Privacy-Preserving and Outsourced Multi-User *k*-Means Clustering

Fang-Yu Rao*, Bharath K. Samanthula*, Elisa Bertino*
*Purdue University
West Lafayette, IN, USA
{raof,bsamanth,bertino}@purdue.edu

Xun Yi†
†RMIT University
Melbourne, VIC, Australia
xun.yi@rmit.edu.au

Dongxi Liu‡
‡CSIRO Computational Informatics
Marsfield, NSW, Australia
dongxi.liu@csiro.au

***Abstract*—Many techniques for privacy-preserving data mining (PPDM) have been investigated over the past decade. Often, the entities involved in the data mining process are end-users or organizations with limited computing and storage resources. As a result, such entities may have to refrain from participating in the PPDM process. To address this issue, outsourcing PPDM tasks to the cloud environment could be an interesting solution. In a cloud-based scenario, n entities outsource their databases (in encrypted format) to the cloud and ask the cloud to perform the clustering task on their combined data in a privacy-preserving manner. We term such a process as privacy-preserving and outsourced distributed clustering (PPODC). In this paper, we propose a novel and efficient solution to the PPODC problem based on the k-means clustering algorithm. The main novelty of our solution lies in avoiding the secure division operations required in computing cluster centers altogether through efficient transformation techniques. The proposed solution protects data confidentiality of all the participating entities under the standard semi-honest model. Also, we discuss two strategies, namely offline computation and pipelined execution, that aim to boost the performance of our protocol. We implement our protocol on a cluster of 16 nodes and demonstrate how our two strategies combined with parallelism can significantly improve the performance of our protocol through extensive experiments using a real dataset.**

I. Introduction

Clustering [1] is one of the widely used techniques in various data mining applications in several fields, including information retrieval [2], machine learning [3], and pattern recognition [4]. Real-life applications related to clustering include categorizing results returned by a search engine in response to a user's query, and grouping persons into categories based on their DNA information.

In general, if the data involved in clustering belongs to a single entity (hereafter referred to as a user), then it can be done in a trivial fashion. However, in some cases, multiple users, such as companies, governmental agencies, and health care organizations, each holding a dataset, may want to collaboratively perform clustering on their combined data and share the clustering results. Due to privacy concerns, users may not be willing to share their data with the other users and thus the distributed clustering task should be executed in a privacy-preserving manner. This problem, referred to as privacy-preserving distributed clustering (PPDC), can be best explained by the following example:

- Consider two health providers each holding a dataset containing the disease patterns and clinical outcomes of their patients. Suppose that the providers want to cluster their combined datasets and identify interesting clusters that would enable directions for better disease prevention. However, suppose that due to the sensitive nature of the data, they are not able to share their data. Hence, they have to collaboratively perform the clustering task on their joint datasets in a privacy-preserving manner. Once the clustering task is done, they can exchange necessary information (after proper sanitization) if needed.

Existing PPDC methods (e.g., [5]–[8]) incur significant cost (computation, communication and storage) on the participating users and thus they are not suitable if the users do not have sufficient resources to perform the clustering task. This problem becomes even more serious when dealing with big data. To address these issues, it is more attractive for the users to outsource their data as well as the clustering task to the cloud. However, the cloud cannot be fully trusted by the users in protecting their data. Thus, to ensure data confidentiality, users can encrypt their databases locally (using a common public key) and then outsource them to the cloud. Then, the goal is for the cloud to perform clustering over the aggregated encrypted data. We refer to the above process as *privacy-preserving and outsourced distributed clustering (PPODC)*.

It is worth noting that if all the encrypted data resides on a single cloud, then the only way for the cloud to execute the clustering task (assuming that users do not participate in the clustering process), without ever decrypting the data, is when the data is encrypted using fully homomorphic encryption schemes (e.g., [9]). Past results [10], however, show that fully homomorphic encryption schemes are very expensive and their usage in practical applications is decades away.

In this paper, we propose a new and efficient solution to the PPODC problem based on the standard k-means clustering algorithm [11], [12] with the use of two cloud service providers (say Amazon and Google) which together form a federated cloud environment. Our proposed solution protects data confidentiality of all the participating users at all times. We emphasize that the concept of federated clouds is becoming increasingly popular and is also identified as one of the ten High Priority Requirements for U.S. cloud adoption in the NIST U.S. Government Cloud Computing Technology Roadmap [13]. Therefore, we believe that developing privacy-preserving solutions under federated cloud environments will become increasingly important in the near future.

A. *System Model and Problem Definition*

In our problem, we consider n users denoted by $U_1, \dots, U_n$. Assume that user U_i holds a database T_i with m_i data records and l attributes, for $1 \leq i \leq n$. Consider a scenario where the n users want to outsource their databases as well as the k-means clustering process on their combined databases to a cloud environment. In our system model, we consider two different entities: (i) the users and (ii) the cloud service providers. We assume that the users choose two cloud service providers C_1 and C_2 to perform the clustering task on their combined data.

We explicitly assume that C_1 and C_2 are semi-honest[1] [14] and they do not collude. After proper service level agreements with the users, C_2 generates a public-secret key pair (pk, sk) based on the Paillier cryptosystem [15] and broadcasts pk to all users and C_1. A more robust setting would be for C_1 and C_2 to jointly generate the public key pk based on the threshold Paillier cryptosystem (e.g., [16], [17]) such that the corresponding secret key sk is obliviously split between the two clouds. Under this case, the secret key sk is unknown to both clouds and only (random) shares of it are revealed to C_1 and C_2. For simplicity, we consider the former asymmetric setting where C_2 generates (pk, sk) in the rest of this paper.

Given the above system architecture, we assume that user U_i encrypts T_i attribute-wise using pk and outsources the encrypted database to C_1. Another way to outsource the data is for users to split each attribute value in their database into two random shares and outsource the shares separately to each cloud (see Section V-B for more details). A detailed information flow between different entities in our system model is shown in Figure 1. Having outsourced the data, the main goal of a PPODC protocol is to enable C_1 and C_2 to perform k-means clustering over the combined encrypted databases in a privacy-preserving manner. More formally, we can define a PPODC protocol as follows:

$$\text{PPODC}(\langle T_1, \ldots, T_n \rangle, \beta) \rightarrow (S_1, \ldots, S_n) \qquad (1)$$

where β is a threshold value agreed upon by all parties and used to check whether the termination condition holds in each iteration of the k-means algorithm (see Sections III and V for more details). Depending on the users' requirements, S_i (output received by U_i) can be the the global cluster centers and/or the final cluster IDs corresponding to the data records of U_i. In this paper, we consider the former case under which S_i's are the same for all users. In general, a PPODC protocol should meet the following requirements:

- **Data Confidentiality:** The contents of U_i's database T_i should never be revealed to other users, C_1 and C_2.
- **Accuracy:** The output received by each party (i.e., S_i's) should be the same as in the standard k-means algorithm.
- **No Participation of Users:** Since the very purpose of outsourcing is to shift the users' load towards the cloud environment, a desirable requirement for any outsourced task is that the computations should be totally performed in the cloud. This will enable the users who do not have enough resources to participate in the clustering task to still get the desired results without compromising their privacy.

B. Main Contributions

In this paper, we propose an efficient and novel PPODC protocol that can enable a group of users to outsource their encrypted data as well as the k-means clustering task completely to a federated cloud environment. The main contributions of this work are five-fold:

[1]Under the semi-honest model, each party follows the prescribed steps of the protocol, but is free to deduce any additional information based on the data it sees during the execution of the protocol.

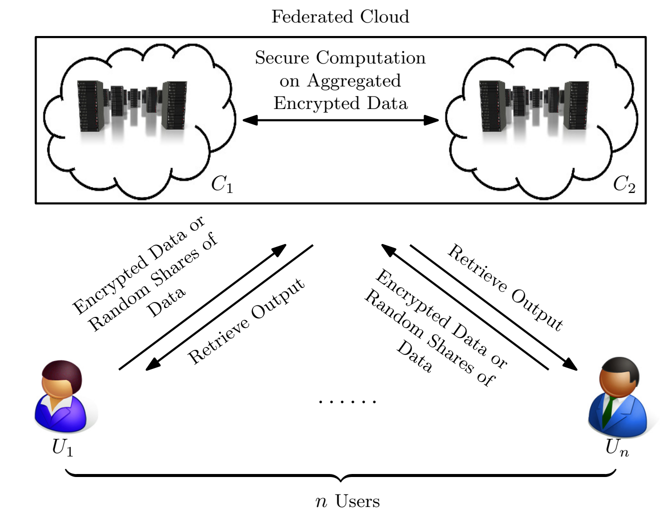

Federated Cloud
Secure Computation on Aggregated Encrypted Data
C_1
C_2
Encrypted Data or Random Shares of Data
Retrieve Output
Encrypted Data or Random Shares of Data
Retrieve Output
......
U_1
U_n
n Users

Fig. 1. The Proposed PPODC Architecture

- We propose new transformations in order to develop an order-preserving Euclidean distance function that enables our proposed protocol to securely assign the data records to the closest clusters, a crucial step in each iteration of the k-means clustering algorithm (see Sections IV-A and V-B2). Also, we propose a novel transformation for the termination condition that enables our protocol to securely evaluate the termination condition over encrypted data (see Sections IV-B and V-B3).
- We discuss novel constructions of two cryptographic primitives, namely secure squared order-preserving Euclidean distance and secure minimum out of k numbers, which are used as building blocks in our proposed solution (see Section V-A).
- The proposed solution satisfies all the desirable properties of PPODC mentioned in the previous subsection, i.e., it protects the confidentiality of each user's data at all times and outputs the correct result. Specifically, we show that the proposed protocol is secure under the standard semi-honest model (see Section V-C). We emphasize that, once the user's data is outsourced to the cloud, the user does not need to participate in any computations of the clustering task.
- We present two strategies to improve the performance of our protocol. The first strategy, referred to as offline computation, allows the cloud to pre-compute some data independent intermediate results, thus boosting online computation time. The second strategy, referred to as pipelined execution, relies on pipelining the underlying computations (see Section 5.4).
- By using a real dataset, we experimentally show that the above techniques can indeed boost the performance of our protocol. Moreover, we demonstrate how the performance of our protocol can be drastically improved using parallel implementation on a cluster of 16 nodes (see Section VI).

The remainder of this paper is organized as follows. Section II discusses related work. Section III introduces definitions and properties related to k-means clustering algorithm. Section IV presents our new transformation techniques. Section V discusses our proposed PPODC solution in detail. Also, we analyze the security guarantees of PPODC and discuss two performance improvement strategies. Section VI presents our experimental results based on a real-world dataset under different parameter settings. Finally, we conclude the paper along with future research in Section VII.

II. Related Work

Several techniques have been proposed for the clustering task under the privacy-preserving data mining (PPDM) model (e.g., [5]–[8]). In PPDM, each user owns a piece of a dataset (typically a vertically or horizontally partitioned dataset) and the goal is for them to collaboratively perform the clustering task on the combined data in a privacy-preserving manner. On the other hand, our work is motivated by the cloud computing model where users can outsource their encrypted databases to a federated cloud environment. Under our problem setting, the federated cloud performs the clustering task over encrypted data and the users do not participate in any of the underlying computations. Hence, existing PPDM techniques for the clustering task are not applicable to the PPODC problem.

Only recently, researchers have started to focus on the clustering task in an outsourced environment (e.g., [18]–[20]). In [18], Upmanyu et al. give a solution to privacy-preserving k-means clustering for data records collected from n users based on secret sharing scheme where at least three non-colluding cloud service providers are required to perform the computation. The work by Lin [19] proposes to utilize the randomized kernel matrix to protect the data privacy in the problem of outsourced kernel k-means computations. Only one service provider is needed in Lin's scheme. But, it is not clear that how much information is disclosed to the service provider since its security is not formally proved in the semi-honest model. The work by Liu et al. [20] is perhaps the most recent work along this direction. However, their solution has the following limitations: (i) it assumes that there is only a single user who wants to perform the clustering task on his/her own data and (ii) the user is required to execute certain intermediate computations and thus needs to be part of the clustering process. Unlike the approach in [20], our solution is proposed under the multi-user setting and the users can completely outsource the computations of the clustering task to a federated cloud environment in a privacy-preserving manner.

III. Preliminaries

Definition 1: Suppose $c = \{t_1, \ldots, t_h\}$ denote a cluster where $t_1, \ldots, t_h$ are data records with l attributes. Then, the center of cluster c is defined as a vector μ_c given by [8]:

$$\mu_c[s] = \frac{t_1[s] + \cdots + t_h[s]}{|c|} = \frac{\lambda_c[s]}{|c|}, \text{ for } 1 \le s \le l \quad (2)$$

where $t_i[s]$ denotes the s^{th} attribute value of t_i and $\lambda_c[s]$ denotes the sum of s^{th} attribute values of all the data records in cluster c, for $1 \le i \le h$. Also, $|c|$ denotes the number of data records in c.

In the above definition, the s^{th} attribute value in μ_c is equivalent to the mean of the s^{th} attribute values of all the data records in cluster c. Note that, if the cluster contains a single data record, then the cluster center is the same as the corresponding data record.

Example 1: Let c be a cluster with three data records given as: $t_1 = \{0, 2, 1, 0, 3\}, t_2 = \{1, 1, 3, 4, 2\}$, and $t_3 = \{0, 1, 0, 2, 0\}$. Based on Definition 1, the center of cluster c is given by $\mu_c[1]$ = 0.333, $\mu_c[2]$ = 1.333, $\mu_c[3]$ = 1.333, $\mu_c[4]$ = 2, $\mu_c[5]$ = 1.666. □

A. Euclidean Distance between t_i and c

We now discuss how to compute the similarity score between a given data record t_i and a cluster c. In general, the similarity score between any two records can be computed using one of the standard similarity metrics, such as Euclidean distance and Cosine similarity. In this paper, we use the Euclidean distance as the underlying similarity metric since the standard k-means algorithm is based on this metric [8], [20].

Definition 2: For any given data record t_i and cluster c, let μ_c denote the cluster center of c (as per Definition 1). Then the Euclidean distance between t_i and c is given as

$$\|t_i - c\| = \sqrt{\sum_{s=1}^{l} (t_i[s] - \mu_c[s])^2} = \sqrt{\sum_{s=1}^{l} \left(t_i[s] - \frac{\lambda_c[s]}{|c|}\right)^2}$$

Example 2: Suppose that $t_i = \{0, 1, 1, 3, 2\}$ and $\mu_c = \{0.333, 1.333, 1.333, 2, 1.666\}$. Then, the Euclidean distance between t_i and c, based on Definition 2, is $\|t_i - c\| = 1.201$. □

In a similar manner, the Euclidean distance between any two given clusters c and c' can be computed using their respective cluster centers. More specifically, $\|c - c'\|$ is given as

$$\sqrt{\sum_{s=1}^{l} (\mu_c[s] - \mu_{c'}[s])^2} = \sqrt{\sum_{s=1}^{l} \left(\frac{\lambda_c[s]}{|c|} - \frac{\lambda_{c'}[s]}{|c'|}\right)^2}$$

where μ_c and $\mu_{c'}$ denote the cluster centers of c and c', respectively. Also, $|c|$ and $|c'|$ denote the number of data records in c and c'.

B. Single Party k-Means Clustering

Consider a user U who wants to apply the k-means clustering algorithm [11], [12] on his/her own database of m records, denoted by $\{t_1, \ldots, t_m\}$. Here we assume that U wants to compute k cluster centers, denoted by $\mu_{c'_1}, \ldots, \mu_{c'_k}$, as the output. However, other desired values, such as the final cluster IDs assigned to each data record can also be part of the output. Since k-means clustering is an iterative algorithm, U has to input a threshold value to decide when to stop the algorithm (termination condition). Without loss of generality, let β denote the threshold value. For simplicity, throughout this paper, we assume that the initial set of k clusters are chosen at random (referred to as the Initialization step).

The main steps involved in the traditional (single party) k-means clustering task [11], [12], using the Euclidean distance as the similarity metric, are given in Algorithm 1. Apart from the initialization step, the algorithm involves three main stages: (i) Assignment (ii) Update and (ii) Termination. In the initialization step, k data records are selected at random and assigned as the initial clusters $c_1, \ldots, c_k$ with their centers (or mean vectors) denoted by $\mu_{c_1}, \ldots, \mu_{c_k}$, respectively. In the assignment stage, for each data record t_i, the algorithm computes the Euclidean distance between t_i and each cluster c_j, for $1 \le j \le k$. The algorithm identifies the cluster corresponding to the minimum distance as the closest cluster

Algorithm 1 k-means$(\{t_1, \ldots, t_m\}, \beta) \rightarrow \{\mu_{c'_1}, \ldots, \mu_{c'_k}\}$

```
Require: User $U$ with $m$ data records $\{t_1, \ldots, t_m\}$ and $\beta$
    Initialization: Select $k$ data records at random and assign them as initial clusters $c_1, \ldots, c_k$ with respective cluster centers as $\mu_{c_1}, \ldots, \mu_{c_k}$
 1: for $j = 1$ to $k$ do
 2:     $c'_j \leftarrow \emptyset$, $\mu_{c'_j} \leftarrow \{\}$ and $sum \leftarrow 0$
 3: end for
    {Assignment Stage}
 4: for $i = 1$ to $m$ do
 5:     for $j = 1$ to $k$ do
 6:         Compute $\|t_i - c_j\|$
 7:     end for
 8:     Add $t_i$ to cluster $c'_h$ such that $\|t_i - c_h\|$ is the minimum, for $1 \leq h \leq k$
 9: end for
    {Update Stage}
10: for $j = 1$ to $k$ do
11:     Compute cluster center for $c'_j$ and assign it to $\mu_{c'_j}$
12: end for
    {Termination Stage - Compare the old clusters ($c_j$'s) with new clusters ($c'_j$'s) and check whether they are close enough}
13: $sum \leftarrow \sum_{j=1}^{k} \|c_j - c'_j\|^2$
14: if $sum \leq \beta$ then
15:     Return $\{\mu_{c'_1}, \ldots, \mu_{c'_k}\}$
16: else
17:     for $j = 1$ to $k$ do
18:         $c_j \leftarrow c'_j$ and $\mu_{c_j} \leftarrow \mu_{c'_j}$
19:     end for
20:     Go to Step 6
21: end if
```

to t_i (say c_h) and assigns t_i to a new cluster c'_h, where $h \in [1, k]$. In the update stage, the algorithm computes the centers of the new clusters, denoted by $\mu_{c'_1}, \ldots, \mu_{c'_k}$. Finally, in the termination stage, the algorithm verifies whether the pre-defined termination condition holds. Specifically, the algorithm checks whether the sum of the squared Euclidean distances between the current and newly computed clusters is less than or equal to the threshold value β. If the termination condition holds, then the algorithm halts and returns the new cluster centers as the final output. Otherwise, the algorithm continues to the next iteration with the new clusters as input.

IV. The Proposed Transformations

It is important to note that cluster centers (denoted by μ_c for a cluster c) are represented as vectors and the entries in the vectors can be fractional values. Since the encryption schemes typically support integer values, we should somehow transform the entries of the cluster centers into integer values without affecting their utility in the k-means clustering process. Therefore, we first define scaling factors for clusters and then discuss a novel order-preserving Euclidean distance function operating over integers. Also, we discuss how to transform the termination condition in k-means clustering algorithm with fractional values into an integer-valued one.

Definition 3: Consider the cluster c_i whose center is denoted by μ_{c_i} (based on Definition 1). We know that μ_{c_i} is a vector and each entry can be a fractional value with denominator $|c_i|$, for $1 \leq i \leq k$. We define the scaling factor for c_i, denoted by α_i, as below:

$$\alpha_i = \frac{\prod_{j=1}^{k} |c_j|}{|c_i|} = \prod_{j=1 \wedge j \neq i}^{k} |c_j|. \tag{3}$$

Also, we define $\alpha = \prod_{j=1}^{k} |c_j|$ as the global scaling factor.

A. Order-Preserving Euclidean Distance

In the assignment stage of k-means clustering, the first step is to compute the Euclidean distance between a data record t_i and each cluster c_j, denoted by $\|t_i - c_j\| = \sqrt{\sum_{s=1}^{l} \left(t_i[s] - \frac{\lambda_{c_j}[s]}{|c_j|}\right)^2}$. It is clear that $\|t_i - c_j\|$ involves the fractional value $\frac{\lambda_{c_j}[s]}{|c_j|}$. In order to compute the encrypted value of $\|t_i - c_j\|$, we need to avoid such fractional values without affecting the relative ordering among the k Euclidean distances $\|t_i - c_1\|, \ldots, \|t_i - c_k\|$, where $c_1, \ldots, c_k$ denote k clusters. Note that since t_i has to be assigned to the nearest cluster, it is important to preserve the relative ordering among the computed k Euclidean distances. For this purpose, we propose a novel order-preserving Euclidean distance function whose evaluation involves only integer values.

We define the order-preserving Euclidean distance (OPED) function between a data record t_i and a cluster c_j as follows:

$$\text{OPED}(t_i, c_j) = \sqrt{\sum_{s=1}^{l} \left(\alpha * t_i[s] - \alpha_j * \lambda_{c_j}[s]\right)^2} \tag{4}$$

where α and α_j denote the global and c_j's scaling factors, respectively. Observe that all the terms in the above equation are integer values (here each attribute is explicitly assumed to take only integer values). Moreover, following Definition 3, we can rewrite the above equation as:

$$\begin{aligned} \text{OPED}(t_i, c_j) &= \sqrt{\sum_{s=1}^{l} \left(\alpha * t_i[s] - \frac{\alpha}{|c_j|} * \lambda_{c_j}[s]\right)^2} \\ &= \alpha * \|t_i - c_j\|. \end{aligned}$$

Since α remains constant for any given set of k clusters (in a particular iteration), we claim that the above OPED function preserves the relative ordering among cluster centers for any given data record. More specifically, given a data record t_i and two clusters c_j and $c_{j'}$, if $\|t_i - c_j\| \geq \|t_i - c_{j'}\|$, then it is guaranteed that OPED$(t_i, c_j) \geq$ OPED$(t_i, c_{j'})$, for $1 \leq j, j' \leq k$ and $j \neq j'$.

B. Termination Condition - Transformation

In the k-means clustering process (see Algorithm 1), the termination condition is given by:

$$\sum_{j=1}^{k} \|c_j - c'_j\|^2 \leq \beta \tag{5}$$

where $c_1, \ldots, c_k$ and $c'_1, \ldots, c'_k$ denote the current and new set of clusters in an iteration, respectively. Remember that $\|c_j - c'_j\| = \sqrt{\sum_{s=1}^{l} \left(\frac{\lambda_{c_j}[s]}{|c_j|} - \frac{\lambda_{c'_j}[s]}{|c'_j|}\right)^2}$ and clearly it consists of fractional values. In order to evaluate this condition over encrypted values, we first need to transform the above termination condition so that all the components are integers. To achieve this, we use the following approach. We define a constant scaling factor (denoted by f) for the termination condition in such a way that by multiplying both sides of Equation 5 with f^2, we can cancel all the denominator values. More specifically, we define the scaling factor for the termination condition as $f = \prod_{j=1}^{k} |c_j| * |c'_j|$. Also, we define the scaling factor for the cluster pair (c_j, c'_j) as $f_j = \frac{f}{|c_j|*|c'_j|} = \prod_{i=1 \wedge i \neq j}^{k} |c_i| * |c'_i|$. Then we define the new termination condition as follows:

$$\sum_{j=1}^{k}\sum_{s=1}^{l}\Big(|c'_j| * f_j * \lambda_{c_j}[s] - |c_j| * f_j * \lambda_{c'_j}[s]\Big)^2 \leq f^2 * \beta. \quad (6)$$

Observe that the above equation consists of only integer values. Now we need to show that evaluating the above equation is the same as evaluating Equation 5. First, we divide the above equation by f^2 on both sides of the inequality. Note that since f^2 remains constant in a given iteration, multiplication of both sides of Equation 6 by f^2 has no effect on the inequality. That is, Equation 6 can be rewritten as:

$$\sum_{j=1}^{k}\sum_{s=1}^{l}\frac{\Big(|c'_j| * f_j * \lambda_{c_j}[s] - |c_j| * f_j * \lambda_{c'_j}[s]\Big)^2}{f^2} \leq \beta.$$

The left-hand side of the above equation can be expanded as below:

$$\sum_{j=1}^{k}\sum_{s=1}^{l}\left(\frac{|c'_j| * f_j * \lambda_{c_j}[s]}{f} - \frac{|c_j| * f_j * \lambda_{c'_j}[s]}{f}\right)^2$$
$$= \sum_{j=1}^{k}\sum_{s=1}^{l}\left(\frac{|c'_j| * \lambda_{c_j}[s]}{|c_j| * |c'_j|} - \frac{|c_j| * \lambda_{c'_j}[s]}{|c_j| * |c'_j|}\right)^2$$
$$= \sum_{j=1}^{k}\|c_j - c'_j\|^2.$$

Based on the above discussions, it is clear that evaluating the inequality $\sum_{j=1}^{k}\|c_j - c'_j\|^2 \leq \beta$ is the same as evaluating Equation 6. Hence, in our proposed PPODC protocol, we consider Equation 6 as the termination condition of k-means clustering and evaluate it in a privacy-preserving manner.

V. The Proposed Solution

In this section, we first discuss a set of privacy-preserving primitives. Then, we present our novel PPODC protocol that utilizes the above transformation techniques and the privacy-preserving primitives as building blocks. As mentioned in Section I-A, in this paper we consider two semi-honest and non-colluding cloud service providers C_1 and C_2 under the Paillier cryptosystem [15]. Specifically, C_2 generates a pair of public-secret key pair (pk, sk) based on the Paillier's scheme such that sk is kept private whereas the corresponding public key pk is broadcasted. Let $E_{pk}(\cdot)$ and $D_{sk}(\cdot)$ denote the encryption and decryption functions under Paillier cryptosystem and N denote the RSA modulus. Note that, under Paillier's encryption scheme, $E_{pk}(a) * E_{pk}(b) \bmod N^2 = E_{pk}(a + b \bmod N)$, $\forall\, a, b \in \mathbb{Z}_N$.

A. *Privacy-Preserving Primitives*

We discuss a set of privacy-preserving primitives under the above two-party (i.e., C_1 and C_2) computation model.

- Secure Multiplication (SMP): Assume that C_1 holds $\langle E_{pk}(a), E_{pk}(b)\rangle$ and C_2 holds sk, where $\langle a, b\rangle$ is unknown to both C_1 and C_2, the goal of the SMP protocol is to compute $E_{pk}(a * b)$. During the execution of SMP, no information regarding the contents of a and b is revealed to C_1 and C_2.
- Secure Squared Euclidean Distance (SSED): In this protocol, C_1 holds two encrypted vectors $E_{pk}(X) = \langle E_{pk}(x[1], \ldots, E_{pk}(x[l])\rangle$ and $E_{pk}(Y) = \langle E_{pk}(y[1]), \ldots, E_{pk}(y[l])\rangle$. The goal of SSED is to compute the encryption of the squared Euclidean distance between X and Y. Specifically, the output is $E_{pk}((\|X - Y\|)^2)$. SSED reveals neither the contents of X and Y nor the Euclidean distance to C_1 and C_2.
- Secure Squared Order-Preserving Euclidean Distance (denoted by SSED$_{\text{OP}}$): Given that C_1 holds an encrypted data record, denoted by $E_{pk}(t_i)$, and an encrypted cluster, denoted by $E_{pk}(c_h)$, the goal of SSED$_{\text{OP}}$ is for C_1 and C_2 to jointly compute $E_{pk}((\text{OPED}(t_i, c_h))^2)$. Here $E_{pk}(c_h) = \langle E_{pk}(\lambda_{c_h}), E_{pk}(|c_h|)\rangle$ and $E_{pk}(\lambda_{c_h}) = \langle E_{pk}(\lambda_{c_h}[1]), \ldots, E_{pk}(\lambda_{c_h}[l])\rangle$. Note that OPED$(t_i, c_h)$ denotes the Euclidean distance between data record t_i and cluster c_h based on the order-preserving Euclidean distance function defined in Equation 4. The output is revealed only to C_1.
- Secure Bit-Decomposition (SBD): Suppose C_1 holds $E_{pk}(z)$, where z is unknown to both parties and $0 \leq z < 2^w$, the goal of SBD is to compute encryptions of the individual bits of z. The output is $[z] = \langle E_{pk}(z_1), \ldots, E_{pk}(z_w)\rangle$, where z_1 (resp., z_w) denotes the most (resp. least) significant bit of z. In SBD, no contents regarding z is revealed to C_1 and C_2.
- Secure Comparison (SC): Given that C_1 holds $\langle [a], [b]\rangle$, the goal of SC is to securely compare a and b. Here $[a]$ and $[b]$ denote the encrypted bit vectors of a and b, respectively. The output is γ, where $\gamma = 1$ if $a \leq b$, and 0 otherwise. At the end, γ is known to both C_1 and C_2.
- Secure Minimum out of k Numbers (SMIN$_k$): In this protocol, we assume that C_1 holds k encrypted bit vectors, denoted by $[d_1], \ldots, [d_k]$, corresponding to integers $d_1, \ldots, d_k$. The goal of SMIN$_k$ is to securely identify the array position corresponding to the minimum value among the k numbers. More specifically, if j^{th} integer is the minimum number among the k values, then the output of SMIN$_k$ is an encrypted vector Γ such that Γ_j is $E_{pk}(1)$ and all the other entries contain encryptions of 0, where $j \in [1, k]$. The SMIN$_k$ protocol should not reveal any information regarding the contents of k numbers (e.g., the minimum value or the array position corresponding to it, etc.) to C_1

Algorithm 2 $\text{SSED}_{\text{OP}}(E_{pk}(t_i), E_{pk}(c_h))$

```
Require: $C_1$ has $E_{pk}(t_i)$, $E_{pk}(c_h) = \langle E_{pk}(\lambda_{c_h}), E_{pk}(|c_h|)\rangle$
1: $C_1$ and $C_2$:
    (a). $b_h \leftarrow \text{SMP}_{k-1}(\tau_h)$, where $\tau_h = \cup_{j=1 \wedge j \neq h}^{k} E_{pk}(|c_j|)$
    (b). $b' \leftarrow \text{SMP}(b_h, E_{pk}(|c_h|))$
    (c). for $1 \leq s \leq l$ do:
        • $a_i[s] \leftarrow \text{SMP}(b', E_{pk}(t_i[s]))$
        • $a'_h[s] \leftarrow \text{SMP}(b_h, E_{pk}(\lambda_{c_h}[s]))$
    (d). $E_{pk}((\text{OPED}(t_i, c_h))^2) \leftarrow \text{SSED}(a_i, a'_h)$
```

and C_2.

Several solutions have been proposed for most of the above privacy-preserving primitives. Recently, Yousef et al. [21] discussed efficient implementations for SMP, SSED, and SBD. Also, an efficient solution to SC was proposed in [22]. We now propose implementations for SSED_{OP} and SMIN_k.

1) The SSED_{OP} Protocol: We discuss a novel solution to the SSED_{OP} problem using the SMP and SSED protocols as sub-routines. The main steps involved in the proposed SSED_{OP} protocol are highlighted in Algorithm 2. We assume that C_1 holds $\langle E_{pk}(c_1), \ldots, E_{pk}(c_k)\rangle$ and C_2 holds sk, where $c_1, \ldots, c_k$ denote k clusters and $E_{pk}(c_h) = \langle E_{pk}(\lambda_{c_h}), E_{pk}(|c_h|)\rangle$. Note that $E_{pk}(\lambda_{c_h}) = \langle E_{pk}(\lambda_{c_h}[1]), \ldots, E_{pk}(\lambda_{c_h}[l])\rangle$. The goal is to compute $E_{pk}((\text{OPED}(t_i, c_h))^2)$ for a given input $E_{pk}(t_i)$ and $E_{pk}(c_h)$, where $1 \leq h \leq k$.

To start with, C_1 and C_2 securely compute the scaling factor for cluster c_h (in encrypted format based on Equation 3) using the extended secure multiplication protocol, denoted by SMP_{k-1}, that takes $k-1$ encrypted inputs and multiplies them (within encryption). Specifically, they jointly compute $b_h = \text{SMP}_{k-1}(\tau_h)$, where $\tau_h = (E_{pk}(|c_j|))_{j\in[1,k]\wedge j\neq h}$. The important observation here is that $b_h = E_{pk}(\prod_{j=1\wedge j\neq h}^{k} |c_j|) = E_{pk}(\alpha_h)$, where α_h is the scaling factor for cluster c_h as defined in Equation 3. Then C_1 and C_2 securely multiply b_h with $E_{pk}(|c_h|)$ using SMP to get $b' = \text{SMP}(b_h, E_{pk}(c_h)) = E_{pk}(|c_1| * \ldots * |c_k|) = E_{pk}(\alpha)$, where α is the global scaling factor. After this, for $1 \leq s \leq l$, C_1 and C_2 jointly compute two encrypted vectors as follows:

$$a_i[s] = \text{SMP}(b', E_{pk}(t_i[s])) = E_{pk}(\alpha * t_i[s]),$$
$$a'_h[s] = \text{SMP}(b_h, E_{pk}(\lambda_{c_h}[s])) = E_{pk}(\alpha_h * \lambda_{c_h}[s]).$$

Finally, with the two encrypted vectors a_i and a'_h as C_1's input, C_1 and C_2 jointly compute the encrypted squared Euclidean distance between them using SSED. Specifically, the output of $\text{SSED}(a_i, a'_h)$ is $E_{pk}(\sum_{s=1}^{l}(\alpha * t_i[s] - \alpha_h * \lambda_{c_h}[s])^2)$. From Equation 4, it is clear that the output $\text{SSED}(a_i, a'_h)$ is equivalent to $E_{pk}((\text{OPED}(t_i, c_h))^2)$.

2) The SMIN_k Protocol: With respect to SMIN_k, the implementation discussed in [21] outputs the encryption of the minimum value (to C_1) among the k numbers as the output. On the other hand, we require an encrypted vector as the output such that the entry corresponding to the minimum value is an encryption of 1 and all the other entries contain encryptions of 0. For this purpose, we consider the implementation given in [23] which is actually an extension to the SMIN_k protocol proposed in [21]. More specifically, the SMIN_k protocol given in [23] computes the encryption of the array index corresponding to the minimum value. We emphasize that the protocol in [23] can be easily extended to output the desired encrypted vector as follows:

- Let $E_{pk}(I_{\min})$ be the output (known only to C_1) computed using the SMIN_k protocol given in [23], where $I_{\min}$ denotes the index corresponding to the minimum value among the k input values. Now C_1 computes $E_{pk}(i - I_{\min})$ and randomizes it to get $\phi[i] = E_{pk}(r_i * (i - I_{\min}))$, where r_i denotes a random number in $\mathbb{Z}_N$ and $1 \leq i \leq k$. Observe that exactly one of the entries in ϕ is equal to encryption of 0 (i.e., when $i = I_{\min}$) and the rest are encryptions of random values. Hereafter, $r \in_R \mathbb{Z}_N$ denotes a random number r in $\mathbb{Z}_N$.
- C_1 computes $u[\pi(i)] = \phi[i]$ and sends it to C_2, for $1 \leq i \leq k$. Here π is a random permutation function known only to C_1.
- Upon receiving u, C_2 decrypts it component-wise using sk to get $u'[i] = D_{sk}(u[i])$. After this, C_2 generates an encrypted vector U as follows. If $u'[i] = 0$, then $U[i] = E_{pk}(1)$, and $E_{pk}(0)$ otherwise. C_2 sends U to C_1.
- Finally, C_1 gets the desired encrypted vector as output by performing the inverse permutation on U, i.e., $\phi'[i] = U[\pi(i)]$, $1 \leq i \leq k$.

Example 3: Suppose that $k = 5$ and the input to SMIN_k is $\langle[3], [6], [13], [2], [9]\rangle$, where $[x]$ denotes the encrypted bit vector corresponding to integer x. The output of the SMIN_k protocol given in [23] is $E_{pk}(I_{\min}) = E_{pk}(4)$ and it will be known only to C_1. Note that since '2' is the minimum among the five input values, the output is the encryption of the array index corresponding to '2' in the input list (i.e., $I_{\min} = 4$). After this, C_1 computes $\phi[1] = E_{pk}(r_1 * (1-4))$, $\phi[2] = E_{pk}(r_2 * (2-4))$, $\phi[3] = E_{pk}(r_3 * (3-4))$, $\phi[4] = E_{pk}(r_4 * (4-4))$, and $\phi[5] = E_{pk}(r_5 * (5-4))$. Without loss of generality, let the random permutation function π (known only to C_1) be as follows. C_1 computes $u =$

$$\begin{array}{ccccccc} i & = & 1 & 2 & 3 & 4 & 5 \\ & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \pi(i) & = & 2 & 5 & 1 & 3 & 4 \end{array}$$

$\langle\phi[3], \phi[1], \phi[4], \phi[5], \phi[2]\rangle$ and sends u to C_2. Upon receiving, C_2 decrypts it using sk and identifies that $D_{sk}(u[3]) = 0$. Note that the rest of the values are random numbers. Then C_2 computes $U = \langle E_{pk}(0), E_{pk}(0), E_{pk}(1), E_{pk}(0), E_{pk}(0)\rangle$ and sends it to C_1. Finally, C_1 computes the final output as $\phi' = \langle E_{pk}(0), E_{pk}(0), E_{pk}(0), E_{pk}(1), E_{pk}(0)\rangle$. □

In the rest of this paper, SMIN_k refers to the implementation given in [23] combined with the above mentioned steps. Due to space limitations, we refer the reader to [21], [23] for more details.

B. The Proposed PPODC Protocol

In this sub-section, we discuss our proposed PPODC protocol which is based on the standard k-means algorithm discussed in Section III-B. As mentioned in Section I-A, our system model consists of n users denoted by $U_1, \ldots, U_n$. User U_j holds a database T_j of m_j data records with l attributes, for $1 \leq j \leq n$. Without loss of generality, let the aggregated database be $T = \bigcup_{j=1}^{n} T_j = \{t_1, \ldots, t_m\}$, where $m = \sum_{j=1}^{n} m_j$ denotes the total number of records in T. For simplicity, let $t_1 \ldots t_{m_1}$ belong to U_1, $t_{m_1+1}, \ldots, t_{m_1+m_2}$ belong to U_2, and so on. We assume that all users agree upon using two cloud service providers C_1 and C_2 for outsourcing their respective databases as well as the k-means clustering task. Remember that, in our system model, C_2 generates a public-secret key pair (pk, sk) based on the Paillier cryptosystem [15] and the public key pk is sent to all users and C_1.

After the users outsource their data (encrypted under pk) to C_1, the goal of PPODC is to enable C_1 and C_2 to jointly compute the global cluster centers using the aggregated encrypted data in a privacy-preserving manner. At a high level, our protocol computes the global cluster centers in an iterative manner until the pre-defined termination condition (given in Equation 6) holds.

The overall steps involved in the proposed PPODC protocol are given in Algorithms 3 and 4. The main steps are shown in Algorithm 3. Our protocol consists of the following three stages:

- **Stage 1 - Secure Data Outsourcing:**
  During this stage, each user U_j has to securely outsource an encrypted version of his/her database T_j to C_1. To minimize the data encryption costs of users, we achieve data outsourcing through randomization techniques. Note that this stage is run only once. At the end of this stage, only C_1 knows the (attribute-wise) encryptions of the n databases.
- **Stage 2 - Secure Computation of New Clusters:**
  In this stage, C_1 initially selects k data records at random (from the aggregated encrypted records) and assigns them as initial clusters (this step is the same as the initialization step in the traditional k-means algorithm). Then, C_1 and C_2 jointly assign each data record to a new cluster. After this, they compute the new cluster centers in encrypted format. The main goal of this stage is similar to the assignment and update stages given in Algorithm 1.
- **Stage 3 - Secure Termination or Update:**
  Upon computing the new cluster centers (in encrypted format), C_1 and C_2 securely verify whether the sum of the squared Euclidean distances between the current and new clusters is less than or equal to β (termination condition based on Equation 6). Here β denotes the pre-defined threshold value agreed upon by all the participating users. If the termination condition holds, then the protocol terminates returning the new cluster centers as the final output. Otherwise, C_1 and C_2 update the current clusters to the new clusters and repeat Stages 2 and 3.

We emphasize that Stage 1 of PPODC is executed only once whereas Stages 2 and 3 are run in an iterative manner. We now discuss the steps in each of these three stages in detail.

1) Stage 1 - Secure Data Outsourcing (SDO): Data are typically encrypted before being outsourced for privacy reasons. However, to avoid computation overhead on the users side due to having to encrypt their data, we consider the following approach for data outsourcing. User U_j generates two random shares for each attribute value of his/her data record t_i. Precisely, for the s^{th} attribute of data record t_i, U_j generates two random shares $(t_i^1[s], t_i^2[s])$ given by $t_i^1[s] = t_i[s] + r_i[s] \bmod N$ and $t_i^2[s] = N - r_i[s]$, where $r_i[s] \in_R \mathbb{Z}_N$ and $1 \le s \le l$. Observe that $t_i[s] = t_i^1[s] + t_i^2[s] \bmod N$. U_j outsources the random shares $t_i^1[s]$ and $t_i^2[s]$ to C_1 and C_2, respectively, instead of encrypting the database attribute-wise and outsourcing it to C_1. Thus, we are able to avoid heavy encryption costs on the users during the data outsourcing step. Here we assume that there exist secure communication channels, which can be established using standard mechanisms such as SSL, between user U_j and the two clouds C_1 and C_2. Each user U_j sends the random shares of his/her data to C_1 and C_2 separately through the secure communication channels.

Upon receiving the random shares of all data records, C_2 computes $E_{pk}(t_i^2[s])$ and sends it to C_1. Then, C_1 computes $E_{pk}(t_i[s]) = E_{pk}(t_i^1[s]) * E_{pk}(t_i^2[s])$, for $1 \le i \le m$ and $1 \le s \le l$.

Algorithm 3 PPODC$(\langle T_1, \ldots, T_n\rangle, \beta) \rightarrow (S_1, \ldots, S_n)$

```
Require: $U_j$ holds a private database $T_j$; $sk$ is known only to $C_2$
   {Stage 1 - Secure Data Outsourcing}
1: for $1 \le i \le m$:
     (a). for $1 \le s \le l$:
            • if $t_i \in T_j$ then:
                 ◦ $U_j$ computes $t_i^1[s] = t_i[s] + r_i[s] \bmod N$, $t_i^2[s] = N - r_i[s]$, and $r_i[s]$ is a random number in $\mathbb{Z}_N$; sends $t_i^1[s]$ to $C_1$ and $t_i^2[s]$ to $C_2$
            • $C_2$ sends $E_{pk}(t_i^2[s])$ to $C_1$
            • $C_1$: $E_{pk}(t_i[s]) \leftarrow E_{pk}(t_i^1[s]) * E_{pk}(t_i^2[s])$
   {Stage 2 - Secure Computation of New Clusters}
2: $C_1$:
     (a). Select $k$ records at random and assign them to initial clusters denoted by $E_{pk}(\lambda_{c_1}), \ldots, E_{pk}(\lambda_{c_k})$, where $c_1, \ldots, c_k$ denote the current clusters
     (b). $E_{pk}(|c_h|) \leftarrow E_{pk}(1)$, for $1 \le h \le k$
3: for $1 \le i \le m$ do:
     (a). $C_1$ and $C_2$:
            • $E_{pk}(d_i[h]) \leftarrow \mathrm{SSED}_{\mathrm{OP}}(E_{pk}(t_i), E_{pk}(c_h))$, for $1 \le h \le k$, where $E_{pk}(c_h) = \langle E_{pk}(\lambda_{c_h}), E_{pk}(|c_h|)\rangle$
            • $[d_i[h]] \leftarrow \mathrm{SBD}(E_{pk}(d_i[h]))$, for $1 \le h \le k$
            • $\Gamma_i \leftarrow \mathrm{SMIN}_k([d_i[1]], \ldots, [d_i[k]])$
            • $\Lambda_{i,h}[s] \leftarrow \mathrm{SMP}(\Gamma_{i,h}, E_{pk}(t_i[s]))$, for $1 \le h \le k$ and $1 \le s \le l$
4: $C_1$:
     (a). for $1 \le h \le k$ do:
            • $W_h[s] \leftarrow \prod_{i=1}^{m} \Lambda_{i,h}[s]$, for $1 \le s \le l$
            • $E_{pk}(|c'_h|) \leftarrow \prod_{i=1}^{m} \Gamma_{i,h}$
   {Stage 3 - Secure Termination or Update}
5: $\gamma \leftarrow \mathrm{SETC}(\Omega, \Omega')$; $\gamma$ denotes the evaluation result of termination condition, $\Omega = \{\langle E_{pk}(\lambda_{c_1}), E_{pk}(|c_1|)\rangle \ldots, \langle E_{pk}(\lambda_{c_k}), E_{pk}(|c_k|)\rangle\}$ and $\Omega' = \{\langle W_1, E_{pk}(|c'_1|)\rangle \ldots, \langle W_k, E_{pk}(|c'_k|)\rangle\}$
6: if $\gamma = 1$ then, for $1 \le h \le k$ and $1 \le s \le l$
     (a). $C_1$:
            • $O_h[s] \leftarrow W_h[s] * E_{pk}(r'_h[s])$ and $\delta_h \leftarrow E_{pk}(|c'_h|) * E_{pk}(r''_h)$, where $r'_h[s]$ and $r''_h \in_R \mathbb{Z}_N$
            • Send $O_h[s]$ and $\delta_h$ to $C_2$; $r'_h[s]$ and $r''_h$ to each user $U_j$
     (b). $C_2$: Send $O'_h[s] \leftarrow D_{sk}(O_h[s])$ and $\delta'_h \leftarrow D_{sk}(\delta_h)$ to each user $U_j$
   else for $1 \le h \le k$
            • $E_{pk}(\lambda_{c_h}) \leftarrow W_h$ and $E_{pk}(|c_h|) \leftarrow E_{pk}(|c'_h|)$
            • Go to Step 3
7: $U_j$, foreach received pair $\langle O'_h, r'_h\rangle$ and $\langle \delta'_h, r''_h\rangle$ do:
     (a). $\lambda_{c'_h}[s] = O'_h[s] - r'_h[s] \bmod N$, $1 \le s \le l$
     (b). $|c'_h| \leftarrow \delta'_h - r''_h \bmod N$
     (c). $\mu_{c'_h}[s] \leftarrow \frac{\lambda_{c'_h}[s]}{|c'_h|}$ and $S_j \leftarrow S_j \cup \{\mu_{c'_h}\}$
```

2) Stage 2 - Secure Computation of New Clusters: Given the (attribute-wise) encrypted versions of users data, during Stage 2 (denoted by SCNC), C_1 and C_2 jointly compute the new cluster centers in a privacy-preserving manner. To start with, C_1 randomly selects k encrypted data records (from the aggregated data) and assigns them as initial clusters. More specifically, the k encrypted data records are assigned to $E_{pk}(\lambda_{c_1}), \ldots, E_{pk}(\lambda_{c_k})$, respectively. For example, if the 3rd data record is selected as the first cluster c_1, then $E_{pk}(\lambda_{c_1}[s])$ is set to $E_{pk}(t_3[s])$, for $1 \leq s \leq l$. Also, $E_{pk}(|c_h|)$ is set to $E_{pk}(1)$ since each initial cluster c_h consists of only one data record, for $1 \leq h \leq k$.

For each encrypted data record $E_{pk}(t_i)$, C_1 and C_2 compute the squared Euclidean distance between t_i and all the clusters based on the order-preserving Euclidean distance function given in Equation 4. To achieve this, C_1 and C_2 jointly execute SSED_{OP} with $E_{pk}(t_i)$ and $E_{pk}(c_h)$ as C_1's private input, for $1 \leq i \leq m$ and $1 \leq h \leq k$, where $E_{pk}(c_h) = \langle E_{pk}(\lambda_{c_h}), E_{pk}(|c_h|)\rangle$. The output of SSED_{OP} is denoted by $E_{pk}(d_i[h])$, where $d_i[h] = (\text{OPED}(t_i, c_h))^2$. Now, C_1 and C_2 jointly execute the following set of operations:

- By using $E_{pk}(d_i[h])$ as C_1's private input to the secure bit-decomposition (SBD) sub-protocol, C_1 and C_2 securely compute encryptions of the individual bits of $d_i[h]$. The output $[d_i[h]] = \langle E_{pk}(d_{i,1}[h]), \ldots, E_{pk}(d_{i,w}[h])\rangle$ is known only to C_1, where $d_{i,1}[h]$ and $d_{i,w}[h]$ are the most and least significant bits of $d_i[h]$, respectively.
- For $1 \leq i \leq m$, with the k encrypted distances as C_1's private input to the secure minimum out of k numbers (SMIN_k) protocol, C_1 and C_2 compute an encrypted bit vector Γ_i. The important observation here is that $\Gamma_{i,g}$ is an encryption of 1 iff $d_i[g]$ is the minimum distance among $\langle d_i[1], \ldots, d_i[k]\rangle$. In this case, t_i is closest to cluster c_g, where $1 \leq g \leq k$. The rest of the values in Γ_i are encryptions of 0. Note that the output of SMIN_k, i.e., Γ_i, is known only to C_1.
- After this, C_1 and C_2 securely multiply $\Gamma_{i,h}$ with $E_{pk}(t_i[s])$ using the secure multiplication (SMP) sub-protocol. Precisely, C_1 and C_2 compute $\Lambda_{i,h}[s] = \text{SMP}(\Gamma_{i,h}, E_{pk}(t_i[s]))$. The observation here is that since $\Gamma_{i,g} = E_{pk}(1)$ only if t_i is closest to cluster c_g, $\Lambda_{i,g} = E_{pk}(t_i)$ denoting that t_i is assigned to new cluster c'_g. Also, $\Lambda_{i,h}$ is a vector of encryptions of 0, for $1 \leq h \leq k$ and $h \neq g$.

Next, C_1 computes the new cluster centers locally by performing homomorphic operations on $\Lambda_{i,h}$ and $\Gamma_{i,h}$ as follows:

- Compute (in encrypted format) the s^{th}-component of the numerator for the center of new cluster c'_h as $W_h[s] = \prod_{i=1}^{m} \Lambda_{i,h}[s]$, for $1 \leq h \leq k$ and $1 \leq s \leq l$. The observation here is that $W_h[s] = E_{pk}(\lambda_{c'_h}[s])$. Remember that $\mu_{c'_h}[s] = \frac{\lambda_{c'_h}[s]}{|c'_h|}$ denotes the center of cluster c'_h.
- Compute the encrypted number of data records that belong to the new cluster c'_h as $E_{pk}(|c'_h|) = \prod_{i=1}^{m} \Gamma_{i,h}$, for $1 \leq h \leq k$.

3) Stage 3 - Secure Termination or Update (STOU): Given the new clusters (in encrypted format) resulting from Stage 2, the goal of Stage 3 is for C_1 and C_2 to verify whether the termination condition (based on Equation 6) holds in a privacy-preserving manner. If the termination condition holds, the new cluster centers are returned as the final output to U_j. Otherwise, the entire iterative process (i.e., Stages 2 and 3) is repeated by using the new clusters as the current clusters. The current and new clusters are $\Omega = \{\langle E_{pk}(\lambda_{c_1}), E_{pk}(|c_1|)\rangle, \ldots, \langle E_{pk}(\lambda_{c_k}), E_{pk}(|c_k|)\rangle\}$ and $\Omega' = \{\langle W_1, E_{pk}(|c'_1|)\rangle \ldots, \langle W_k, E_{pk}(|c'_k|)\rangle\}$), respectively.

First, by using the current and new clusters, C_1 and C_2 need to securely evaluate the termination condition (SETC) based on Equation 6. The main steps involved in SETC are given in Algorithm 4 which we explain in detail below:

- To start with, C_1 and C_2 compute $\tau_i = E_{pk}(|c_i| * |c'_i|)$ using $\langle E_{pk}(|c_i|), E_{pk}(|c'_i|)\rangle$ as C_1's private input to the SMP protocol, for $1 \leq i \leq k$. The output τ_i is known only to C_1.
- By using τ_i's, they compute $V_i = \text{SMP}_{k-1}(\tau'_i)$, where $\tau'_i = (\tau_j)_{j \in [1,k] \wedge j \neq i}$. Here SMP_{k-1} denotes the SMP protocol with $k-1$ encrypted inputs that need to be securely multiplied. More specifically, $V_i = E_{pk}(\prod_{j=1 \wedge j \neq i}^{k} |c_i| * |c'_i|)$, for $1 \leq i \leq k$. The important observation here is
$$V_i = E_{pk}\left(\prod_{j=1 \wedge j \neq i}^{k} |c_i| * |c'_i|\right) = E_{pk}(f_i)$$
where f_i is the scaling factor for cluster pair (c_i, c'_i) defined in Section IV-B. Then, they compute an encrypted value $Z_i = \text{SMP}(V_i, V_i) = E_{pk}(f_i^2)$.
- After this, they securely multiply V_1 and τ_1 using SMP protocol. The output of this step is
$$V = \text{SMP}(V_1, \tau_1) = E_{pk}\left(\prod_{j=1}^{k} |c_j| * |c'_j|\right) = E_{pk}(f)$$
, where f is the scaling factor for the termination condition as defined in Section IV-B. They compute $Y = \text{SMP}(V, V) = E_{pk}(f^2)$.
- For $1 \leq i \leq k$, C_1 and C_2 securely multiply each component in the current and new clusters with $|c'_i|$ and $|c_i|$, respectively. Specifically, for $1 \leq i \leq k$ and $1 \leq s \leq l$, they compute
$$G_i[s] = \text{SMP}(E_{pk}(\lambda_{c_i}[s]), E_{pk}(|c'_i|)) = E_{pk}(\lambda_{c_i}[s] * |c'_i|),$$
$$G'_i[s] = \text{SMP}(W_i[s], E_{pk}(|c_i|)) = E_{pk}(\lambda_{c'_i}[s] * |c_i|).$$
Note that $W_i[s]$ computed in Stage 2 equals $E_{pk}(\lambda_{c'_i}[s])$.
- Now, by using the secure squared Euclidean distance (SSED) protocol with input vectors G_i and G'_i, C_1 and C_2 jointly compute $H_i = \text{SSED}(G_i, G'_i)$. Precisely, they compute the encryption of squared Euclidean distance between vectors in G_i and G'_i given by
$$H_i = E_{pk}\left(\sum_{s=1}^{l} (\lambda_{c_i}[s] * |c'_i| - \lambda_{c'_i}[s] * |c_i|)^2\right).$$
- Given Z_i and H_i, C_1 and C_2 can securely multiply

them using SMP to get

$$H'_i = \text{SMP}(H_i, Z_i)$$
$$= E_{pk}\left(f_i^2 * \sum_{s=1}^{l} (\lambda_{c_i}[s] * |c'_i| - \lambda_{c'_i}[s] * |c_i|)^2\right).$$

At the end of the above process, C_1 has $Y = E_{pk}(f^2)$ and H'_i, for $1 \leq i \leq k$. Now C_1 locally computes:

$$\beta' = Y^\beta = E_{pk}(f^2 * \beta) \quad \text{and}$$
$$M = \prod_{i=1}^{k} H'_i$$
$$= E_{pk}\left(\sum_{i=1}^{k}\sum_{s=1}^{l} (\lambda_{c_i}[s] * f_i * |c'_i| - \lambda_{c'_i}[s] * f_i * |c_i|)^2\right).$$

At this point, C_1 has encryptions of the integers corresponding to both the left-hand and right-hand sides of the termination condition given in Equation 6. Therefore, the goal is to now securely compare them using the secure comparison (SC) protocol. However, the existing SC protocols (e.g., [22]) require encryptions of individual bits of the integers to be compared rather than the encrypted integers itself. Hence, we need to first convert the encrypted integers of the left-hand and right-hand sides into their respective encrypted bit representations using the secure bit-decomposition (SBD) protocol. Given β' and M, they can be securely compared as follows:

- C_1 and C_2 convert β' and M into their encrypted bit vectors using the SBD sub-protocol. Specifically, they compute $L = \text{SBD}(M) = [\sum_{i=1}^{k}\sum_{s=1}^{l}(\lambda_{c_i}[s] * f_i * |c'_i| - \lambda_{c'_i}[s] * f_i * |c_i|)^2)]$ and $R = \text{SBD}(\beta') = [f^2 * \beta]$. Note that $[x]$ denotes the encrypted bit vector of an integer x. Also, remember that the outputs of SBD, i.e., L and R, are known only to C_1.
- By using L and R as C_1's private input to the SC protocol, C_1 and C_2 securely evaluate the termination condition:

$$\sum_{i=1}^{k}\sum_{s=1}^{l}\left(\lambda_{c_i}[s] * f_i * |c'_i| - \lambda_{c'_i}[s] * f_i * |c_i|\right)^2 \leq f^2 * \beta.$$

  The output $\gamma = \text{SC}(L, R) = 1$ if the termination condition holds, and $\gamma = 0$ otherwise (here γ is known to C_1 and C_2).

Finally, once the termination condition has been securely evaluated, C_1 locally proceeds as follows:

- If $\gamma = 1$ (i.e., when the termination condition holds), the newly computed clusters are the final clusters which need to be sent to each user U_j. For this purpose, C_1 takes the help of C_2 to obliviously decrypt the results related to the new cluster centers. More specifically, C_1 initially picks random numbers $\langle r'_h[1], \ldots, r'_h[l], r''_h\rangle$ and computes $O_h[s] = W_h[s] * E_{pk}(r'_h[s]) = E_{pk}(\lambda_{c'_h}[s] + r'_h[s] \bmod N)$ and $\delta_h = E_{pk}(|c'_h|) * E_{pk}(r''_h) = E_{pk}(|c'_h| + r''_h \bmod N)$, for $1 \leq h \leq k$ and $1 \leq s \leq l$. After this, C_1 sends $O_h[s]$ and δ_h to C_2. In addition, C_1 sends $r'_h[s]$ and r''_h to each user U_j (through separate and secure communication channels).
- For $1 \leq h \leq k$, $1 \leq s \leq l$, C_2 successfully decrypts the received encrypted values using his/her secret share sk to get $O'_h[s] = D_{sk}(O_h[s])$ and $\delta'_h = D_{sk}(\delta_h)$ which it forwards to each user U_j (via secure communication channels). Observe that, due to the randomization by C_1, the values of $O'_h[s]$ and δ'_h are random numbers in $\mathbb{Z}_N$ from C_2's perspective.
- Upon receiving the entry pairs $\langle O'_h, r'_h\rangle$ and $\langle \delta'_h, r''_h\rangle$, user U_j removes the random factors to get $\lambda_{c'_h}[s] = O'_h[s] - r'_h[s] \bmod N$ and $|c'_h| = \delta'_h - r''_h \bmod N$, for $1 \leq h \leq k$ and $1 \leq s \leq l$. Finally, U_j computes the final cluster center as $\mu_{c'_h}[s] = \frac{\lambda_{c'_h}[s]}{|c'_h|}$ and adds it to his/her resulting set S_j.
- Otherwise, when $\gamma = 0$, C_1 locally updates the current clusters to new clusters by setting $E_{pk}(\lambda_{c_h}) = W_h$ and $E_{pk}(|c_h|) = E_{pk}(|c'_h|)$, for $1 \leq h \leq k$. Then, the above process is iteratively repeated until the termination condition holds, i.e., the protocol goes to Step 3 of Algorithm 3 and executes Steps 3 to 6 with the updated cluster centers as input.

Algorithm 4 SETC(Ω, Ω')

```
Require: $C_1$ has $\Omega = \{\langle E_{pk}(\lambda_{c_1}), E_{pk}(|c_1|)\rangle, \ldots, \langle E_{pk}(\lambda_{c_k}), E_{pk}(|c_k|)\rangle\}$, $\Omega' = \{\langle W_1, E_{pk}(|c'_1|)\rangle \ldots, \langle W_k, E_{pk}(|c'_k|)\rangle\}$
1: $C_1$ and $C_2$:
     (a). $\tau_i \leftarrow$ SMP$(E_{pk}(|c_i|), E_{pk}(|c'_i|))$, for $1 \leq i \leq k$
     (b). for $1 \leq i \leq k$ do:
          • $V_i \leftarrow$ SMP$_{k-1}(\tau'_i)$, where $\tau'_i = \cup_{j=1 \wedge j \neq i}^{k} \tau_j$
          • $Z_i \leftarrow$ SMP$(V_i, V_i)$
     (c). $V \leftarrow$ SMP$(V_1, \tau_1)$ and $Y \leftarrow$ SMP$(V, V)$
     (d). for $1 \leq i \leq k$ and $1 \leq s \leq l$ do:
          • $G_i[s] \leftarrow$ SMP$(E_{pk}(\lambda_{c_i}[s]), E_{pk}(|c'_i|))$
          • $G'_i[s] \leftarrow$ SMP$(W_i[s], E_{pk}(|c_i|))$
     (e). $H_i \leftarrow$ SSED$(G_i, G'_i)$, for $1 \leq i \leq k$
     (f). $H'_i \leftarrow$ SMP$(H_i, Z_i)$, for $1 \leq i \leq k$
2: $C_1$: $\beta' \leftarrow Y^\beta$ and $M \leftarrow \prod_{i=1}^{k} H'_i$
3: $C_1$ and $C_2$:
     (a). $L \leftarrow$ SBD$(M)$ and $R \leftarrow$ SBD$(\beta')$
     (b). $\gamma \leftarrow$ SC$(L, R)$
```

C. Security Analysis

In this section, we show that the proposed PPODC protocol is secure under the standard semi-honest model [14], [24]. Informally speaking, we stress that all the intermediate values seen by C_1 and C_2 in PPODC are either encrypted or pseudo-random numbers.

First, in the data outsourcing process (i.e., Step 1 of Algorithm 3), the values received by C_1 and C_2 are either random or pseudo-random values in $\mathbb{Z}_N$. At the end of the data outsourcing step, only C_1 knows the encrypted data records of all users and no information regarding the contents of T_j (the database of user U_j) is revealed to C_2. Therefore, as long as the underlying encryption scheme is semantically secure (which is the case in the Paillier cryptosystem [17]), the aggregated encrypted databases do not reveal any information to C_1. Hence, no information is revealed to C_1 and C_2 during Stage 1 of PPODC.

The implementations of SMP, SSED, SBD, and SMIN$_k$ sub-protocols given in [23], [25] are proven to be secure under the semi-honest model [14]. Also, the SC protocol given in [22] is secure under the semi-honest model. In the proposed SSED$_{\text{OP}}$ protocol, the computations are based on using either SMP or SSED as a sub-routine. As a result, SSED$_{\text{OP}}$ can be proven to be secure under the semi-honest model. Also, since SMIN$_k$ is directly constructed from the solution given in [23], it can be proven secure under the semi-honest model. In short, the privacy-preserving primitives utilized in our protocol are secure under the semi-honest model.

We emphasize that the computations involved in Stages 2 and 3 of PPODC are performed by either C_1 locally or using one of the privacy-preserving primitives as a sub-routine. In the former case, C_1 operates on encrypted data locally. In the latter case, the privacy-preserving primitives utilized in our protocol are secure under the semi-honest model. Also, it is important to note that the output of a privacy-preserving primitive which is given as input to the next primitive is encrypted. Since we use a semantically secure Paillier encryption scheme [15], all the encrypted results (which are revealed only to C_1) from the privacy-preserving primitives do not reveal any information to C_1. Note that the secret key sk is unknown to C_1. By Composition Theorem [24], we claim that the sequential composition of the privacy-preserving primitives invoked in Stages 2 and 3 of our proposed PPODC protocol is secure under the semi-honest model. Putting everything together, it is clear that PPODC is secure under the semi-honest model.

D. Complexity Analysis

In this subsection, we theoretically analyze the computational and communication costs incurred in each stage of the proposed PPODC protocol. The results regarding computational costs are given in Table I. Here m denotes the sum of the numbers of data records of all users, l denotes the number of attributes, k denotes the number of clusters, $w_1 = 2k\log_2(m/k) + \log_2 l + 2\log_2(ub)$ is the maximum bit-length to express the order-preserving distance, and $w_2 = 4k\log_2(m/k) + \log_2 k + \log_2 l + 2\log_2(ub)$ is the maximum bit-length to express the left-hand side of Equation 6, where ub represents the maximum possible attribute value. It is important to note that Stage 1 of PPODC is run only once whereas Stage 2 and Stage 3 are run in an iterative manner until the termination condition holds.

In addition, the total communication costs for each stage of PPODC are analyzed and the results are shown in Table II. Here K denotes the size (in bits) of Paillier encryption key [15]. Following from our results, we can observe that the costs of Stage 2 are significantly higher than the costs incurred in Stage 3 in each iteration.

TABLE II. COMMUNICATION COSTS FOR THE PROPOSED PPODC PROTOCOL

Stage	Communication Costs (in bits)
Stage 1	$4m * l * K$
Stage 2	$(2m * (6l * k + 5w_1 * k - 3(w_1 + 1)) + 20k) * K$
Stage 3	$(3k^2 + 9k * l + 5w_2 + 3k + 3) * 2K$

E. Boosting the Performance of PPODC

We emphasize that a direct implementation of the PPODC protocol is likely to be inefficient due to involved expensive cryptographic operations. To address this issue, we propose two strategies to boost its performance: *(i) offline computation* and *(ii) pipelined execution*. In what follows, we extensively discuss how these two strategies are applicable to improving the performance of PPODC.

1) Offline Computation: In the Paillier cryptosystem [15], encryption of an integer $a \in \mathbb{Z}_N$ is given by $E_{pk}(a) = g^a * r^N \bmod N^2$, where g is the generator, N is the RSA modulus, and r is a random number in $\mathbb{Z}_N$. It is clear that Paillier's encryption scheme requires two expensive exponentiation operations. In this paper, we assume $g = N + 1$ (a commonly used setting that provides the same security guarantee as the original Paillier cryptosystem) as this allows for a more efficient implementation of Paillier encryption [26]. More specifically, when $g = N + 1$, we have $E_{pk}(a) = (N+1)^a * r^N \bmod N^2 = (a*N+1)*r^N \bmod N^2$. As a result, an encryption under Paillier is reduced to one exponentiation operation. Our main observation from the above derivation is that the encryption cost under Paillier can be further reduced as follows. The exponentiation operation (i.e., $r^N \bmod N^2$) in the encryption function can be computed in an offline phase and thus the online cost of computing $E_{pk}(a)$ is reduced to two (inexpensive) multiplication operations[2]. Additionally, the encryption of random numbers, 0s and 1s is independent of the underlying data and thus can be precomputed by the corresponding party (i.e., C_1 or C_2).

We emphasize that the actual online computation costs (with an offline phase) of the privacy-preserving primitives used in our protocol can be much less than their costs without an offline phase. For example, consider the secure multiplication (SMP) primitive with $E_{pk}(a)$ and $E_{pk}(b)$ as C_1's private input. During the execution of SMP, C_1 has to initially randomize the inputs and send them to C_2. That is, C_1 has to compute $E_{pk}(a) * E_{pk}(r_1) = E_{pk}(a + r_1 \bmod N)$ and $E_{pk}(b) * E_{pk}(r_2) = E_{pk}(b + r_2 \bmod N)$, where r_1 and r_2 are random numbers in $\mathbb{Z}_N$. This clearly requires C_1 to compute two encryptions: $E_{pk}(r_1)$ and $E_{pk}(r_2)$. However, since r_1 and r_2 are integers chosen by C_1 at random, the computation of $E_{pk}(r_1)$ and $E_{pk}(r_2)$ is independent of any specific instantiation of SMP. That is, C_1 can precompute $E_{pk}(r_1)$ and $E_{pk}(r_2)$ during the offline phase, thus boosting its online computation time. In a similar manner, C_1 and C_2 can precompute certain intermediate results in each privacy-preserving primitive.

2) Pipelined Execution: We are able to further reduce the online execution time by adopting the technique of pipelined execution. Take the execution of SMP for example, by which C_1 would like to compute $E_{pk}(a_1 * b_1)$ and $E_{pk}(a_2 * b_2)$ given $\langle E_{pk}(a_1), E_{pk}(b_1)\rangle$ and $\langle E_{pk}(a_2), E_{pk}(b_2)\rangle$, respectively. Here C_1 does not have to wait for C_2's response after sending $E_{pk}(a_1+r_{11})$ and $E_{pk}(b_1+r_{12})$. Instead, after sending $E_{pk}(a_1 + r_{11})$ and $E_{pk}(b_1 + r_{12})$ to C_2, C_1 immediately computes $E_{pk}(a_2 + r_{21})$ and $E_{pk}(b_2 + r_{22})$. By observing that the time needed for C_2 to process $E_{pk}(a_1 + r_{11})$ and $E_{pk}(b_1 + r_{12})$ is approximately the same as the time needed for C_1 to process the intermediate result returned from C_2 afterward[3], we expect that we could further save at least half of the online execution time in the long run when we have a lot of SMP operations to perform.

Similarly, suppose that C_1 is given $E_{pk}(z_1)$ and $E_{pk}(z_2)$ and would like to invoke SBD to derive $\langle E_{pk}(z_{1,1}), \ldots, E_{pk}(z_{1,w})\rangle$ and $\langle E_{pk}(z_{2,1}), \ldots, E_{pk}(z_{2,w})\rangle$, where $z_{i,1}$ ($z_{i,w}$) denotes the most (resp. least) significant bit of z_i. We observe that C_1 does not have to sequentially

[2]The time that takes to perform one exponentiation under $\mathbb{Z}_{N^2}$ is equivalent to $\log_2 N$ multiplication operations. Therefore, multiplication operation is inexpensive compared to exponentiation.

[3]There will be two exponentiations for both C_1 and C_2.

TABLE I. COMPUTATIONAL COSTS FOR DIFFERENT STAGES IN THE PROPOSED PPODC PROTOCOL

Stage	**Computational Costs**
Stage 1	$5m * l$ multiplications
Stage 2 (per iteration)	$m * (k * (11l + 13w_1 + 10) - 10w_1 - 2)$ exponentiations $m * (k * (23l + 16w_1 + 1) - 16w_1 - 3) - k(l + 1)$ multiplications
Stage 3 (per iteration)	$k * (16l + 15) + 16w_2 + 1$ exponentiations $k * (32l + 30) + 17w_2 - 1$ multiplications

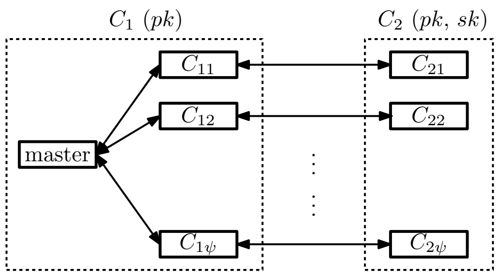


Fig. 2. A Cluster-Based Implementation Model

compute $E_{pk}(z_{1,w})$, …, $E_{pk}(z_{1,1})$, followed by the computation of $E_{pk}(z_{2,w})$, …, $E_{pk}(z_{2,1})$[4]. Like what we have seen in the execution of SMP, to generate $E_{pk}(z_{1,w})$ from $E_{pk}(z_1)$, C_1 first computes $E_{pk}(z_1 + r_1)$ and sends it to C_2, followed by one exponentiation and some processing by C_2. The intermediate result generated by C_2 is then sent back to C_1 to derive $E_{pk}(z_{1,w})$ and the necessary information to derive $E_{pk}(z_{1,(w-1)})$ in the next iteration. By noticing that the time for C_2 to process $E_{pk}(z_1 + r_1)$ and the time for C_1 to process the intermediate result (from C_2) is approximately the same, i.e., one exponentiation on each party, C_1 could prepare $E_{pk}(z_2 + r_2)$ immediately without waiting for the intermediate result from C_2 regarding $E_{pk}(z_1+r_1)$. We expect to save around half of the online execution time by applying this technique to the SBD protocol.

VI. EXPERIMENTAL RESULTS

First of all, we emphasize that PPODC is 100% accurate in the sense that the outputs returned by PPODC and the standard k-means clustering algorithm (applied on the corresponding plaintext data) are the same. Therefore, in this section, we extensively analyze the running time of PPODC by performing various experiments using a real dataset under different parameter settings. Note that ours is the first work to address the PPODC problem requiring a federated cloud consisting of only two service providers and thus there exists no prior work to compare with our protocol.

A. *Implementation and Dataset Description*

We implemented the protocols in C using the GNU Multiple Precision Arithmetic (GMP) library [27]. The experiments were conducted on a local cluster consisting of 16 computers [5], each with an Intel® Xeon™ CPU E5-5320 at 1.86GHz [6] and 8 GBytes of memory, running Linux version 3.17.7.

[4]Recall that the SBD protocol securely extracts the encrypted bit vectors of z_1 and z_2 starting from the least significant bit to the most significant bit [25].

[5]We emphasize that this is not a dedicated cluster for our own computing tasks. It is a public cluster shared among users in the Computer Science department at Purdue University.

[6]We note that this is not a top-notch CPU. According to http://www.cpubenchmark.net/, it is at least 5 times slower than Intel® Core™ i7-4790K, a popular CPU widely used nowadays.

TABLE III. COMPUTATION TIME OF PRIVACY-PRESERVING PRIMITIVES WHEN m = 6,000, l = 10, k = 4, AND n = 1 (IN MILLISECONDS)

Primitive	Online + Offline	Online	Pipelined Online
SSED_{OP}	1,186	356	177
SBD	3,372	1,333	585
SMIN_k	33,486	16,021	11,897
SMP	59	23	11

Figure 2 depicts our implementation model and shows how the cluster servers communicate with each other when executing the protocols over TCP/IP. The system consists of three components: 1) the master node, 2) a number (ψ) of servers performing the tasks of C_1, and 3) the same number of servers performing the tasks of C_2. Since we have 16 servers, we could have up to 8 pairs of servers performing the tasks needed in PPODC. The master node is directly connected with those servers of C_1, whereas each of C_1's server is paired with a corresponding server of C_2. The master node is in charge of the coordination of the execution of tasks needed in PPODC, i.e., within each iteration of k-means clustering, the master node needs to instruct each pair of servers about which task in an iteration to perform. When the task assigned to a pair of servers is complete, this pair will contact the master node for further instruction. Depending on the task just been accomplished, the master node would either assign the next task to this pair of servers, or put them to wait for others to finish.

For our experiments, we used the KEGG Metabolic Reaction Network (Undirected) dataset from the UCI KDD archive [28] that consists of 65,554 data records and 29 attributes. Since some of the attribute values are missing in the dataset, we removed the corresponding data records and the resulting dataset consists of 64,608 data records. As part of the preprocessing, we normalized the attribute values and scaled them into the integer domain $[0, 1000]$. Then we selected sample datasets (from the preprocessed data) by choosing data records at random based on the parameter values under consideration. We fixed the Paillier encryption key size to 1,024 bits (a commonly accepted key size) in all our experiments. For each sample dataset, we share each of its data record attribute-wise among the servers of C_1 and C_2 as mentioned in Stage 1 of PPODC. Note that the secret key sk is stored on C_2's servers.

B. *Performance of PPODC*

We evaluate the performance of our protocol based on the following parameters: the number of data records (m), the number of attributes in each data record (l), the number of clusters (k), and the number of pairs of servers (denoted by ψ) deployed when executing PPODC. On our local cluster, ψ varies from 1 to 8 as described previously. Also, we analyze the performance of PPODC based on different modes of execution: (i) the basic implementation without any optimization (denoted by *Online + Offline*), (ii) the implementation moving the computation of random ciphertexts to the offline phase (denoted by *Online*), (iii) the implementation that adopts the pipelined execution for SMP and SBD protocol assuming that all needed random ciphertexts are computed offline (denoted by *Pipelined Online*). We note that SMP is intensively used as a sub-protocol in both SSED_{OP} and SMIN_k. Therefore, to further reduce the online computation time, in the third execution mode of PPODC, we replace each invocation of the SMP protocol with its pipelined version. In the following experiments, unless otherwise specified, we use $m = 6{,}000$, $l = 10$, $k = 4$, $\psi = 8$ and pipelined execution as the default parameters to execute PPODC.

We first measure the running time of each secure primitive used in our protocol under different execution modes. To have a more accurate result, we execute each primitive on a single pair of servers for 100 times and compute the average. The results are shown in Table III. From the results it can be seen that SSED_{OP}, SBD, and SMIN_k are much more expensive than SMP. This is as expected because the execution time needed by the first three primitives depends on m, l, and k as well. Precisely, SSED_{OP} depends on l, whereas SBD and SMIN_k are directly related to all those three parameters. Recall that the squared Order-Preserving Euclidean Distance (OPED) between a record t_i and a cluster c_j could be expressed as $\alpha^2 * \|t_i - c_j\|^2$, where $\alpha = \prod_{j=1}^{k} |c_j|$, the product of cardinalities of all clusters. It can be seen that $\text{OPED}(t_i, c_j)^2 \leq (m/k)^{2k} * (ub^2) * l$, where ub is the upperbound of attribute values for each dimension and is set to 1,000 in our experiment. In order for SMIN_k to correctly output a list of ciphertexts indicating the closest cluster for a given data record t_i, SBD should be invoked in a way that all the possible nonzero bits of $\text{OPED}(t_i, c_j)^2$ are preserved. For $m = 6{,}000$, $l = 10$, $k = 4$, we thus need to securely decompose $E_{pk}(\text{OPED}(t_i, c_j)^2)$ into $\langle E_{pk}(z_1), \ldots, E_{pk}(z_{108})\rangle$, where z_1 (resp., z_{108}) represents the most (resp., least) significant bit of $E_{pk}(\text{OPED}(t_i, c_j)^2)$. In Table III, we can also observe that by moving the generation of random ciphertexts to the offline phase, we are able to save at least half of the online computation time for all 4 primitives. Furthermore, we see the effectiveness after pipelining the execution of SBD and SMP. Specifically, for SSED_{OP}, SBD, and SMP, we can save at least 81% of the online execution time if both those two optimizations are employed. For SMIN_k, we are able to save 64% of the online execution time as well.

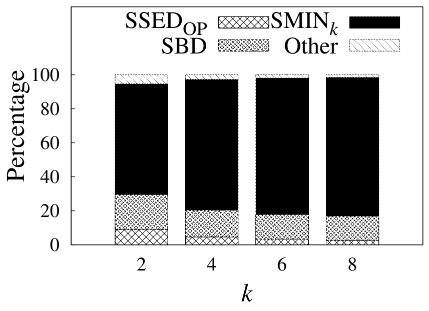


Fig. 4. Cost Breakdown when $m = 6{,}000$, $l = 10$, $\psi = 8$ under pipelined execution of SMP and SBD

Next, we break down the total execution time required by each sub-protocol used in PPODC. The results are presented in Figure 4. In Figure 4, we categorize the cost of PPODC into 4 parts, each denoting the time spent in each part respectively. We note that the part represented by 'Other' includes the time used by SMP, aggregation of cluster information, as well as the secure evaluation of termination condition (SETC). From Figure 4 it can also be observed that when $m = 6{,}000, l = 10$, and $\psi = 8$, the percentage of the time consumed by SBD and SMIN_k grows for increasing k values. For instance, when $k = 2$, SBD and SMIN_k combined take 85% of the execution time, while they take 95% of the execution time when $k = 8$. The increased percentage of cost by SBD and SMIN_k comes from the fact that the time needed by these two protocols depends not only on the bit-length to express the order-preserving distances but also on the number of clusters. Precisely, when m and l are fixed, the bit-length to express the order-preserving distances is linear in $2k \cdot \log_2(m/k)$. Also, the number of encrypted distances to be bit-decomposed and then compared (by SMIN_k) also grows linearly in k. Hence, it can be seen that the time needed by SBD and SMIN_k grows quadratically in k, while the time needed in other parts (SSED_{OP} and SMP) mostly grows linearly in k.

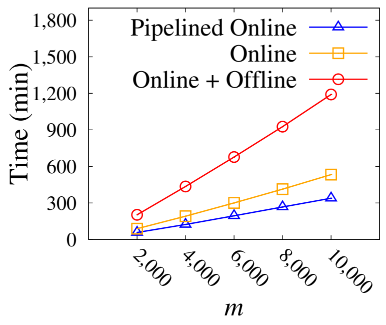
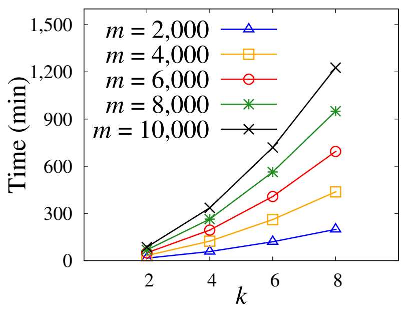


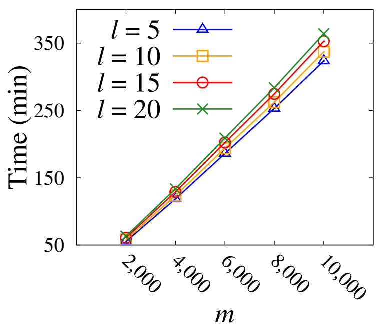
Fig. 5. m vs. execution mode for $l = 10, k = 4$, and $\psi = 8$

We also assess the execution time of PPODC for varying values of parameters m, l, and k. The results are given in Figure 3. In Figure 3(a), it can be seen that the online execution time of PPODC grows for increasing values of m and k. It is also obvious that the execution time is more sensitive to k than to m or l, which is expected because SMIN_k and SBD contribute to at least 85% of the execution time of PPODC and both of them have time complexity quadratic in k. In general, the total execution time of PPODC grows almost linear with the number of data records. Similar behavior is seen in Figures 3(b) and 3(c). The total execution time is approximately linear in m but the variation in l only slightly affects the execution time.

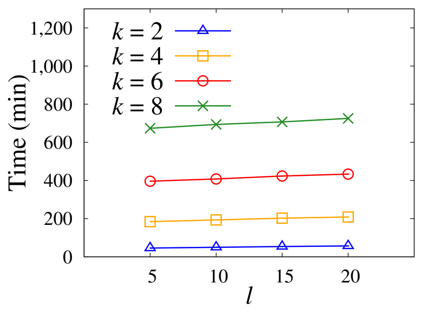
We report the running time needed by PPODC under different execution modes for varying values of m in Figure 5. The execution time grows approximately linear with m. In addition, from Table III, by moving the computation of random ciphertexts to the offline phase, it is clear that we save around 55% of the online execution time. If we further pipeline the execution of all invocations to SMP and SBD protocols, we save another 16% of the execution time compared to the basic implementation without any optimization. More specifically, it would take up to 1,190 minutes to perform one single iteration of PPODC ($m = 10{,}000$, $l = 10$, $k = 4$, and $\psi = 8$) if none of the optimizations is adopted, whereas only 337 minutes are needed when both optimizations are applied, a saving of 71% with respect to the online execution time. We further note that the average execution time is 293 minutes for each of C_1's server (excluding waiting time) and 183 minutes for each of



(a) Running time for $l = 10$



(b) Running time for $k = 4$



(c) Running time for $m = 6{,}000$

Fig. 3. Pipelined online computation costs of PPODC for encryption key size 1,024 bits, $\psi = 8$, and varying values of l, k, and m

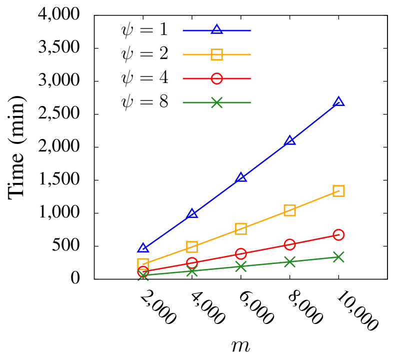


Fig. 6. m vs. number of server pairs for $l = 10$ and $k = 4$ under pipelined execution of SMP and SBD

C_2's server. The total execution time (337 minutes) needed is less than the sum of their respective execution time due to the pipelined execution of all instances of SMP and SBD protocols (including those invoked in SSED_{OP} and SMIN_k). To be specific, on average, 87% of the time is spent on computation for each of C_1's servers, whereas only 54% of the time is spent on computation for each of C_2's servers.

Lastly, we assess the effectiveness of parallel execution of the secure primitives involved in Stage 2 of our protocol. Recall that in our protocol we can almost achieve record level parallelism. To be precise, before the execution of PPODC, the master node will evenly distribute the n data records evenly to those ψ pairs of servers. Hence, when performing the Step 3 in PPODC, a pair of servers does not have to wait for other pairs of servers since a data record is assigned to exactly one pair of servers for processing. We thus expect the parallel execution of PPODC to be very effective, which is confirmed by Figure 6. When $m = 10{,}000$, $l = 10$, $k = 4$, it takes 2,676 minutes for 1 pair of servers to finish one iteration of PPODC, while the time is reduced to 337 minutes when 8 pairs of servers are available. The speedup is $2{,}676/337 = 7.94$, which is very close to 8.

VII. CONCLUSIONS

In this paper, we proposed a novel privacy-preserving and outsourced distributed clustering (PPODC) protocol under a federated cloud environment that can perform the k-means clustering on the users aggregated encrypted data in a privacy-preserving manner. At the core of our protocol, we proposed new transformations to construct an order-preserving Euclidean distance function and evaluate the termination condition of the k-means clustering algorithm over encrypted data. The proposed PPODC protocol ensures data confidentiality of all users and incurs negligible costs on the user side. In addition, we discussed two techniques to boost the performance of our protocol. A key feature of our protocol is that most of its underlying computations can be parallelized. We implemented our protocol on a cluster of 16 nodes using a real dataset and our experimental results show that its performance can be drastically improved with parallelism. As future work, we will extend the research ideas proposed in this paper to other data mining tasks, such as classification, association rule mining, and regression analysis.

REFERENCES

[1] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, vol. 31, pp. 264–323, 1999.

[2] P. Patrick and L. Dekang, "Document clustering with committees," in *SIGIR*. ACM, 2002, pp. 199–206.

[3] R. Michalski and R. Stepp, *In Machine Learning: An Artificial Intelligence Approach.* Tiago Publishing Co., 1983, ch. Learning from Observation: Conceptual Clustering, pp. 331–363.

[4] B. Andrea and B. Palma, "A survey of fuzzy clustering algorithms for pattern recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 29, no. 6, pp. 778–785, 1999.

[5] C. Su, J. Zhou, F. Bao, T. Takagi, and K. Sakurai, "Two-party privacy-preserving agglomerative document clustering," in *ISPEC*. Springer-Verlag, 2007, pp. 193 – 208.

[6] J. Vaidya and C. Clifton, "Privacy-preserving k-means clustering over vertically partitioned data," in *SIGKDD*, 2003, pp. 206–215.

[7] G. Jagannathan and R. Wright, "Privacy-preserving distributed k-means clustering over arbitrarily partitioned data," in *ACM SIGKDD*, 2005, pp. 593–599.

[8] P. Bunn and R. Ostrovsky, "Secure two-party k-means clustering," in *ACM CCS*, 2007, pp. 486–497.

[9] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *ACM STOC*, 2009, pp. 169–178.

[10] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," in *EUROCRYPT*. Springer, 2011, pp. 129–148.

[11] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[12] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. Academic Press Professional, Inc., 1990.

[13] NIST, "Nist us government cloud computing technology roadmap," Volume I: High Priority Requirements to Further USG Agency Cloud Computing Adoption, November 2011.

[14] O. Goldreich, *The Foundations of Cryptography*. Cambridge University Press, 2004, vol. 2, ch. General Cryptographic Protocols.

[15] P. Paillier, "Public key cryptosystems based on composite degree residuosity classes," in *Eurocrypt*. Springer-Verlag, 1999, pp. 223–238.

[16] I. Damgård and M. Jurik, “A generalisation, a simplification and some applications of paillier’s probabilistic public-key system,” in *PKC*. Springer-Verlag, 2001, pp. 119–136.

[17] C. Hazay, G. L. Mikkelsen, T. Rabin, and T. Toft, “Efficient rsa key generation and threshold paillier in the two-party setting,” in *CT-RSA*. Springer-Verlag, 2012, pp. 313–331.

[18] M. Upmanyu, A. M. Namboodiri, K. Srinathan, and C. V. Jawahar, “Efficient privacy preserving k-means clustering,” in *Intelligence and Security Informatics, Pacific Asia Workshop, PAISI 2010, Hyderabad, India, June 21, 2010. Proceedings*, 2010, pp. 154–166. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13601-6_17

[19] K. Lin, “Privacy-preserving kernel k-means outsourcing with randomized kernels,” in *13th IEEE International Conference on Data Mining Workshops, ICDM Workshops, TX, USA, December 7-10, 2013*, 2013, pp. 860–866. [Online]. Available: http://dx.doi.org/10.1109/ICDMW.2013.29

[20] D. Liu, E. Bertino, and X. Yi, “Privacy of outsourced k-means clustering,” in *ACM ASIACCS*, 2014, pp. 123–134.

[21] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, “Secure k-nearest neighbor query over encrypted data in outsourced environments,” in *ICDE*. IEEE, 2014, pp. 664–675.

[22] I. F. Blake and V. Kolesnikov, “One-round secure comparison of integers,” *J. of Math. Cryptology*, vol. 3, pp. 37–68, 2009.

[23] B. K. Samanthula, Y. Elmehdwi, and W. Jiang, “k-nearest neighbor classification over semantically secure encrypted relational data,” IEEE TKDE. To appear, http://arxiv.org/abs/1403.5001.

[24] O. Goldreich, *The Foundations of Cryptography*. Cambridge Univ. Press, 2004, vol. 2, ch. Encryption Schemes. [Online]. Available: http://www.wisdom.weizmann.ac.il/~oded/PSBookFrag/enc.ps

[25] B. K. Samanthula, C. Hu, and W. Jiang, “An efficient and probabilistic secure bit-decomposition,” in *ASIACCS*, 2013, pp. 541–546.

[26] I. Damgrd, M. Jurik, and J. B. Nielsen, “A generalization of paillier’s public-key system with applications to electronic voting,” *Int. J. Inf. Secur.*, pp. 371–385, 2010.

[27] The GNU MP Bignum Library, https://gmplib.org/.

[28] M. Naeem and S. Asghar, “KEGG Metabolic Reaction Network Data Set,” The UCI KDD Archive, 2011, https://archive.ics.uci.edu/ml/datasets/KEGG+Metabolic+Reaction+Network+(Undirected).