Privacy-Preserving Clustering Using Representatives over Arbitrarily Partitioned Data*

Yu Li, Sheng Zhong
Computer Science and Engineering Department
State University of New York at Buffalo
Amherst, NY 14260, U.S.A
Email: {yli32,szhong}@buffalo.edu

***Abstract*—The challenge in privacy-preserving data mining is avoiding the invasion of personal data privacy. Secure computation provides a solution to this problem. With the development of this technique, fully homomorphic encryption has been realized after decades of research; this encryption enables the computing and obtaining results via encrypted data without accessing any plaintext or private key information. In this paper, we propose a privacy-preserving clustering using representatives (CURE) algorithm over arbitrarily partitioned data using fully homomorphic encryption. Our privacy-preserving CURE algorithm allows cooperative computation without revealing users' individual data. The method used in our algorithm enables the data to be arbitrarily distributed among different parties and to receive accurate clustering result simultaneously.**

I. Introduction

With the advent of the Big Data Era, people are beginning to care more about the security of their private data. For example, when people store their data on the cloud server, there are always concerns that the service provider will use their data illegally. Cryptographers provide many elegant encryption schemes (e.g., RSA) to ensure the security of data transmission, but but they are not adequate for protecting the privacy of customers' sensitive information. After the invention of RSA, Rivest, Adleman, and Dertouzos introduced privacy homomorphism in[1]. For example, RSA is a multiplicatively homomorphic encryption scheme that can efficiently compute a ciphertext that encrypts the product of the original plaintext. Based on this privacy homomorphism method, people are realizing that they can store encrypted data instead of plaintext on the cloud. Therefore, the cloud service provider can use these data to conduct research without compromising the customers' privacy.

Clustering is the most commonly used data mining method to determine the relationship between objects. With the development of network technology, collecting large amounts of data about clients and customers has become much easier. In addition, many clustering algorithms have been proposed to solve the efficiency problem of clustering for large databases. Clustering using representatives (CURE) is such an algorithm that can efficiently cluster a large database in such a way that objects in the same group are more similar to each other than to those in other groups.

However, determining how to protect the privacy of customers is a big challenge in this age of information explosion. For example, in the field of customer behavior analysis and targeted marketing, people are more willing to communicate with others who share the same interests, they are cautious about revealing their private information to the public. Privacy-preserving data mining is the only way to solve this problem and to not invade the privacy of customers. Another example is in the bioinformatics field when two medical researchers want to study a certain disease caused by a gene. They cannot share these data with each other due to the privacy rules established by HIPAA. But privacy-preserving data mining using homomorphic encryption can determine the relationship between these data without revealing any information to the other concerned parties.

Privacy-preserving data mining is usually used for horizontally or vertically partitioned database. In [2], Jagannathan introduced a stronger assumption called the*"arbitrarily partitioned"* database, meaning that any of the different attributes for different features can be owned by any party. In this paper, we propose a privacy-preserving solution to a simple clustering algorithm ("$CURE$") over arbitrarily partitioned data. To be concrete, recall the example of collaborative customer behavior analysis and targeted marketing that we mentioned above. We assume that the attributes involved in this research are interests in food, job, religion, and entertainment, but all these data are arbitrarily partitioned by two network companies (like Facebook and LinkedIn). The two companies want to jointly cluster the people into several different groups so that people are can more conveniently to communicate with others who share their interests. In addtion, what happens if the two companies are not willing to share the real data with each other. To solve this problem, we propose a privacy-preserving method for clustering using representatives. We use a novel cryptographic techniques called fully homomorphic encryption, that can enable strong privacy protection for distributed clustering. In this paper, we present a detailed analysis of the accuracy and privacy protection of our algorithm.

Our contributions can be summarized as follows:

- We are the first to study privacy protection in clustering using representatives and to propose a privacy-preserving algorithm.
- To the best of our knowledge, we are also the first to study a privacy protection in clustering algorithm using fully homomorphic encryption.
- In terms of privacy, our algorithm leaks no knowledge about each party's data except the clustering result obtained.

*This paper was supported by NSF CNS-0845149 and CCF-0915374

The rest of the paper is organized as follows. In Section 2, we describe the related work in privacy-preserving data mining. In Section 3, we describe the technical preliminaries including the CURE algorithm and the cryptographic tool that we use in this paper. Section 4 describes our privacy-preserving protocol for the CURE algorithm when the data are arbitrarily partitioned between two parties. The analysis of the computation complexity and privacy of our protocol is giving in Section 5.

II. Related Work

Privacy preserving data mining using cryptographic protocols has developed for more than 20 years since Lindell et al. provided a privacy-preserving algorithm for ID3 in [3]. Privacy-preserving data mining based on cryptographic protocols can obtain more accurate results than methods based on a randomization algorithm. Since the cryptographic protocol is based on computation complexity, the security of the method is also better than randomization-based algorithms.

The cryptographic protocols that are used in privacy-preserving data mining are mainly based on secure computation. The most famous one is provided by Yao[4], [5] who first introduced secure two-party computation. After that, a secure multi-party computation was proposed, the goal of which is to enable several parties to jointly compute a function without revealing their inputs to each other. Another important work[1]is provided by Rivest, Adleman, and Dertouzous following the invention of RSA encryption[6]. They introduced a concept called "privacy homomorphism". Since this development, many homomorphic encryption schemes have been proposed. RSA encryption[6] is a multiplicatively homomorphic encryption scheme. ElGamal also provided an elegant multiplicatively homomorphic encryption scheme in [7]. Benaloh provided an additively homomorphic encryption scheme in [8] and Paillier proposed another one [9] in 1999. However, no fully homomorphic encryption was realized until Gentry proposed one [10] using an ideal lattice in 2009. Gentry's scheme allows one to compute arbitrary functions over ciphertext without the decryption key[11]. After Gentry proposed his fully homomorphic encryption scheme, van Dijk et al. proposed a simpler one using integers instead of the ideal lattice.

Based on these cryptographic protocols, many privacy-preserving algorithms have been proposed in the past decade. Lindell et al. provided a privacy-preserving algorithm[3] for ID3 using Yao's protocol[5]. Yang et al. proposed a privacy-preserving classification method for customer data using a variant of ElGamal's encryption [12]. Chen et al. proposed a privacy-preserving backpropagation neural network learning in [13]. All of these privacy-preserving algorithms are based on either additively homomorphic encryption or multiplicatively homomorphic encryption. Since Gentry developed the first fully homomorphic encryption scheme, many researchers have tried to provide privacy-preserving applications based on the FHE. For example, FHE enables a person to submit queries to a search engine (e.g., the user submits an encrypted query, and the search engine computes a succinct encrypted answer without ever looking at the query in plaintext). More broadly, fully homomorphic encryption improves the efficiency of secure multi-party computation. Recently, Chu et al. proposed a privacy-preserving simrank algorithm using Gentry's FHE scheme.

To be more concrete, many privacy-preserving clustering algorithms have been proposed in the past decades. Vaidya et al. proposed a privacy-preserving K-means clustering over vertically partitioned data in [14]. However, none of these researchers have been able to develop a privacy-preserving clustering algorithm using fully homomorphic encryption; therefore, we are the first to offer this type of encryption.

III. Technical Preliminary

A. Arbitrarily Partitioned Data

In two-party distributed privacy-preserving data mining, Party A and Party B have different attributes for different features. More specifically, suppose that training dataset D consists n samples ($D = \{d_1, d_2, d_3, ..., d_n\}$), and that each sample d_i contains m attributes, which denotes $d_i = \{x_{i1}, x_{i2}, ..., x_{im}\}$. The arbitrarily partitioned data based on the different attributes of the different features randomly distributed by two parties.

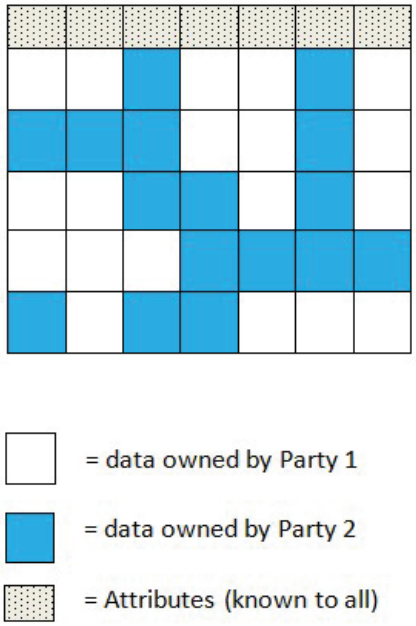

Fig. 1: Data arbitrarily partitioned by two parties

B. Fully Homomorphic Encryption Scheme

Homomorphic encryption is a form of encryption that allows specific types of computations to be carried out on ciphertext and for an encrypted result to be obtained, the decryption of which matches the result of the operations performed in plaintext. For instance, one person could add two encrypted numbers and then another person could decrypt the result without either of them being able to determine the value of the individual numbers.

To generalize this property, consider the constraint $\delta(F(\epsilon(m_1), \epsilon(m_2), ...\epsilon(m_n))) = F(m_1, m_2, ...m_n)$ for some set of functions F on all m_i in the plaintext space. We say

that an encryption scheme is partially homomorphic if F is a (finite or infinite) proper subset of computable functions. We can also say that an encryption scheme is fully homomorphic if F is the set of all computable functions.

In [10], Gentry provided the first fully homomorphic encryption scheme using ideal lattices. In [15], van Dijk et al. and Gentry et al. constructed a simple fully homomorphic encryption scheme based on Gentry's scheme[10]. van Dijk et al.'s scheme[15] only uses addition and multiplication over integers instead of ideal lattices, which is much simpler conceptually. In van Dijk et al.'s scheme, they proposed two phases to construct a fully homomorphic encryption. The first phase is constructing a somewhat homomorphic encryption. The second phase is squashing the decryption circuit to make it fully homomorphic. The following is a brief introduce of these two phases.

1) Somewhat Homomorphic Encryption: In this phase, there are four steps to construct a somewhat homomorphic encryption. They are key generation, encryption, evaluate and decryption[15].

- Key Generation.
  In the key generation step, an odd η-bit integer p is randomly chosen from $(2\mathbb{Z}+1)\cap[2^{\eta-1},2^{\eta})$ as the private key. The corresponding public key is $pk =< x_0, x_1, ..., x_\tau >$, which $x_i \leftarrow D_{\gamma,\rho}(p)$ and $D_{\gamma,\rho}(p) = x = pq + r$ and $q \leftarrow \mathbb{Z}\cap[0, 2^\gamma/p), r \leftarrow \mathbb{Z}\cap(-2^\rho, 2^\rho)$
- Encryption.
  Then choose a random subset $s \in 1, 2, ..., \tau$ and a random integer r in $(-2^{\rho'}, 2^{\rho'})$, and output $c \leftarrow [m + 2r + 2\sum_{i\in S}(x_i)]_{x_0}$
- Evaluate
  Given the binary circuit C_ε with t inputs, and t ciphertexts c_i, apply the integer addition and multiplication gates of C_ε to the ciphertexts, performing all the operations over the integers, and return the resulting integer
- Decryption.
  Output $m' \leftarrow (c \mod p) \mod 2$.

2) Squashing the Decryption Circuit: In van Dijk et al.'s fully homomorphic scheme, they followed Gentry's approach[10] to make their scheme bootstrappable in this phase.

- Key Generation.
  An odd η-bit integer p is randomly chosen from $(2\mathbb{Z}+1)\cap[2^{\eta-1},2^{\eta})$ as the private key. The corresponding public key is $pk =< x_0, x_1, ..., x_\tau >$, which $x_i \leftarrow D_{\gamma,\rho}(p)$ and $D_{\gamma,\rho}(p) = x = pq + r$ and $q \leftarrow \mathbb{Z}\cap[0, 2^\gamma/p), r \leftarrow \mathbb{Z}\cap(-2^\rho, 2^\rho)$. Set $x_p \leftarrow$
  After generate the private key and public key, choose at random a $\theta - bit$ vector with Hamming weight $\theta, s =< s_1, ..., s_\theta >$, and let $S = i : s_i = 1$.
  Choose at random integers $u_i \in \mathbb{Z}\cap[0, 2^{k+1}), i = 1, ..., \theta$ subject to the condition that $\sum_{i\in S} u_i = x_p (mod 2^{k+1})$. Set $y_i = u_i/2^k$ and $y = y_1, ..., y_\theta$. Hence each y_i is a positive number smaller than 2, with k bits of precision after the binary point. Also, $[\sum_{i\in S} y_i]_2 = (1/p) - \Delta_p$ for some $|\Delta_p| < 2^{-k}$.

TABLE I: Denotations of van Dijk et al's scheme[15]

γ is the bit-length of the integers in the public key
η is the bit-length of the secret key
ρ is the bit-length of the noise
τ is the number of integers in the public key

  Output the secret key $sk = s$ and public key $pk = (pk^\star, y)$.
- Encryption and Evaluate.
  Generate a ciphertext $c^\star$ as before (i.e., an integer). Then for $i \in 1, ..., \theta$, set $z_i \leftarrow [c^\star \cdot y_i]_2$, keeping only $n = \lceil log\theta \rceil + 3$ bits of precision after the binary point for each z_i. Output both $c^\star$ and $z =< z_1, ..., z_\theta >$.
- Decryption.
  Output $m' \leftarrow [c^\star - \lfloor \sum_i s_i z_i \rceil]_2$

C. Notations for CURE algorithm

CURE is an efficient data clustering algorithm for large databases[16]. It is more robust for outliers and can identify clusters with non-spherical shapes and wide variances in size.

The CURE algorithm uses distance to denote the relationship of each pair of points. Then it merges the closest pair into the same cluster. It also uses a constant number of well scattered points to represent one cluster so that it can speed up the clustering process for large database. Here we briefly introduce how the CURE algorithm works. First, it calculates the distance of each pair of points and finds the nearest neighbor of each point. It then puts the nearest neighbor of each point alone with that point into a queue in increasing order. When the size of the queue is bigger than k, it merges the top two clusters and updates the distance to the new cluster. In the merge process, it uses a constant number of well scattered points to represent the cluster that is shrunk toward the center of the cluster by a fraction α. Then it iteratively process above progress until the size of the queue equals to k. Then, the algorithm outputs these k clusters. [16] gives more details regarding the algorithm. Figure 2 gives an example of the CURE clustering of seven different clusters.

IV. Model Description

A. Definitions and Problem Statement

In this paper, we present a privacy-preserving CURE algorithm over arbitrarily partitioned data in different parties. The goal of our method is to protect the data privacy and to achieve the computation of the clustering simultaneously. For simplicity, we will mainly focus on a two-party scenario.

In a two-party scenario, data are arbitrarily partitioned distributed between the two parties. That means for each instance d_i, Party A holds data x_{ij} and Party B holds data $x_{ij\prime}$. Without loss of generality, for each $d_i \in \mathbb{D}, d_i = x_{ij} \cup x_{ij\prime}$, Party A and Party B want to determine clustering using the CURE algorithm cooperatively without revealing data to each other. For this problem, we propose a privacy-preserving CURE algorithm using fully homomorphic encryption.

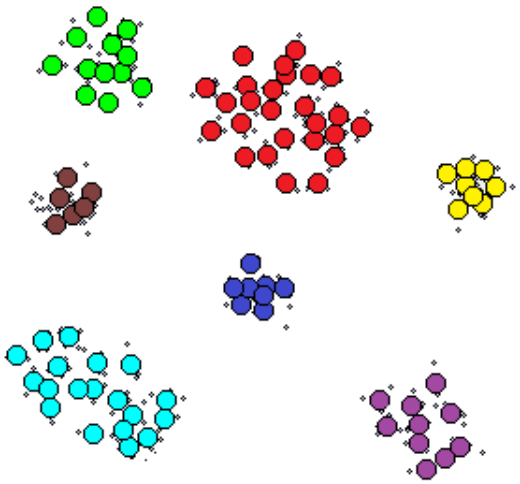

Fig. 2: CURE algorithm with 7 clusters

B. Privacy-preserving CURE algorithm using fully homomorphic encryption

We now describe our privacy-preserving CURE algorithm using fully homomorphic encryption. Party A and Party B share a data set in an arbitrary partition, which we explained in section 3.1. Since the CURE algorithm uses random sampling to handle large data sets, in this step, we can simply use Reservoir sampling to obtain a smaller data set. It is easy to see that this step does not influence the user's privacy. In addition, the CURE algorithm also uses partitioning for increasing the speed. The basic idea is to partition the sample space into p partitions. Each partition contains n/p elements. In the first pass, we partially cluster each partition until the final number of clusters is reduced to n/pq for the constant $q \geq 1$. Then, we run a second clustering pass on n/q partial clusters for all the partitions. For the second pass, we only store the representative points since the merge procedure only requires representative points of previous clusters before computing the new representative points for the merged cluster. Obviously, this step only reduces the execution times. Therefore, the main challenge of the privacy-preserving CURE algorithm is how to protect privacy when Party A and Party B jointly execute the CURE algorithm.

In the original CURE algorithm, a k-d tree is used to store the representative points, and it can sort the multi-dimensional point data efficiently so that it is easy to retrieve. But since we need to preserve the privacy of the data, and this k-d tree which contains a sorting process will influence the efficiency enormously, we simply use a k-d matrix to store these points in stead of a k-d tree. In our two-party model, the k-d matrix is arbitrarily partitioned by Party A and Party B. We propose how to build a heap Q securely using van Dijk et al's scheme.

First, Party A generates a secret key $sk_A = p_A$ and Party B generates his secret key $sk_B = p_B$ based on van Dijk et al.'s scheme. Since the whole computation process will be based on bit manipulation. Party A and Party B need to convert their data to a binary format. Party A generates his public key pk_A and encrypts his data and sends them to Party B. Then Party B encrypts his data using Party A's public key. For clarity, we assume that parties are both using Euclidean distance as their $Dist$. Therefore, the main challenge of the privacy-preserving CURE cluster algorithm is obtaining a secure comparison of the distance and determining how to find the closet one. Since we use Euclidean distance, we have

$$dist(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + ...(q_n - p_n)^2} \quad (1)$$

Since we only need to find the two points that have smallest distance, we can use the square of distance for easily computation. For each pair of points, since the attributes of the two points are arbitrarily partitioned by the two parties, the two data sets belong to the same attribute that is either owned by one party or distributed by the two parties. Without generality, we assume that Party A owns a_i^p and Party B owns b_j^p. We can simple get $(a_i^p - b_j^p)^2 = (a_i^p)^2 + (b_j^p)^2 + 2a_i^p \times b_j^p$. $(a_i^p)^2$ and $(b_j^p)^2$ can be calculated by each party itself. For $2a_i^p \times b_j^p$, we can use the full homomorphism of van Dijk et al. scheme to obtain the random share of the distance as shown in algorithm 1 and algorithm 2. For example, in algorithm 2, the random share for Party A is $R_A + dist\{i,j\} + R_B$, and for Party B is R_B. Another random share for Party A is R_A, and for Party B is $R_A + dist\{i',j'\} + R_B$. After obtaining the random share of two distances, we can use secure compare which is proposed in [17](algorithm 3) to compare them. Secure compare[17] uses two binary numbers as the input. First, Party A encrypts each bit and sends them to Party B in order. Party B then calculates $c_i = E_A(x_B^i - x_A^i + 1 + \sum_{j=1}^{i-1} w_j)$ for each i. If $x_A > x_B$, then there must exist one bit that the bits before i are the same and x_A^i is 1 and x_B^i is 0, therefore making $D_A(c_i) = 0$. Since $(x_B^i - x_A^i + 1 + \sum_{j=1}^{i-1} w_j) = x_B^i - x_A^i + 1 + \sum_{j=1}^{i-1}(x_A^j + x_B^j - 2 \cdot x_A^j \cdot x_B^j)$, this can be calculated by Party B using the full homomorphism of van Dijk et al.'s scheme. After getting the $index(i)$ which indicates the instance that has the smallest distance from instance i, we need to compare them again to get smallest one $v := index(u)$ using the algorithm 3 again. Next we can merge u and v. First, we iteratively select c points. We can use varietal algorithm 2 to calculate the distance between p and $\frac{|u|\, u.mean + |v|\, v.mean}{|u| + |v|}$. It is easy to get this varietal algorithm, as we can assume that $\frac{|u|\, u.mean + |v|\, v.mean}{|u| + |v|}$ is a point. When calculating it, we can split it into several parts that contain u and v. Then we can use method in algorithm 2 to get the value $R_A + dist(p, \frac{|u|\, u.mean + |v|\, v.mean}{|u| + |v|} + R_B)$. Then Party A and Party B jointly apply algorithm 3 to secure get the $temSet$. After that, we use variant algorithm 2 again to choose the point that is furthest from the previously selected points. That means we let the points shrink toward the mean by a fraction α using a varietal algorithm 2. Then we can insert this new merged cluster into the matrix dataset. After that, we can use the same method (algorithm 2) to update the array $index(i)$. Then we iteratively process above procedure until the size of array Z is equal k. Lastly, we can output these K clusters.

Algorithm 1 Privacy-preserving CURE cluster algorithm

Input Party A and party B with their own data$S$
Output k clusters
begin
Party A generate his private key p_A and public key pk_A
Party B generate his private key p_B and public key pk_B
initialize an array $Z := index(i)$.
for each of instances i {
for each of instances$j \neq i$ **do**{
By applying algorithm 2, party A and party B can jointly compare the distance of i and j and store the smaller one into $index(i)$
}
end for
$index(i)$ indicates the instance that has the smallest distance from instance i

While $size(Z) > k$ **do**
{
Party A and party B jointly apply algorithm 2 to get $u := extractmin(Z)$
$v := index(u)$ //v := u.closest//
delete(Z, u)
$w := u \cup v$
$temSet := \phi$
for $i := 1$ **to** c **do**{
$maxDist := 0$
for each point p in cluster w **do**{
if $i = 1$
Party A and B jointly apply algorithm 2 to get $minDist := dist(p, \frac{|u|\, u.mean + |v|\, v.mean}{|u| + |v|})$
else
Party A and B jointly apply algorithm 2 to get $minDist := mindist(p, q) : q \in tmpSet$
Party A and B jointly apply algorithm 3 to secure compare
if $(minDist \geq maxDist)$**{**
$maxDist := minDist$
$maxPoint := p$
}
}
$tmpSet := tmpSet \cup maxPoint$
}
for each point p in temSep **do**
Party A and B jointly apply variant algorithm 2 to get $w := w \cup \{p + \alpha * (\frac{|u|\, u.mean + |v|\, v.mean}{|u| + |v|} - p)\}$
$insert(M, w)$
$w.closest := x$ //x is an arbitrary cluster in Z//
for each $x \in Z$ **do**{
Party A and B jointly apply algorithm 3 to secure compare
if $dist(w, x) < dist(w, w.closest)$
$w.closest := x$
if$x.closest$ is either u or v {
if $dist(x, x.closest) < dist(x, w)$
$x.closest := closest\_cluster(T, x, dist(x, w))$
else
$x.closest := w$
else if $dist(x, x.closest) > dist(x, w)$**{**
$x.c1osest := w$
}
}
$insert(Z, w)$
}
end

Algorithm 2 Privacy-preserving find minimum distance

Input two distance patitioned by party A and B
Output the one has smaller distance
begin
Party A sends $\{E_A(a_i^p)\}$ to party B for all p_{th} attribute belong to party A and all b_j^p belong to party B.
Party A also sends $\{E_A(a_j^q)\}$ to party B for all q_{th} attribute belong to party A and all b_i^q belong to party B.
Party B calculates $\{E_A((b_i^\beta - b_j^\beta)^2 - (2a_i^p \times b_j^p + 2a_j^q \times b_i^q) + R_B)\}$ and sends this to party A
Party A calculates $\{E_A((a_i^\alpha - a_j^\alpha)^2 + (b_i^\beta - b_j^\beta)^2 - (2a_i^p \times b_j^p + 2a_j^q \times b_i^q) + R_B)\}$ and then decrypts it to get $dist\{i, j\} + R_B$.
Using same method, party B can get $dist\{i', j'\} + R_A$.
Party A calculates $R_A + dist\{i, j\} + R_B$
Party B calculates $R_A + dist\{i', j'\} + R_B$
By applying algorithm 3, party A and party B can jointly $securecompare(R_A + dist\{i, j\} + R_B, R_A + dist\{i', j'\} + R_B)$ and store the smaller one.

Algorithm 3 Secure Compare two binary number[17]

Input Party A holds $X_A = \{x_A^1, x_A^2 ... x_A^n\}$. Party B holds $X_B = \{x_B^1, x_B^2 ... x_B^n\}$
Output $X_A > ? X_B$
begin
Party A sends $\{E_A(x_A^i)\}$ to party B.
Party B calculates $c_i = E_A(x_B^i - x_A^i + 1 + \sum_{j=1}^{i-1} w_j)$ where $w_j = x_A^j \oplus x_B^j = x_A^j + x_B^j - 2 \cdot x_A^j \cdot x_B^j$
Party B permutes c_i and send them to party A
Party A decrypts all c_i using p_A.
if exist any $c_i == 0$
$X_A > X_B$
else
$X_A \leq X_B$
end

V. Correctness and Security Analysis

We have now described our privacy-preserving CURE algorithm. In this section, we conduct a detailed analysis of the correctness and security of our algorithm.

A. Correctness analysis

After using our privacy-preserving CURE algorithm, Party A and Party B can combine their partial knowledge of the dataset to get the finally clusters. Now we show that this combined result is correct, that is equal to the clusters when using the original algorithm without privacy protection.

Theorem 1. *For the clustering process, our algorithm 1 (with the input arbitrarily partitioned by the two parties) produces the clusters that are the same as the corresponding clusters when running the original version of the CURE algorithm with the whole dataset without any privacy protection*

Proof. We first show the correctness of algorithm 2. Party A and Party B can jointly compare the two numbers $R_A + dist\{i, j\} + R_B$ and $R_A + dist\{i', j'\} + R_B$ since the correctness of the secure compare two binary number algorithm is guaranteed in [17]. Obviously, the comparison result is same as the comparison result between $dist\{i, j\}$ and $dist\{i', j'\}$, so the correctness of the distance comparison is proven. Now we show that in each iteration, our algorithm can correctly obtain the $R_A + dist\{i, j\} + R_B$. We note that the distance we used in our comparison is the square of the

Euclidian distance. The components of the $dist\{i,j\}$ either belong to Party A or Party B. Since we use van Dijk et al.'s fully homomorphic encryption, we can directly achieve the $E_A(R_A+dist\{i,j\}+R_B)$ through the communication of Party A and Party B. The fully homomorphic property list below has been guaranteed in[15].

- Multiplication homomorphic property $\delta(\epsilon(m_1) \times \epsilon(m_2) \times \epsilon(m_3)... \times \epsilon(m_n)) = m_1 \times m_2 \times m_3... \times m_n$
- Addition homomorphic property $\delta(\epsilon(m_1) + \epsilon(m_2) + \epsilon(m_3)... + \epsilon(m_n)) = m_1 + m_2 + m_3... + m_n$

Since we let the distance hide behind the two random numbers R_A and R_B, neither of the parties can obtain the intermediate result. This completes the proof of the theorem.

B. Security analysis

In this subsection, we explain why our algorithm is secure in the semi-honest model. In the semi-honest model, we say an algorithm is secure if neither party can learn anything beyond its output from the information obtained throughout the algorithm. In our algorithm, they can only get information from others when they communicate with each other. There are two parts that communicate with each other. In exchanging $E_A(R_A + dist\{i,j\} + R_B)$, since R_B is randomly chosen by Party B, Party A can not learn the value of $dist\{i,j\}$ from $dist\{i,j\} + R_B)$. This is also applied to Party B. In secure compare two binary numbers algorithm, the security also has been guaranteed in [17]. Therefore, our algorithm is secure in the semi-honest model.

VI. Conclusions

In this paper, we have proposed a privacy-preserving algorithm for clustering using representatives when the input data are arbitrarily partitioned between two parties. Our algorithm is correct and provides strong privacy guarantees.

References

[1] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[2] Geetha Jagannathan and Rebecca N Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 593–599. ACM, 2005.

[3] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology CRYPTO 2000*, pages 36–54. Springer, 2000.

[4] Andrew C Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[5] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[6] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[7] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

[8] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 544–553. ACM, 1994.

[9] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology–EUROCRYPT 1999*, pages 223–238. Springer, 1999.

[10] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[11] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[12] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology–EUROCRYPT 2000*, pages 539–556. Springer, 2000.

[13] Tingting Chen and Sheng Zhong. Privacy-preserving backpropagation neural network learning. *Neural Networks, IEEE Transactions on*, 20(10):1554–1564, 2009.

[14] Jaideep Vaidya and Chris Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215. ACM, 2003.

[15] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology–EUROCRYPT 2010*, pages 24–43. Springer, 2010.

[16] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *ACM SIGMOD Record*, volume 27, pages 73–84. ACM, 1998.

[17] Ivan Damgard, Martin Geisler, and Mikkel Kroigard. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*, 1(1):22–31, 2008.