to Clifton and Vaidya, we use additive secret sharing, which allows us to compute all distances by locally adding up the correct shares. Our algorithm for securely computing the closest cluster mean for each entity has three phases. Pseudo code of these three phases are in Algorithm 2.

Phase 1: In the first phase of the secure closest cluster computation algorithm each party secret shares its sub-distance

Algorithm 2 Secure Closest Cluster Computation

```
Require: entity $e$, cluster means $\mu_1, \dots, \mu_k$
Ensure: Closest cluster to $e$
  do in parallel for each party $i \in \{1, \dots, r\}$
    for each cluster $c \leftarrow 1, \dots, k$ do
      Phase 1 :
      $X_{ic} \leftarrow$ local component of the distance from $e$ to cluster mean $\mu_c$
      for every other party $j$ do
        $\alpha_{ij} \leftarrow$ random number
        send $\alpha_{ij}$ to party $j$
      end for
      $\alpha_{ii} \leftarrow X_{ic} - \sum_{j \neq i} \alpha_{ij}$
      Phase 2 :
      $T_{ic} = \sum_j \alpha_{ji}$
      send $T_{ic}$ to party $r$ (Party 1 does not send anything!)
    end for
  end parallel
  Phase 3 : (Phase 3 only involves parties $1, 2, 3$ and $r$)
  do in parallel for parties $i = 1, r$
    for each cluster $c \leftarrow 1, \dots, k$ do
      Party 1: $D_{1c} \leftarrow T_{1c}$
      Party r: $D_{rc} \leftarrow \sum_{i=2}^{r} T_{ic}$
    end for
    $(D'_{i1}, \dots, D'_{ik}) \leftarrow$SecurePermute$(D_{i1}, \dots, D_{ik})$
  end parallel
  return SecureFindMinimum$(D'_{i1}, \dots, D'_{ik})$
```

for each cluster mean with every other party. Let X_{ic} be the ith sub-distance between the entity that is being evaluated and the cluster mean c. Each party (i) creates a random number α_{ij}^c for every other party, and sends α_{ij}^c to party j for all $c \in \{1, \dots, k\}$. The ith party keeps $\alpha_{ii}^c = X_{ic} - \sum_{j \neq i} \alpha_{ij}^c$ to himself. Note that X_{ic} cannot be computed unless all parties come together to recover it.

Phase 2: After the completion of Phase 1, for every cluster c, we let T_{ic} denote party i's secret share of the distance between the given entity and cluster mean c ($T_{ic} = \sum_{j=1}^{r} \alpha_{ji}^c$).

Since all α_{ij}^c, $i \neq j$, are random numbers, T_{ic} is also a random number, so nothing can be learned from it. Every party, apart from the first party, now sends T_{ic} to the rth party.

Phase 3: This phase only involves parties $1, 2, 3$ and r. Party r adds the values T_{ic} to compute $D_{rc} = \sum_{i=2}^{r} T_{ic}$. Party one defines $D_{1c} = T_{1c}$. The distance between the given entity and cluster mean c is now $D_c = D_{1c} + D_{rc}$. Since party 1 did not send T_{1c}, party r cannot learn the distance. The task of party 1 and r is now to find the minimum element from the list $(D_1, \dots, D_k)$, where party 1 knows D_{1c}, and party r knows D_{rc} of each element. This is done by comparing each of the elements with the current smallest element one by one. The comparison presents two problems:

1. How does parties 1 and r compare two numbers $D_c = D_{1c} + D_{rc}$ and $D_{c'} = D_{1c'} + D_{rc'}$ such that neither of them will learn the values of D_c and $D_{c'}$?
2. How do we prevent parties 1 and r from learning the

ordering of all the distances? They should only learn the minimum distance?

The first problem is solved by observing that $D_c < D_{c'}$ if and only if $D_{1c} - D_{1c'} < D_{rc'} - D_{rc}$. Party 1 knows the left hand side, and party r knows the right hand side. They can now compute the result of the comparison by applying the so called "Yao's millionaires problem". We will describe this in detail in Sec. 4.3.

To solve the second problem above, parties 1 and r permute the order of the elements in the vector $(D_1, \ldots, D_k)$ with the help of party 2 and party 3. We discuss this further in Sec. 4.2 below.

Once Phase 3 is completed parties 1 and r can reveal the index of the cluster which is closest to the entity being considered.

4.2 Secure Permutation

Since parties 1 and r obtain the result of all the comparisons of the distances from a given entity to all the cluster-means they will not only know which cluster is closer to the given entity, but also know which cluster is furthest away, second furthest away and so forth. This clearly gives parties 1 and r an advantage over the other parties. To prevent parties 1 and r from obtaining this information, we first make a "secure permutation" of all the cluster distances. After finding the minimum element in the permuted list of distances parties 1 and r are only told the true identity if the cluster at minimum distance.

The secure permutation is the bottleneck for the protocol by Clifton and Vaidya since it amounts to more than 90% if the computation time. The reason being that in their protocol party 1 has to create kr public-key encryptions for each entity in the database.

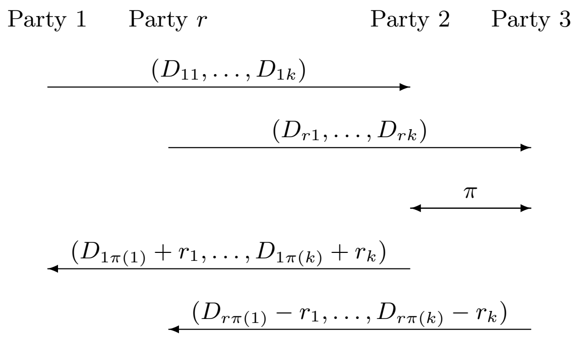
Party 1
Party r
Party 2
Party 3
$(D_{11}, \ldots, D_{1k})$
$(D_{r1}, \ldots, D_{rk})$
π
$(D_{1\pi(1)} + r_1, \ldots, D_{1\pi(k)} + r_k)$
$(D_{r\pi(1)} - r_1, \ldots, D_{r\pi(k)} - r_k)$

Figure 1: Secure Permutation Protocol

In our protocol we take advantage of secret sharing to make a considerably more efficient permutation protocol. Parties 1 and r need the help of two other parties, parties 2 and 3, for the permutation. Party 1 simply sends his secret shares to party 2, and party r sends his secret shares to party 3. Then parties 2 and 3 agree on a permutation and each of them apply the permutation to the vector of shares they received. To make sure that parties 1 and r cannot recognize the elements in the vector they get back, party 2 adds a random number r_i to element i, while party 3 subtracts r_i (recall that each element of the vectors are additive secret shares of a distance, so adding and subtracting the same random number will not change the result of the protocol). After applying the permutations, they send the vector back to the parties they got them from. The protocol can be seen in Fig. 1.

Since we are assuming the presence of confidential and authentic channels, parties 2 and 3 can agree on the permutations in any way they want.

4.3 Secure Minimum Element

In the last step of the "Secure Closest Cluster Computation" parties 1 and r compare the distances between the current entity and all the cluster means. Each of them has a secret share of the k entries in the permuted vector of these distances. To find the minimum, they perform $k - 1$ comparisons with the current minimum element (they do not compare the first distance). After finding the minimal element, party 1 informs party 2 of the permuted identity of the closest cluster. Since party 2 knows the permutation, she can announce the real identity of the closest cluster to all parties.

Clifton and Vaidya[12] suggest using Yao's circuit evaluation for the comparisons. They argue that even though Yao's protocol is very inefficient, it may be plausible to use it for the comparison, since they only perform kn comparisons in each iteration of the k-means algorithm. Nonetheless, in our experiments, we use a protocol proposed by Savaş, Pedersen, and Kaya in [7] in both our protocol and our implementation of Clifton and Vaidya's protocol.

In the comparison protocol of [6, 7], two players who wish to compare two integers, each create two secret shares of their own inputs and send these shares to two semi-honest non-colluding third parties. The protocol utilizes the fact that additive secret sharing is homomorphic with respect to addition, thus bitwise additive secret sharing is homomorphic with respect to bitwise addition, XOR operation of bits. The protocol of [7], like Fischlin's protocol[4], uses a trick by Sander, Yung, Young [11] to convert XOR homomorphic secret sharing into AND homomorphic secret sharing without disclosing the secret. For the proofs of correctness and security, the readers are referred to [7].

We have modified the protocol from [7] slightly, by observing that the two semi-honest third parties are actually not needed in the protocol. In the original protocol, the parties involved in secure comparison secret-share their inputs with 2 semi-honest third parties, sending one share to a third party and the remaining share to the other third party. This is equivalent to each party secret-sharing their inputs among themselves; each party sends a secret share of its input to the other party while keeping the remaining secret share to itself. Therefore, in our protocol, the parties who are involved in the secure comparison (parties 1 and r) apply the protocol in [7] by secret-sharing their inputs among themselves. In the original protocol[7], two third parties end up with secret shares of a binary vector. The vector is as long as the inputs that are compared. If the first input to the comparison is greater than the other, then the vector has exactly one 1-bit at the first position where the first input is greater than the second input. Since the position of the 1-bit gives information about the relative difference between these two inputs, the two third parties agree on a permutation and permute the vector before it is sent to the party who will learn the result of the comparison. In our protocol, parties 1 and r do not perform the permutation themselves since they are the ones that will learn the result of the comparison. Instead, we use the permutation algorithm described above in Sec. 4.2.

5. PRIVACY DISCUSSION

Before going into the discussion about privacy, we have to define what we mean by private information in the algorithm. Above we set as our goal that the only information party i will learn after the algorithm is:

- The final mean μ_{ci} for each cluster $c \in \{1, \dots, k\}$.
- The cluster index for each entity $j \in \{1, ..., n\}$.

No other information other than these two should be learned out of algorithm. The actual values of data attributes belonging to the data holders are obviously private information. Since these values are not shared among data holders during the execution of our algorithm, we may say these values are kept private. Portions of distances calculated by each party according to their set of attributes are also deemed private information since one may recover the actual values of the entities by knowing the distances of the entities to the cluster means.

5.1 Privacy in our Protocol

In order to examine how our method protects privacy, we focus on the "Securely Computing the Closest Cluster" algorithm which is the only part of the algorithm in which there are interactions between the parties. In Phase 1 of the algorithm, all parties secret-share their local distance values with the other parties. Since each party keeps one share for himself and since *all* shares are needed to recover the local distances, no information will leak even if all the remaining $r-1$ parties collude.

In Phase 2 of the algorithm all parties send their T_{ij} values to party r. Since T_{ij} are secret-shares of the total distance (it is the sum of the secret-shares of the local distances) and since party 1 does not send T_{1j} to party r, parties 1 and r cannot gain any information unless they collude. Notice that T_{1j} contains shares from all other parties' local distance components.

Phase 3 of the algorithm consists of the permutation and secure comparison sub-phases. In the permutation phase parties 1 and r send their secret shares of the distance to parties 2 and 3 respectively. Now parties 2 and 3 are in the same situation as parties 1 and r were after Phase 2. Collusion between parties 2 and 3 allow them to learn an entity's distance to each cluster mean, therefore the permutation phase is privacy-preserving under the assumption that parties 2 and 3 are non-colluding. The secure comparison is privacy-preserving under the assumption that parties 1 and r are non-colluding; detailed proof is explained in [7].

Finally the "Securely Computing the Closest Cluster" algorithm returns the closest cluster to the given entity. Clearly this is exactly what the parties are supposed to learn in the last round. However, all parties will see how entities change cluster in each iteration of the k-means clustering algorithm. An entity, which fluctuates between two clusters, can be assumed to be approximately half-way between the two cluster means. This is more information than is strictly allowed. However, the original algorithm by Clifton and Vaidya[12] suffers from the same problem as well.

5.2 Security Comparison

In both our protocol and the protocol by Clifton and Vaidya [12], privacy breaches may occur when two or more parties collude. Therefore, in both the protocols, some non-collusion assumptions have to be made for some specific parties. However, the gained information in Clifton's and Vaidya's protocol as a result of collusion is much more severe. In the protocol by Clifton and Vaidya, each party, upon computing local distances X_{ic}, sends X_{ic} to party 1. Party 1 adds random values α_{ic} to X_{ic} of each party. After this phase, each party sends $X_{ic} + \alpha_{ic}$ to party r. Here, collusion between parties 1 and r allows them to learn the value of X_{ic} of all other parties, which is no doubt a severe privacy breach. In our protocol such a privacy breach is mitigated by the use of secret sharing. Each party secret-shares X_{ic} with every other party, so the value of X_{ic} can be recovered only if all parties come together. As a solution to this problem, Clifton and Vaidya[12] propose an extension to their protocol that increases the number of colluding parties needed to reveal X_{ic} of each party. In essence, they apply their permutation steps more than once according to a chosen anti-collusion parameter. If this parameter is denoted as p, they repeat the permutation algorithm $p-1$ times by choosing a different party at each time to play the role of party 1. This method increases security of the protocol; however, it also increases computation and communication cost considerably.

In our protocol, we have non-collusion assumptions for parties $1, 2, 3$ and r. If permuter parties (parties 2 or 3) collude, they can reveal each entity's distance to each cluster mean μ_c. Also, if parties $1, r$ and one of the permuter parties(2 or 3) collude, they can reveal each entity's distance to each cluster mean μ_c. We can say that collusion between 2 specific parties(parties 2 and 3) leads to a privacy breach in our protocol at the worst case. We can also do the same trick as in the protocol of [12] and increase the non-collusion threshold by applying the permutation algorithm more than once, at each iteration picking different two parties to play the roles of party 2 and 3. Since our permutation algorithm does not contain any encryption or similar expensive operations, it can be applied more than once without bringing too much computation and communication overhead.

The comparison of the security of the two protocols reveal that collusion between 2 specific parties leads to some privacy breach in both protocols. However, in [12] the leaked information is more severe in case of a collusion. Some information specific to a party (X_{ic} for party i) can be learned in [12], whereas in our protocol leaked information is global information, not bound to a specific party.

6. COST ANALYSIS

A privacy preserving distributed data mining algorithm aiming to be used in real life applications should not bring too much communication and computation overhead. In the following two subsections we analyze the communication and computation overheads of our algorithm. The overheads mainly occur in the "Securely Finding the Closest Cluster" part of the algorithm. Thus, we only analyze this portion of our algorithm. It should be noted that both communication and computation overheads of the k-means clustering algorithm depend on the dataset. The number of iterations required before the termination criteria is met depends on the data and the initial cluster means. Therefore, in the communication and computation cost analysis, we only consider one iteration of the k-means algorithm. We let r be the number of parties, n the number of entities in the database and k the number of clusters.

6.1 Communication Cost Analysis

Most of the communication overhead in our protocol is created in Phase 1 of the "Securely Finding the Closest Cluster" algorithm. In this phase, for each entity, one party sends secret shares of its portion of the distance to every other party for each of the k clusters. This is equal to sending each party a vector of length k. Each entry of this vector is a secret share of 32 bits. Since there are r parties and each of them send a shared secret vector of length k to every other party, the communication cost of this step of our protocol is $32r(r-1)kn$ bits.

In the original work of Clifton and Vaidya[12], every party sends its local distance value *encrypted* to party 1. Party 1 adds random values to these local distance values by using the additive homomorphic property of the public key encryption scheme used, and sends the distorted values back to each data holder. Assuming that a public key encryption scheme with 1024 bits of key and block size is used, which is the minimum for security purposes, the communication cost of this phase in the work of Clifton and Vaidya is $2(r-1)1024kn = 2048(r-1)kn$ bits.

From the above analysis we observe that for values of r up to $2048/32 = 64$, our algorithm has smaller communication cost in phase 1 as compared to the work of Clifton and Vaidya[12].

Phase 2 of our algorithm is very similar to Clifton and Vaidya's protocol in terms of communication cost, each party apart from party 1 will send a 32 bit integer to the r^{th} party, therefore the communication costs here are the same, $32n(r-2)k$ bits.

In Phase 3 of the algorithm we run a protocol for Yao's Millionaires problem which gives us very little communication overhead. The only communication is one call to the permutation protocol in each call to the comparison protocol. The vectors permuted are of length 32λ, where λ is a security parameter given to the comparison protocol. Typical values for λ are around 50. We have applied the same comparison algorithm in our implementation of the protocol by Clifton and Vaidya, so the two protocols have the same computational overhead in this step.

To sum up, the total communication cost of our algorithm is much lower up to a certain number of parties (up to 64 parties when using 1024-bit encryption in [12]). We confirm this bound in our experiments in Section 7.

6.2 Computation Cost Analysis

In terms of computation overhead, our protocol is more efficient than the protocol of Clifton and Vaidya. Since we use secret sharing rather than encryption, our protocol uses primitive operations only. However, the algorithm of Clifton and Vaidya uses public key encryptions which requires expensive modular exponentiation operations on very large numbers. In Phase 1 of Clifton and Vaidya's protocol two public key encryptions and one public key decryption are needed for every entity and cluster pair. In total, $3nk$ modular exponentiations are needed in Clifton and Vaidya's protocol. Given the primitive nature of the operations in our protocol, our protocol gives much lower computation overhead, which is confirmed by our experiments in Section 7.

7. EXPERIMENTAL RESULTS

We implemented our protocol and the protocol of Clifton and Vaidya[12] and performed simulations on them in order to validate the theoretical findings for both communication and computation costs given in Section 6. For testing purposes in our simulations we used two datasets. The first dataset is a synthetic control chart time series data taken from the UCI Machine Learning repository. The dataset consists of 600 items, all of them with 60 attributes. The second dataset is a spatio-temporal dataset consisting of trajectories of school buses. The number of measurement points for each trajectory is over 1050. Since each measurement consists of one x and one y coordinate, each item in the dataset has over 2100 attributes. For both of the datasets, the attributes are partitioned among the parties evenly; that is, every party has equal or near equal number of attributes. However, the distribution of attributes does not affect neither of the two algorithms considerably (since the first step of each iteration is to compute local distances).

It is important to note that the initial cluster assignments of entities greatly affects the execution time of the k-means clustering algorithm. In order to make a fair comparison, we make sure that initial cluster assignments of the entities in the datasets are the same for each test for both of the protocols.

The tests are done on a 2.0 Ghz Intel Core2Duo mobile PC with 2GB's of RAM. The implementations are done in the C#.NET programming language. In the implementation of the protocol in [12], we use the Paillier[10] public key encryption scheme with 1024 bit cipher texts. Normally Paillier encryption is considered secure when it is used with ciphertexts of 2048 bits. However, in our experiments we use a weaker version with ciphertexts of 1024 bits. This makes the protocol of [12] faster for the purpose of our experiments; but as can be seen from the experiments, even with this weak implementation of Paillier, our scheme performs better.

We have performed two tests with each of the datasets. First test is to see how much communication overhead our protocol brings and how the communication overhead in our protocol compares to communication overhead in the protocol of Clifton and Vaidya[12]. The total amounts of transmissions caused by the protocols with respect to the number of parties are depicted in Figures 2 and 3 for two different datasets. As can be seen from these figures, our protocol has lower communication cost up to a certain threshold for the number of parties. This threshold is expected to be 64 as analyzed in Section 6.1. The simulation results given in Figure 3 show a threshold value around 61. The reason of this minor decrease is that the ciphertexts produced in the protocol of [12] are not always 1024 bits, but sometimes a bit less than that due to the nature of the public key cryptosystem used. Figure 2 shows the communication cost up to 60 parties only. We cannot have more than 60 parties for this dataset since the dataset has 60 attributes.

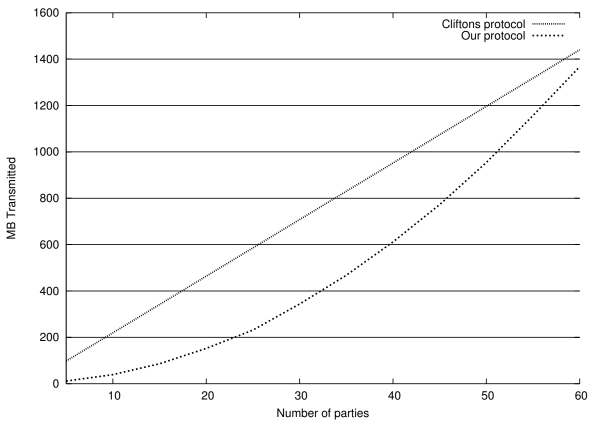

Figure 2: Communication cost with the timeseries dataset.

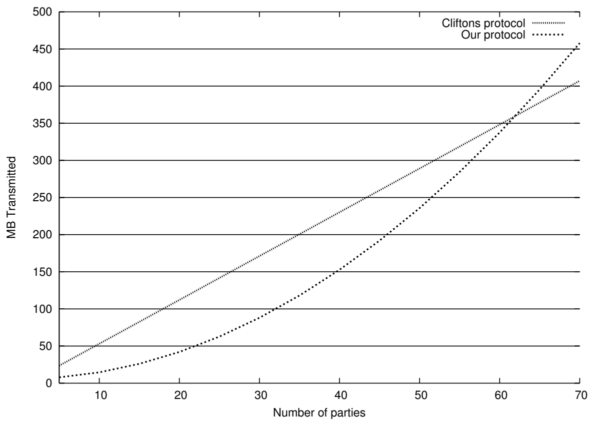

Figure 3: Communication cost with the trajectories dataset.

The second test that we have performed is to analyze and compare the computational overheads brought by our protocol and the protocol of Clifton and Vaidya[12]. Execution times of the protocols with respect to the number of parties are shown in Figures 4 and 5 for the two datasets (the time axis is in log scale). As expected from the analysis described in Section 6.2, our protocol has a smaller computational overhead compared to Clifton and Vaidya[12]. The execution times of our protocol are always less than 30 seconds, whereas the execution times of [12] are thousands of seconds. The reason of this significant performance difference lies in the use of public key cryptography. In order to compare each entity to each cluster mean, Clifton and Vaidya[12] have to perform expensive public key encryptions and decryptions linear in the number of parties. In our protocol, however, we do not rely on such computationally expensive public key operations.

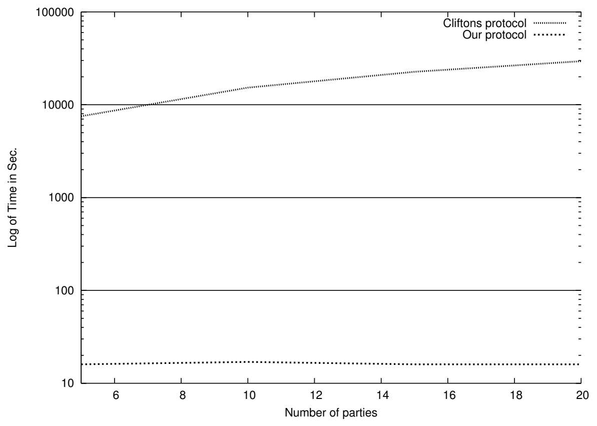

Figure 4: Timing with the timeseries dataset.

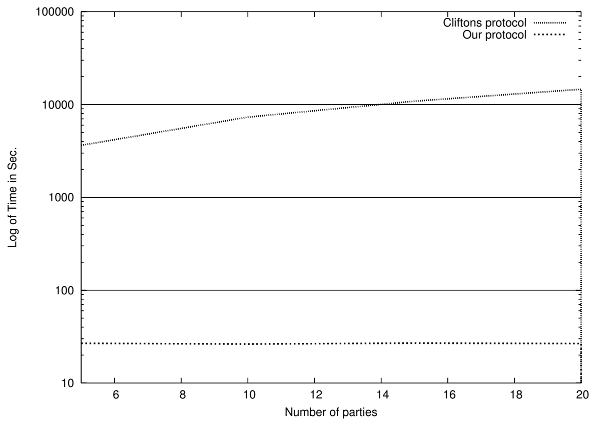

Figure 5: Timing with the trajectories dataset.

8. CONCLUSIONS

In this paper, we proposed a new algorithm for applying k-means clustering to vertically partitioned data without compromising the privacy of individuals. We have taken the work by Clifton and Vaidya[12] as a basis. Clifton and Vaidya use the additive homomorphic property of certain public key encryptions, but we based our protocol on additive secret sharing which is also homomorphic with respect to addition. Due to the fact that we use secret sharing rather than encryption, we do not suffer from the bit expansion drawback of public key encryption schemes which gives us less communication overhead. Since we are not doing any expensive modular exponentiations our protocol also gives us less computation overhead as well.

9. REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 439–450. ACM, 2000.

[2] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1988. ACM.

[3] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Y. Zhu. Tools for privacy preserving distributed data mining. *SIGKDD Explor. Newsl.*, 4(2):28–34, 2002.

[4] Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *Progress in Cryptology - CT-RSA 2001: The Cryptographers' Track at RSA Conference 2001*, volume 2020 of *Lecture Notes in Computer Science*, page 457, 2001.

[5] Murat Kantarcioglu and Chris Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Trans. Knowl. Data Eng.*, 16(9):1026–1037, 2004.

[6] S. V. Kaya, T. B. Pedersen, E. Savaş, and Y. Saygin. Efficient privacy preserving distributed clustering based on secret sharing. In *PAKDD 2007 International Workshops: Emerging Technologies in Knowledge Discovery and Data Mining*, pages 280–291. Springer, 2007.

[7] Selim Volkan Kaya. Toolbox for Privacy Preserving Data Mining. Master's thesis, Sabanci University, Istanbul, TURKEY, July 2007.

[8] Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. Cryptographically private support vector machines. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 618–624. ACM, 2006.

[9] Kun Liu, Hillol Kargupta, and Jessica Ryan. Random projection-based multiplicative data perturbation for privacy preserving distributed data mining. *IEEE Trans. Knowl. Data Eng.*, 18(1):92–106, 2006.

[10] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology — EUROCRYPT '99. International Conference on the Theory and Application of Cryptographic Techniques*, Lecture Notes in Computer Science, pages 223–238. Springer-Verlag, May 1999.

[11] Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for nc1. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 554, Washington, DC, USA, 1999. IEEE Computer Society.

[12] Jaideep Vaidya and Chris Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215, New York, NY, USA, 2003. ACM Press.

[13] Jaikumar Vijayan. House committee chair wants info on cancelled dhs data-mining programs. *Computer World*, September 18 2007.

[14] Rebecca Wright and Zhiqiang Yang. Privacy-preserving bayesian network structure computation on distributed heterogeneous data. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 713–718, New York, NY, USA, 2004. ACM.

[15] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164, 1982.