Efficient Privacy Preserving K-Means Clustering

Maneesh Upmanyu, Anoop M. Namboodiri, Kannan Srinathan, and C.V. Jawahar

International Institute of Information Technology, Hyderabad
{upmanyu@research.,anoop@,srinathan@,jawahar@}iiit.ac.in

Abstract. This paper introduces an efficient privacy-preserving protocol for distributed K-means clustering over an arbitrary partitioned data, shared among N parties. Clustering is one of the fundamental algorithms used in the field of data mining. Advances in data acquisition methodologies have resulted in collection and storage of vast quantities of user's personal data. For mutual benefit, organizations tend to share their data for analytical purposes, thus raising privacy concerns for the users. Over the years, numerous attempts have been made to introduce privacy and security at the expense of massive additional communication costs. The approaches suggested in the literature make use of the cryptographic protocols such as *Secure Multiparty Computation (SMC)* and/or *homomorphic encryption schemes* like Paillier's encryption. Methods using such schemes have proven communication overheads. And in practice are found to be slower by a factor of more than 10^6. In light of the practical limitations posed by privacy using the traditional approaches, we explore a paradigm shift to side-step the expensive protocols of SMC. In this work, we use the paradigm of *secret sharing*, which allows the data to be divided into multiple shares and processed separately at different servers. Using the paradigm of secret sharing, allows us to design a provably-secure, cloud computing based solution which has negligible communication overhead compared to SMC and is hence over a million times faster than similar SMC based protocols.

Keywords: K-Means, Privacy, Security, Secret Sharing.

1 Introduction

K-means clustering [1] [2] is one of the most widely used techniques for statistical data analysis. Researchers use cluster analysis to partition the general population of consumers into market segments and to better understand the relationships between different groups of consumers/potential customers. However the collected data may contain sensitive or private information, thus heightening the privacy concerns [3] [4]. The privacy and secrecy considerations can prohibit the organizations from sharing their sensitive data with each other. The solution should not just be provably secure i.e. it leaks no additional useful information, but should also minimize the additional overheads in terms of communication and computation costs required to introduce privacy. Addressing the problem requires many practical challenges to overcome before a possible wide-scale deployment. Solutions were sketched to extract knowledge by making the participating parties to compute common functions, without having to actually reveal their individual data to any other party [5] [6]. Vaidya *et al.* [7] summarize the state of art methods available for privacy preserving data mining. More detailed reviews of the previous work can be found in Verykios *et al.* [8].

Previous solutions can be primarily categorized as, *i)* those using *Data Perturbation techniques*, and *ii)* those employing *Secure Multiparty Computation (SMC)*. The first category of approaches introduces noise and data transformations to achieve partial privacy [9] [10] [11]. The clustering is then done of the noisy version of the data, resulting in approximately correct clusters [5] [12]. Such approaches compromise privacy for practicality, however the key advantage is the negligible communication overhead needed by such approaches.

The second category of approaches aims to achieve complete privacy. This is done using the well known cryptographic protocol of SMC [13]. SMC facilitates a group of people, each with its own private data, to perform some common computation task on the aggregate of their data. SMC ensures that, in the process, no personal information of data is revealed to any one [14]. However, the SMC based protocols are found to be extremely computationally expensive [13]. In other words, an operation which requires a single round of communication in a non-secure implementation, would require hundreds of thousands of rounds of communication (depending on the domain size) to achieve the same operation in a secure implementation using SMC. For data mining applications, the sheer volume of the data involved makes the protocol infeasible in terms of the communication cost. For example, Vaidya *et al.* [15], İnan *et al.* [16] and Wright *et al.* [17] use SMC as a subroutine to propose privacy preserving clustering. However, the huge computational costs makes these solution of limted practical interest.

Another set of proposed approaches uses the semantically secure additive or multiplicative homomorphic encryption schemes [18] [19]. In such a protocol, one party encrypts its data using its public key, and share the encrypted data with the other party for computation. Interactive protocols are then designed to carry out the clustering algorithm [20] [10]. The overheads of encryption and the communication costs needed to carry out clustering limits the scope of such algorithms. Interaction can be reduced with the usage of a doubly homomorphic scheme [21]. However, the only known doubly homomorphic scheme is the one recently proposed by Craig Gentry [22] and would most likely lead to a computationally intensive theoretical solution.

In this work, we achieve the security at the level of SMC while keeping the communication costs to a level similar to that of the first category. We achieve this using the paradigm of the *Secret Sharing*[23] [24] over a mesh of processing servers. Our solution is first of its type, and is both efficient and mathematically simple. In the process we also side-step the communication bottlenecks posed by the usage of SMC and asymmetric encryption schemes. Our proposed solution is not only computationally efficient but also secure independent of whether or not $P \neq NP$. We however do assume the servers to be non-colluding and having the ability to generate random numbers.

We address the scenario of N parties, sharing an arbitrary partitioned data [17], wishing to privately collaborate for doing cluster analysis on their aggregated data. In our setting, the attribute names form the public information. Each of the entity is either completely owned by one of the users, or the attributes are shared among the N users, where the share of some users can also be ϕ. If a record is 'completely' owned by anyone, then its existence remains hidden from other users. If any of the attributes for an

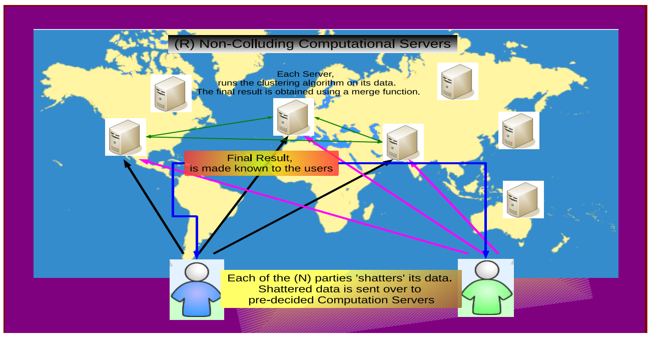
(R) Non-Colluding Computational Servers
Each Server,
runs the clustering algorithm on its data.
The final result is obtained using a merge function.
Final Result,
is made known to the users
Each of the (N) parties 'shatters' its data.
Shattered data is sent over to
pre-decided Computation Servers

Fig. 1. Sample Mesh of servers. Each of the N users shatters their private data (Sec: 2) and sends over the shares to the pre-selected R servers for computation. The final result is obtained by merging (Sec: 2) the outputs of the computational servers. In above example, N is 2 and R is 3.

entity are with more than one user, then a weighted average of the attribute values is considered for the computational purpose. Entities are indexed using a mutually agreed upon indexing scheme. The indexing scheme addresses the two concerns of *i)* hiding the entity's identity from the servers, and *ii)* a common index for accessing the vertically partitioned data. We now look at the architecture of our proposed solution.

We propose a 'cloud computing' based solution that utilizes the services of R, $(R > 2)$, non-colluding servers. Each of the N users, is required to compute the R secret shares of its private data using a *shatter function* (see the algorithm, defined in Sec: 2). Each share is then sent over to a specific server for processing. Note that the shatter function ensures that the computed secret shares on its own reveal no information about the original private data. The cloud of employed servers, now runs the K-means algorithm using just the secret shares. The protocol ensures that none of the users/servers have sufficient information to reconstruct the original data, thus ensuring privacy. As shown, later in the paper, the shatter function that we choose allows efficient computations using just the shares. That is, unlike SMC, the number of rounds of communication to implement an operation on secret shares is equivalent to that required in a non-secure implementation of the same operation. The advantage of this is that it significantly reduces the communication costs over the similar SMC based protocols, thus making privacy preserving clustering practical. Figure 1 shows a pictorial description of the proposed architecture, while the algorithm is discussed in detail in Sec: 3.

2 The Building Blocks of Security

We use the paradigm of Secret Sharing (SS) to achieve privacy and efficiency. Secret Sharing (SS) [25] [26] [23] refers to the methods for distributing a secret among a group of servers, each of which is allocated a share of the secret. The secret can be reconstructed only when the shares are combined together; on their own, they have no meaningful information. In our problem setting, we ask each of the collaborating users to compute the secret shares of their private data, and send them over to the processing servers. The processing servers then privately collaborate (without reconstructing the actual data) to run the K-means algorithm over the secret shares. *Note that*, not all SS methods allows computation on the secret shares. In order to achieve this, we adopt the Chinese Remainder Theorem (CRT) based secret sharing schemes [23] [27].

However, in the SS schemes of Asmuth *et al.* [23], and Goldreich *et al.* [27], the size (the number of bits) to represent each share is greater than the size of the original data. In other words, for R servers, using these schemes results in a minimum of R fold storage increase. Data expansion is important since it results in cost overheads in terms of storage and interaction among the servers. It becomes even more critical for applications such as data mining that deals with voluminous data.

Understanding the similar limitations, Upmanyu *et al.* [24] recently proposed an efficient method to do privacy preserving surveillance on videos (voluminous data). In this work, we extend their method and propose secure protocols to privately carry our collaborative clustering. The data to be clustered using K-means can be thought of as points in a D dimensional Cartesian space. The data is bounded, i.e. it has a fixed range, and its scale invariant, i.e. even if we scale the axis, the cluster assignment will still be the same. These two are the required desirable properties of the data, that are sufficient for one to adopt the secret sharing scheme as proposed by Upmanyu *et al.* in [24]. We therefore, adopt their *Shatter* (to compute the secret shares) and *Merge* (to reconstruct the secret) functions for the Cartesian data and design a communication and computationally efficient solution to achieve privacy preserving K-means clustering.

Our proposed solution can be summarized as a three step protocol, 1) each user computes the secret shares of his private data, 2) shares are then sent over to a cloud of servers and clustering is privately carried out over the shares, and 3) the users reconstructs the cluster assignment and the cluster centers using the *Merge* function. Before we jump into describing the K-means protocol in Sec: 3, for the sake of completeness we briefly describe the *Shatter* and *Merge* functions as defined in [24]. We also provide an outline of the analysis of the computational and communication overheads and the privacy achieved in each of the sub-steps of the protocol. For those interested, the detailed analysis can be done in a manner similar to in [24]. The *Shatter* and *Merge* functions as defined in [24] are as follows:

Shatter Function $\phi(x)$ **-** *Compute and store the secret shares of the private data :* is defined as the one that splits the data x into R parts, $x_1, x_2, ..., x_R$, such that each share, x_i, by itself does not reveal any information about x. The participating users pre-decide a set of R primes $P_1, \cdots, P_R$ and a scale factor S. The *Shatter function* is defined as:

$$x_i = \phi(x, P_i) = (x \cdot S + \eta) \bmod P_i, \tag{1}$$

where x_i is i^{th} secret share, and η is an independent random number for each secret x, such that $0 \leq \eta \leq S/2$. The secret share x_i is stored with the i^{th} server and on its own gives little meaningful information of x.

In our scenario, each user can shatter his data (each attribute of a record is shattered independently, η is random for each attribute) and sends over the shares to the specific servers for storage. The size of each share is given by $log(P_i)$ per attribute.

Merge Function $\mu()$ **-** *Reconstruct the secret :* given, $x_i = \phi(x, P_i)$ for different prime P_is, the secret x can be recovered using *CRT* [28] by solving a system of congruence. The *merge function* $\mu()$ is defined as:

$$x = \mu(x_i, P_i) = \frac{CRT(x_i, P_i)}{S} \tag{2}$$

CRT recovers $(x \cdot S \; + \; \eta)$, which is appropriately scaled down (integer division by the scale factor) to get the actual value of x. Note that η, which was randomly chosen for each attribute value is not used for recovering the secret. The CRT hence forms our recovery transformation $\mu()$. In our scenario, $\mu()$ is used for reconstructing the cluster centers as computed by the clustering algorithm.

3 The Proposed Algorithm

Following notations are used for describing the protocol. Let L be the number of entities, each made up of D attributes. K be the number of clusters required, and $\boldsymbol{C}_i$, $1 \leq i \leq K$, denotes the cluster locations. The data is arbitrary partitioned among N users. R $(R > 2)$ is the number of computation servers employed. Each server is associated with a unique prime P_i, therefore the number of primes is also R. Each entity is represented in a D dimensional space. The common distance metrics; such a Euclidean, Manhattan or Minkowski; are used for finding the distances. To explain the algorithm we will consider a Euclidean space. As the final output of the privacy-preserving K-means (PPKM) algorithm, each user learns the cluster assignment of the entities owned by them, i.e. which of their entities belong to each clusters. If agreed upon, the location of the K-clusters is also revealed to the users.

The complete protocol can be divided into two phases. The *first phase* deals with *i)* choosing the appropriate primes and the scale factor, *ii)* shattering the data, and *iii)* secure aggregation of the data at the servers. The *second phase* of the protocol deals with the clustering algorithm on the aggregate of the shattered data available with the R computational servers. The basic algorithm follows directly from the standard K-means algorithm [29], which consists of three steps, *i)* Initialization, *ii)* Lloyd Step, and *iii)* Stopping Criterion. The complete protocol is as follows:

3.1 Phase One: Secure Storage

The first step is the selection of an appropriate residue number system (RNS) [24] for secure storage. We extend the *analytical method* [24] to compute the parameters required for ensuring the security and privacy in our problem setting. For a value of R we select $P_1, \cdots, P_R$, such that their product, P, is larger than any intermediate value we have to represent in our algorithm. This range can be easily computed from the range of values we expect in the computations. Scaling the axis and translating the origin of an Euclidean space does not change the final cluster assignment. Hence we represent negative numbers with an implicit sign [30], i.e. $-x \equiv 2M - x$. Floating point data is taken care of by appropriately scaling the dataset to retain a certain decimal precision.

Let $[-U, U]$ be the range of numbers we expect in the computations on secret shares. We choose P_j's such that $P = \prod_{j=1}^{R} P_j \geq 2U$. Typically, one could just choose the smallest of the R consecutive primes satisfying the above property. For complete obfuscation of the data, the scaling factor chosen should be higher than the largest prime [24]. We now analytically choose the optimal set of parameters for our problem setting.

Parameter Selection: Let $[-M, M]$ be the attributes domain. Then the points can be represented in a $D - dimensional\ Euclidean\ space$, $\mathbb{R}^D_{2M}$. Let W_1 be the square of the maximum possible Euclidean distance between two points, i.e. the distance between the two extreme points, thus we get $W_1 = 4M^2D$. Also let W_2 be the maximum sum of the coordinates we can get for a cluster (needed for computing the cluster's mean). This is easily computable as $W_2 = 2ML$ (entire database belong to a single cluster). Let W be the upper range of number we expect in K-means, therefore we have $W = max(W_1, W_2)$. Let us now assume, S to be the required scale factor to get complete privacy. The input data is scaled using this factor. This can be viewed as scaling the axis of the Euclidean space by S, i.e. a point x in the old coordinate system is mapped to $S \cdot x$ in the new scaled space. Therefore, we get $U = max(W_1 \cdot S^2, W_2 \cdot S)$. The primes now need to be chosen such that:

$$S \geq \max_j P_j, \text{ and } P \geq 2U. \tag{3}$$

Simplifying the above, we find that if:

$$S \approx (2W)^{\frac{1}{R-2}} \tag{4}$$

then the individual servers will have little meaningful information [24].

Each of the N-parties uses the *shatter function* (Eqn: 1), to compute secret shares of their respective data. The shares are then sent over to the servers for processing. Note that we make no assumptions on how the attributes of various data points are partitioned among the N-parties. If $\mathcal{D}$ is the (virtual) database arbitrarily shared among the N parties. Each server j basically then stores the shatter of $\mathcal{D}$ w.r.t. P_j.

Privacy: Each server stores only the shattered share of the data. As long as the servers do not collude, little meaningful information of the entities is learned by any of the servers. This follows directly from the security of the shattering scheme [24]. In this entire phase the only information learned is of how the data is actually being partitioned among the users, i.e., for each entity which all attributes are being held by which user. However we note that, in practice this information gain is not significant, and known a prior [15]. The indexing scheme employed ensures that the identity of the entity remains unknown to the servers.

3.2 Phase Two: Secure K-Means

At the end of the phase one, each computation server stores the secret shares (w.r.t. prime P_j) of the database $\mathcal{D}$. Since the scaling factor S was kept positive, the distance comparison in the original space will be equivalent to distance comparison in the new scaled space. Thus, the cluster assignment of the entities in the scaled space would be identical to what we would have expected in the original space. The final cluster locations are obtained from the cluster centers that are learned in the transformed space after appropriately scaling down and removing the introduced randomness.

Our algorithm will follow the same iterative structure as that of the standard K-means algorithm [29]. The objective is to cluster the data (available as secret shares), without leaking any information to any of the servers. RNS being doubly homomorphic, the operations of addition and multiplication can be independently carried out at each server. However division and comparison (both used in K-means) are difficult to do privately in the RNS. We overcome these difficulties by designing communicationlly efficient, privacy preserving protocols for them over one round of communication.

We now give a step by step description of the protocol used for phase two. *Note here*, that the N users are oblivious of algorithm and the data involved in phase two. The contribution of this paper is not to improve upon the K-means algorithm as such but to propose an efficient protocol to privately carry out the clustering.

Step one: Initialization Let $\boldsymbol{C}_1$, $\boldsymbol{C}_2$, $\cdots$, $\boldsymbol{C}_K$ be the K cluster centers, where each C_k is a D dimensional vector. The clusters are initialized as the K entities from the database $\mathcal{D}$ chosen in a pseudo-random fashion. Since, we want to keep the actual cluster locations also private, we thus store only their secret share components. i.e. for a cluster location $\boldsymbol{C}_k$, $1 \leq k \leq K$, the computational server j, $1 \leq j \leq R$, stores the vector $\boldsymbol{C}_{kj}$, where, $\boldsymbol{C}_{kj}$ is the secret share of $\boldsymbol{C}_k$ w.r.t. P_j.

The servers commonly choose the indices of K entities as the initial cluster centers. The secret shares of the chosen K entities, present with the servers, are used as the secret shares of the initial cluster centers C_k. That is, at server j, C_{kj} initialized to the secret share of the chosen entity. The pseudo-code of the algorithm is given in Algo: 1.

Privacy: Servers do not learn any additional information of the data. The initialization is done, directly using the secret shares. This is done independently at each server, thus resulting in zero computation and communication overheads over TTP.

Algorithm 1. PPKM: Initialization

```
1: for each cluster, $k$ = 1 to K do
2:     Choose a random entity index $l$, $l \leq L$
3:     We want to initialize $\boldsymbol{C}_k = \boldsymbol{X}_l$, where $\boldsymbol{X}_l$ be the $D$ dimensional vector of entity $l$.
4:     for each server, $j$ = 1 to R do
5:         Let $\boldsymbol{X}_{lj}$ be the data corresponding to entity $l$ available with the server. We know $\boldsymbol{X}_{lj}$
           is shatter of $\boldsymbol{X}_l$ with mod $P_j$, and was stored with the server during phase one.
6:         Initialize, $\boldsymbol{C}_{kj}$ to $\boldsymbol{X}_{lj}$, where $C_{kj}$ is the shatter share of $C_k$ with mod $P_j$.
7:     end for
8: end for
```

Step two: Lloyd Step In an attempt to minimize the objective function, each iteration reclassifies and recomputes the new cluster locations. The algorithm terminates when it detects *'no change'* (defined by the termination criterion) in the cluster locations. Every iteration can be represented as a sequence of three steps as described below.

i) Finding Closest Cluster Centers: As stated before, since the scaling factor was set to a positive number, finding the closest point is equivalent to finding the one with the minimum of the distances squared in the scaled space. Thus, for every data entity $\boldsymbol{X}_l$, $1 \leq l \leq L$, we find the square of the Euclidean distance to each of the cluster centers $\boldsymbol{C}_k$. The distance square between two D dimensional vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$, is defined as

$$\sum_{d=1}^{D}(X_d^2 + Y_d^2 + 2.X_d.Y_d) \tag{5}$$

which is a set of additions and multiplications. Now, RNS being doubly homomorphic, the above equation can be directly computed using the secret shares. Hence, every server can independently compute the respective secret shares of the distances between the L data points and the K cluster locations. For every data point $\boldsymbol{X}_l$, let $\boldsymbol{T}_l$ be the K length vector, whose share T_{lk} denotes the distance square between data point $\boldsymbol{X}_l$ and cluster center $\boldsymbol{C}_k$. The task is to, without actually reconstructing, compute T_{lk} from the shatter shares of $\boldsymbol{X}_l$ and to assign the point $\boldsymbol{X}_l$ to a closest cluster k.

T_{lk} is represented in the RNS such that T_{lkj} denotes the secret share of T_{lk} (w.r.t. P_j) available at server j. Now, each of the server j can use the Eqn: 5 to compute the share (T_{lkj}) using its locally available secret shares of $\boldsymbol{X}_{lj}$ and $\boldsymbol{C}_{kj}$.

Next, for each data point l, we need to find the cluster k such that T_{lk} is minimum. This would require reconstructing and comparing T_{lk}'s. However, to maintain privacy, the actual distances, T_{lk}'s should be kept private. We overcome this dilemma by applying a clever permutation and randomization scheme. $\boldsymbol{T}_{lk}$ is secured by applying another layer of randomization on the secret shares before sending them over for comparison to another untrusted server (thresholder). Finding the minimum of the K numbers is an $O(K)$ algorithm, i.e. the current minimum has to be compared against the next potential candidate. We next describe the protocol to find the minimum of two numbers, Z_1 and Z_2. This can then be repeated $K-1$ times to find the minimum of K numbers.

Finding the minimum: $(Z_1 - Z_2) \leq 0$ implies $Z_1 \leq Z_2$ else otherwise. In-order to check for this, at each server, we can compute the difference $Z_{1j} - Z_{2j}$ and send over the difference shares to an untrusted server for reconstruction and comparison. However, this naive approach reveals to the thresholder the distance between the two data points. We secure this by randomizing the secret shares of the differences before sending it over for comparison. We can even keep the random number itself unknown to any of the servers by the following protocol.

Each of the R servers chooses a random number r_i and sends over $r_i \ mod \ P_j$ to server j. Thus, each server j, has $\sum_{i=1}^{i=R} r_i \ \% \ P_j$ or $r \ \% \ P_j$, where $r = \sum_1^R r_i$ (Algo 2: steps 5-12). The servers uses this to randomize its share of difference. The randomized difference shares are then sent over to an un-trusted server who reconstructs the randomized difference and returns the comparison against zero for finding the minimum of the two. The smaller number is then compared against the next potential candidate. After a series of $K-1$ comparisons a data point is confidently and privately assigned

Algorithm 2. Find Minimum of K Numbers Protocol

```
 1: Let $Z_1$, $Z_2$, ... $Z_K$ be the $K$ numbers we want to find minimum of
 2: R is the number of computational servers, each knowing $Z_{kj}$, for $1 \leq k \leq K$ and $1 \leq j \leq R$, where $Z_{kj}$ is the shatter share of $Z_k$ with mod $P_j$. Note that the actual value of $Z_k$ is kept secret from all the servers.
 3: Initialize $minIndex$ = 1
 4: for every index, k = 2 to K do
 5:     for every server, j = 1 to R do
 6:         Select a positive random number $r_j$ and share the modulo of $r_j$ with every other server (step 7).
 7:         for every other server: i = 1 to R do
 8:             Send $r_{ji}$ = $r_j$ mod $P_i$ to the server $i$.
 9:         end for
10:     end for
11:     for every server, j = 1 to R do
12:         Let $r'_j$ be the summation of the $R$ random numbers received at each server $j$.
13:         Compute the difference of the secret shares of $Z_{minIndex}$ and $Z_k$. Randomize the difference by multiplying with $r'_j$.
14:         The randomized difference share is sent over to the thresholder.
15:     end for
16:     Thresholder applies the merge function to obtain $R'.(Z_{minIndex} - Z_k)$, where $R'$ is the summation of R positive random numbers $r_j$. The randomized difference is compared with 0 and the result sent back to the servers.
17:     if Threshold Result $> 0$ then
18:         minIndex = k
19:     end if
20:     For next iteration, the role of the thresholder is switched to another pseudo-randomly chosen server.
21: end for
22: Return $min\_index$
```

to a nearest cluster center. Note that the communication costs can further be reduced by choosing the random numbers offline, i.e. when the systems are idle. Each server maintains the list of the secret shares of the random numbers, r's used in the final protocol.

Correctness: Consider a point $\boldsymbol{X}$, for which we want to find which is closer $\boldsymbol{Y}$ or $\boldsymbol{Z}$. Let the points be *shattered* with scale S and randomization $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ respectively. Thus, we have:

$$(X_1, X_2, \cdots, X_D) \rightarrow (S \cdot X_1 + a_1, \cdots, S \cdot X_D + a_D) \tag{6}$$

$$(Y_1, Y_2, \cdots, Y_D) \rightarrow (S \cdot Y_1 + b_1, \cdots, S \cdot Y_D + b_D) \tag{7}$$

$$(Z_1, Z_2, \cdots, Z_D) \rightarrow (S \cdot Z_1 + c_1, \cdots, S \cdot Z_D + c_D) \tag{8}$$

Let us assume Y is closer than Z, then following holds:

$$\sum (X_i - Y_i)^2 \leq \sum (X_i - Z_i)^2 \tag{9}$$

Using the secret shares, the corresponding distances in the scaled space are computed as:

$$Dist_1 = \sum (S(X_i - Y_i) + (a_i - b_i))^2 \quad (10)$$

$$Dist_2 = \sum (S(X_i - Z_i) + (a_i - c_i))^2 \quad (11)$$

Given that Eqn: 9 holds, the protocol is correct if $Dist_1 \leq Dist_2$. From the constraints given in Sec: 3.1, we know $0 \leq a_i, b_i, c_i \leq S/2$, thus we get $-S/2 \leq (a_i - b_i) \leq S/2$.

$$\sum (S(X_i - Y_i - 1/2))^2 \leq Dist_1 \leq \sum (S(X_i - Y_i + 1/2))^2 \quad (12)$$

$$\sum (S(X_i - Z_i - 1/2))^2 \leq Dist_2 \leq \sum (S(X_i - Z_i + 1/2))^2 \quad (13)$$

Thus, the protocol satisfies correctness if Eqn: 14 is true whenever Eqn: 9 is true.

$$\sum (S(X_i - Y_i + 1/2))^2 \leq \sum (S(X_i - Z_i - 1/2))^2 \quad (14)$$

This will hold if the Cartesian System is designed so as to nullify the effect of the additional $\pm 1/2$ in Eqn: 14. This is achieved by having the step-size in the Cartesian system as 2, i.e. the data is scaled by 2 before choosing the parameters (Sec: 3.1).

Privacy: The protocol is secure against both the GCD and factorization based attacks. The servers are made to jointly choose the randomization, which is different for every threshold operation. This ensures security against the factorization based attacks. The role of the thresholder is also switched among the R servers in an random order, thus ensuring security against the GCD based attacks.

ii) Updating Cluster Locations: Once each of the L data points has been assigned to one of the K clusters, the next step is to recompute the cluster locations. For every cluster k, the cluster center is updated to the center of mass of the newly assigned points to the cluster. Thus, the new coordinate of the cluster k is a (weighted) mean of the corresponding coordinates of the n_k points assigned to the cluster k. Let n_k be the number of data points assigned to cluster k. For any cluster k, each server stores the secret shares of the data points. Each server j, can thus independently compute the sum (Sum_{kdj}) using the secret shares of the n_k data points. The updated cluster location is then obtained by dividing the sum of co-ordinates by n_k. However as we know that the generic division is not defined in the RNS, therefore we cannot directly divide the sum's shares. Furthermore, so as to maintain complete privacy, we will like to keep the updated cluster locations unknown from all the servers. Therefore, an interactive protocol, similar to the one used for thresholding is employed for the job. We now describe the *privacy-preserving division protocol (PPDP).*

PPDP: Consider a number X, secret shares of which are stored at the R servers. The task is to privately divide X by n, such that the secret X and the quotient $q = \lfloor \frac{X}{n} \rfloor$ is kept private from all of the servers. At the end of the protocol, all that the server j gets is the secret share of q w.r.t. P_j. PPDP is achieved through a single round of interaction, and the secret data, X, is secured using a permutation and a randomization method.

Just as in previous protocol (Algo: 2, steps 5-12), the R servers jointly computes two random numbers r and r', such that server j knows only the shares of them. Each server now randomizes its share of X according to Eqn: 15, before sending it over to an un-trusted server. As in the previous protocol, this server is switched among the R servers in a pseudo-permutation fashion. The randomized shares are then reconstructed using the *merge* function to compute X' (Eqn: 15).

Division is then performed to compute the randomized quotient q', as given by Eqn: 17, where q is the actual quotient that we wish to compute (Eqn: 16). We next compute the secret shares of q' and sends them over to the specific servers for de-randomization. Each server computes its share of quotient, q_j, from q'_j using Eqn: 18. The secret share of the cluster center is then updated to the computed share of the quotient. The pseudo-code of the protocol is given in algorithm 3.

Algorithm 3. Privacy Preserving Division Protocol (PPDP)

```
 1: $R$ computational servers, stores i) $X_j$ = shatter of $X$ with mod $P_j$, ii) n
 2: Randomly select $r$, $r'$, in the manner similar to as described in steps(5-12) of algorithm 2.
 3: Let at each server $j$, $r_j$, $r'_j$ be the shatter shares of the two chosen random numbers $r$ and $r'$.
 4: for each server, j = 1 to R do
 5:     Compute $X'_j = r_j \cdot (X_j + r'_j \cdot n)\ mod\ P_j$
 6:     Send $X'_j$ to the thresholder (switched among servers in a pseudo random order).
 7: end for
 8: Thresholder uses the merge function to compute $X'$
 9: Compute $q' = \lfloor \frac{X'}{n} \rfloor$
10: Send over the $q'_j$ to server $j$, where $q'$ is the shatter share of $q'$ with mod $P_j$.
11: for each server, j = 1 to R do
12:     De-randomize the received quotient to get $q_j = (q'_j * r_j^{-1} - r'_j)\ mod\ P_j$
13: end for
14: Now, $q_j$ is the required shatter share of the quotient, $q$, with prime $P_j$.
```

$$X \rightarrow X' = r \cdot (X + r' \cdot n) \tag{15}$$

$$q = \frac{X}{n} \tag{16}$$

$$q' = \frac{X'}{n} = r \cdot (q + r') \tag{17}$$

$$q_j = (q'_j * r_j^{-1} - r'_j)\ mod\ P_j \tag{18}$$

Privacy: The PPDP method provides high level of privacy for the secret data. The randomization parameters r and r' are jointly chosen and remains unknown to all. The randomization of the secret data, X, is itself done using the secret shares. The randomization function (Eqn: 15) is designed so as to safeguard against the potential attacks such as factorization and GCD based. In the entire process, no additional meaningful information is leaked to any one. The method not only provides provable privacy but is also efficient with communication cost limited to one round of interaction.

iii) Checking Termination Criterion: At the end of every iteration, we check for the closeness of the new clusters. The 'closeness' is defined as *i)* minimizing the total energy of the clusters, the energy of a cluster k is given as $E_k = \sum_1^{n_k}(\|\boldsymbol{x}_l - \boldsymbol{c}_l\|)$, *ii)* the new clusters locations are close to the old ones. i.e $\sum_1^K(\|\boldsymbol{c}_k - \boldsymbol{c}'_k\|)$, or iii) the number of points making transition across clusters is small.

If the closeness is below the threshold, then we go to step three otherwise continue with next iteration. Any of these definitions can be privately implemented using the approaches like already described.

Step three: Knowledge Revelation. At the termination of the Lloyd step, the cluster centers are stored as the secret shares at the R serves. The cluster assignment of the anonymized entities is also available. To learn the cluster locations, the servers are made to collude under legal agreements. The identity of the entities is known only to the data owner, and hence he is the only one who learns the final cluster assignment. The cluster locations can be revealed, only if agreed upon.

Analysis. We have proposed a provably secure protocol, the proofs of which are similar to those in [24]. The computation overhead at each server is limited to the randomization. The communication overhead is due to one round of interaction to simulate division and comparison operation. However, this overhead is negligible when compared to SMC. No numerical comparisons are provided, due to *a)* space constrains, and *b)* theoretical efficiency, an operation taking hundreds of rounds of communication in SMC is do able using zero or at max one round of interaction in our protocol.

4 Conclusion

We propose a novel 'cloud computing' based solution using the paradigm of Secret Sharing to privately cluster an arbitrary partitioned data among N users. Traditional approaches uses primitives such as SMC or PKC, thus compromising the efficiency and in return provide very high level of privacy which is usually an overkill in practice. The paper contributes a different approach to solve the problem. We show that privacy need not be always at the cost of efficiency. We exploit the properties of the data and the problem to circumvent the limitations faced by traditional methods (that are general-purpose). Our solution does not demand any trust among the servers or users. Security is based on the standard assumptions of honest-but-curious, non-colluding servers having ability to generate random numbers. As expected, the protocol is costly compared to the one with zero-security. However, the additional communication costs are kept to a minimum (one round) and are negligible compared to those of SMC. With the RNS being doubly homomorphic, the paradigm of shattering and merging is generic and has potential to extend over to even more diverse data mining applications.

References

1. Duda, R., Hart, P.: Pattern Classification and Scene Analysis. John Wiley and Sons, Chichester (1973)
2. Fukunaga, K.: Introduction to Statistical Pattern Recognition. Academic Press, London (1990)
3. Cranor, L.F.: Internet privacy. Commun. ACM 42(2), 28–38 (1999)

4. Turow, J.: Americans and online privacy: The system is broken. Technical Report (2003)
5. Agrawal, R., Srikant, R.: Privacy-preserving data mining. SIGMOD 29(2), 439–450 (2000)
6. Lindell, Y., Pinkas, B.: Privacy preserving data mining. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 36–54. Springer, Heidelberg (2000)
7. Vaidya, J., Clifton, C.: Privacy-preserving data mining: why, how & when. Security & Privacy, 19–27 (2004)
8. Verykios, V.S., Bertino, E., Fovino, I.N., Provenza, L.P., Saygin, Y., Theodoridis, Y.: State-of-the-art in privacy preserving data mining. SIGMOD Rec. 33(1), 50–57 (2004)
9. Kargupta, H., Datta, S., Wang, Q., Sivakumar, K.: On the privacy preserving properties of random data perturbation techniques. In: ICDM, pp. 99–106 (2003)
10. Bunn, P., Ostrovsky, R.: Secure two-party k-means clustering. In: CCS, pp. 486–497 (2007)
11. Liu, K., Giannella, C., Kargupta, H.: A Survey of Attack Techniques on Privacy-Preserving Data Perturbation Methods. Privacy-Preserving Data Mining 34(15), 359–381 (2008)
12. Oliveira, S.R.M.: Privacy preserving clustering by data transformation. In: 18th Brazilian Symposium on Databases, pp. 304–318 (2003)
13. Goldreich, O.: The Foundations of Cryptography, vol. 2. Cambridge Univ. Press, Cambridge (2004)
14. Lindell, Y., Pinkas, B.: Secure multiparty computation for privacy-preserving data mining. Cryptology ePrint Archive, Report 2008/197 (2008)
15. Vaidya, J., Clifton, C.: Privacy-preserving k-means clustering over vertically partitioned data. In: KDD (2003)
16. Inan, A., Kaya, S.V., Saygin, Y., Savas, E., Hintoglu, A.A., Levi, A.: Privacy preserving clustering on horizontally partitioned data. Data Knowl. Eng. 63(3), 646–666 (2007)
17. Jagannathan, G., Wright, R.N.: Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In: KDD, pp. 593–599 (2005)
18. Upmanyu, M., Namboodiri, A.M., Srinathan, K., Jawahar, C.V.: Blind authentication: A secure crypto-biometric verification protocol. IEEE-Transactions on Information Forensics and Security, TIFS (to appear, 2010)
19. Orlandi, C., Piva, A., Barni, M.: Oblivious neural network computing via homomorphic encryption. In: EURASIP, pp. 1–10 (2007)
20. Jha, S., Kruger, L., Mcdaniel, P.: Privacy preserving clustering. In: di Vimercati, S.d.C., Syverson, P.F., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 397–417. Springer, Heidelberg (2005)
21. Rappe, D.: Homomorphic cryptosystems and their applications. Ph.D. dissertation, University of Dortmund (2004)
22. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, pp. 169–178 (2009)
23. Asmuth, C., Bloom, J.: A modular approach to key safeguarding. IEEE Transactions on Information Theory 29, 208–210 (1983)
24. Upmanyu, M., Namboodiri, A.M., Srinathan, K., Jawahar, C.V.: Efficient privacy preserving video surveillance. In: International Conference on Computer Vision, ICCV (2009)
25. Shamir, A.: How to share a secret. ACM Communications 22(11), 612–613 (1979)
26. Beimel, A., Chor, B.: Universally ideal secret sharing schemes. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 183–195. Springer, Heidelberg (1993)
27. Goldreich, O., Ron, D., Sudan, M.: Chinese remaindering with errors. IEEE Transactions on Information Theory 46 (2000)
28. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein., C.: The chinese remainder theorem. In: Introduction to Algorithms, pp. 873–876. MIT Press, McGraw-Hill (2001)
29. Mitchell, T.: Machine Learning. McGraw-Hill, New York (1997)
30. Ulman, Z.: Sign detection and implicit-explicit conversion of numbers in residue arithmetic. IEEE Transactions on Computers C-32 (1983)