EURASIP Journal on Information Security
a SpringerOpen Journal

RESEARCH Open Access

Privacy-preserving distributed clustering

Zekeriya Erkin[1*], Thijs Veugen[1,2], Tomas Toft[3] and Reginald L Lagendijk[1]

Abstract

Clustering is a very important tool in data mining and is widely used in on-line services for medical, financial and social environments. The main goal in clustering is to create sets of similar objects in a data set. The data set to be used for clustering can be owned by a single entity, or in some cases, information from different databases is pooled to enrich the data so that the merged database can improve the clustering effort. However, in either case, the content of the database may be privacy sensitive and/or commercially valuable such that the owners may not want to share their data with any other entity, including the service provider. Such privacy concerns lead to trust issues between entities, which clearly damages the functioning of the service and even blocks cooperation between entities with similar data sets. To enable joint efforts with private data, we propose a protocol for distributed clustering that limits information leakage to the untrusted service provider that performs the clustering. To achieve this goal, we rely on cryptographic techniques, in particular homomorphic encryption, and further improve the state of the art of processing encrypted data in terms of efficiency by taking the distributed structure of the system into account and improving the efficiency in terms of computation and communication by data packing. While our construction can be easily adjusted to a centralized or a distributed computing model, we rely on a set of particular users that help the service provider with computations. Experimental results clearly indicate that the work we present is an efficient way of deploying a privacy-preserving clustering algorithm in a distributed manner.

1 Introduction

As a powerful tool in data mining, clustering is widely used in several domains, including finance, medicine and social networks, to group similar objects based on a similarity metric. In many cases, the entity that performs the clustering operation has access to the whole database, while in some other cases, databases from different resources are merged to improve the performance of the clustering algorithms. A number of examples can be given as follows:

- *Social networks.* Users are clustered by the service provider based on their profile data. The clustering result can be used for creating self-help groups or generating recommendations. Obviously, in many cases, users would not like to share their profile data with anyone else but with the people that are in the same group.
- *Banking.* Several banks might want to merge their customer databases for credit card fraud detection or to classify their users based on past transactions to identify profitable customers.
- *Medical domain.* Different holders of medical databases might be willing to pool their data for medical research, either for scientific, economic or marketing reasons [1]. Another case can be the Centre for Disease Control that would like to identify trends based on data from different insurance companies [2].

However, regardless of the application setting with one or more data resources, in many cases, data are privacy sensitive or commercially valuable: the data owners might not want to reveal their sensitive data to the service provider, for instance in social networks, as the data can be processed for other purposes, transferred to other third parties without user consent or stolen by outsiders. In the case of multiple data resources from different entities as in banking, the data owners might not want to take risks in sharing their customer data with other competitors. Clearly, such privacy-related concerns might result in several drawbacks: people not joining social networks

*Correspondence: z.erkin@tudelft.nl
[1] Department of Intelligent Systems, Delft University of Technology, Delft, 2628 CD, The Netherlands
Full list of author information is available at the end of the article

or database owners preferring to process data on their own.

In this paper, we focus on a setting with a central entity that provides services based on clustering of multiple users, each one having a private preference vector. Our goal is to prevent the service provider from learning the privacy-sensitive data of the users, without substantially degrading the performance of the clustering algorithm. Thus, we focus on the following:

- *Privacy.* To protect the privacy of users, we encrypt the preference vectors and provide only these encrypted vectors to the service provider, who does not have the decryption key. However, it is still possible for the service provider to cluster people using our cryptographic protocol. Throughout the protocol, the preferences, intermediate cluster assignments and the final results of the clustering algorithm are all encrypted and thus unknown to the service provider or any other person in the network. This approach, which has proved itself useful in the field of privacy-enhanced technologies [3], guarantees privacy protection to the users of the social network without disrupting the service.
- *Performance.* While processing encrypted data as explained above provides privacy protection, it also comes with a price: expensive operations on the encrypted data, in terms of computational and communication costs. To improve the efficiency, we approach this challenge in two directions: (1) custom-tailored cryptographic protocols that use data packing and (2) a setting in which the service provider creates user sets and assigns additional responsibilities to one of the users in each set to be able to use less expensive cryptographic sub-protocols for the computations, avoiding expensive computations such as the ones in [4]. Moreover, having such a construction, centralized or distributed clustering scenarios can be realized, as discussed further below.

The service provider is defined as the entity that wants to cluster users based on their private preference vectors. Each user also participates in the clustering computations, and a set of users, named helper users, are chosen randomly to perform additional tasks. As the number of user sets increases, it becomes easier to parallelize operations and thus achieve better performance. However, this setting with one set of users and a single helper user can also be considered to realize clustering algorithms for the scenarios with multiple entities, each having a private database: users belong to different entities, and the helper user becomes a privacy service provider [4]. Thus, our construction can easily be reshaped according to the application.

In this paper, we choose the K-means clustering algorithm for finding the group of similar people based on their similarities. We choose the K-means algorithm since it is known to be a very efficient data mining tool that is widely used in practice as its implementation is simple and the algorithm converges quickly [5]. Our goal is to provide an efficient, privacy-preserving version of the clustering algorithm. Even though the idea of processing encrypted data for clustering has been addressed before in the literature, its realization in an efficient way has been a challenge. To improve the state of the art, we contribute in the following aspects:

- We propose a flexible setting, which can be interpreted as a centralized or a distributed environment with several servers. This enables a wide variety of business models.
- We build our system based on the semi-honest security model, in which we assume that involved parties are following the protocol steps. For the application settings, where the central entity is expected to go beyond the bounds of the protocol, our protocol can be tweaked to work in the malicious model with a cost of increased computation and communication [6]. However, we provide an alternative that is in between: we distribute trust among a number of helper users instead of relying on a single party. Especially in a setting with distributed databases, this substantially limits the power of a malicious central party.
- We exploit the construction with helper users to avoid more expensive cryptographic protocols such as secure comparisons [4], achieving significant performance gain compared to related work in the field.
- We employ custom-tailored cryptographic protocols with data packing [4,7,8] to reduce the communication and computation costs of using homomorphic encryption.

We emphasize that our proposal is an improvement of the ideas from [9] and [10] in terms of efficiency and requires reasonable security and business assumptions. Our main contribution is to show that realizing privacy-preserving K-means clustering with existing tools is feasible to deploy. To prove our claim of achieving high efficiency, we also give the test results of our proposal on a synthetic data set of 100,000 users.

The rest of the paper is organized as follows: Section 2 gives an overview on the state of the art. Section 3 introduces K-means clustering algorithm and homomorphic

encryption briefly and presents our security assumptions and the notation used throughout the paper. Section 4 describes the privacy-preserving version of the K-means clustering algorithm in detail. Section 5 discusses the security aspects of our proposal, and Section 6 presents the complexity analysis and the numerical test results. Section 7 gives a discussion on the practicality of deploying our protocol in real life. Finally, Section 8 concludes this paper.

2 Related work

The idea of privacy-preserving data mining was introduced by Agarwal and Srikant [11] and Lindell and Pinkas [1]. In their work, the aim is to extract information from users' private data without having to reveal individual data items. Since then, a number of privacy protection mechanisms for finding similar items or people have been proposed in [2,12-16], which address the widely applied K-means clustering algorithm. The proposed methods apply either cryptographic tools [2,12-14] or randomization techniques from signal processing [15,16] to protect the private data, which are either horizontally or vertically partitioned.

In general, the cryptographic proposals are based on secure multiparty computation techniques [6], which make any two-party privacy-preserving data mining problem solvable, for instance by using Yao's secure circuit evaluation method [17]. Even though Yao's method can be used to implement any function in a privacy-preserving manner, heavy computation or communication costs in such circuits make the solutions feasible only for small circuit sizes. However, algorithms like clustering require large circuit sizes for realization. In [2,12,14], the authors attempt to solve the clustering problem in a two-party setting which is suitable for deploying techniques based on secret sharing. Apart from the difference in settings, [2] suffers from a problem during the centroid update procedure where an integer division is misinterpreted as multiplication by the inverse, which is not correct, as explained with an example in [12]. On the other hand, [13] has a multiuser setting but requires three non-colluding entities for the clustering algorithm, and the authors overcome the problem of updating centroids by allowing users to perform the division algorithm locally. In order to do that, the users learn the intermediate cluster assignments, meaning more information leakage.

As a different approach from using secure multiparty computation techniques, Oliviera and Zaiane [15,16] suggested using techniques from signal processing based on randomization and geometric transformation of data to hide private data of individuals. In these works, the privacy of the users is achieved by perturbing their data in a predefined way. Then, the data is made publicly available for processing. This approach is fast since the operations can be handled by each user simultaneously. However, data perturbation leads to unavoidable data leakage [18,19].

In [9], Erkin et al. proposed a method based on encryption and secure multiparty computation techniques for clustering users in a centralized system. In that work, Erkin et al. kept the preference vector of each user in the system hidden from all other users and the service provider and reveal the centroid locations to the service provider for achieving better performance in terms of run-time and bandwidth. The proposed method requires the participation of all users, and the average communication and computation cost is high due to homomorphic encryption. In [10], Beye et al. proposed an improved version of K-means clustering by proposing a three-party setting. In that work, users' private data are stored by one party and the decryption key by the other. A third party helps with the computations. Due to this three-party setting, Beye et al. proposed a highly efficient algorithm based on garbled circuits [20] that does not require oblivious transfer protocols [6]. While the overall system is highly efficient, the authors rely on trusting three separate parties that may not collude.

3 Preliminaries

In this section, we briefly introduce the K-means clustering algorithm, present our security assumptions, describe homomorphic encryption and introduce the notation used throughout the paper.

3.1 K-means clustering

Data clustering is a common technique for statistical data analysis where data is partitioned into smaller subgroups with their members sharing a common property [5]. As a widely used technique, K-means clusters data into K groups using an iterative algorithm. Particularly, each user i is represented as a point in an R-dimensional space, denoted with $P_i = (p_{(i,1)}, \ldots, p_{(i,R)})$, and assigned to the closest cluster among K clusters, $\{C_1, \ldots, C_K\}$. The algorithm starts with choosing the constant value K, which is the number of clusters in the data set. Each cluster k is represented by its centre (also named centroid), $C_k = (c_{(k,1)}, \ldots, c_{(k,R)})$, which is initially a random point. In every iteration, the distances $D_{(i,k)}$ between the ith user P_i and cluster centre C_k for $k \in \{1, \ldots, K\}$ are calculated and each user is assigned to the closest cluster. Once every user is associated with a cluster, centroid locations are recalculated by taking the arithmetic mean of the users' locations within that cluster. For the next iteration, the distances are recalculated and users are assigned to the closest cluster. This procedure, given in Algorithm 1, is repeated until either a certain

number of iterations is reached or centroid locations converge.

Algorithm 1 The K-means clustering algorithm.

Input: K (randomly) chosen locations as cluster centroid.
Output: Cluster indices for each user.

1: For each user P_i, compute the distances to each cluster centroid C_k for $k \in \{1, \ldots, K\}$.
2: Assign each user to the closest cluster.
3: Recalculate the centroid locations by taking the arithmetic mean of the users' locations within that cluster.
4: Repeat steps 1, 2 and 3 until one of the termination conditions is reached: (a) a certain number of iterations or (b) centroid locations converge.
5: Output the cluster indices for each user.

3.2 Security assumptions

We consider the semi-honest security model, which assumes that the involved parties are honest and follow the defined protocol steps but are also curious to obtain more information. Therefore, the parties can store previously exchanged messages to deduce more information than they are entitled to. This model does not consider any malicious activity by the parties such as manipulating the original data.

We assume that the service provider creates groups of people randomly to help in computations, in which a number of people takes more responsibility in computations. Our assumption is that these randomly chosen users do not collude with the service provider in revealing other users' personal data. The risk of information leakage by such parties is reduced, as explained later, by randomly choosing such helper users in each iteration of the algorithm. Note that as the number of helper users increases, the security of the system improves since the trust is divided among multiple entities, rather than one single entity.

Note that even though we assume that the service provider acts according to the protocol description, it is possible that the service provider can create dummy users and assign them as helper users. There are two approaches to cope with this problem. Firstly, each user participating in the computations can be asked to use certifications to prove their identity. Secondly, the helper users can be chosen truly random by deploying another sub-protocol. For this, the ideas from [21] can be used. Furthermore, in the case of malicious acts requiring input verification by the users, the techniques known from cryptography like commitment schemes and zero-knowledge proofs can be deployed at the cost of increased complexity.

Finally, we also assume that all underlying communication channels are secure: both integrity and authentication of all messages are obtained via standard means, e.g. IPSec or SSL/TLS [22].

3.3 Homomorphic encryption

The public-key cryptosystem Paillier [23] is *additively homomorphic*, meaning that multiplication of two cipher texts results in a new cipher text whose decryption equals the sum of the two plain texts. Given the plain texts m_1 and m_2, the additively homomorphic Paillier works as follows:

$$\mathcal{D}_{\mathrm{sk}}\left(\mathcal{E}_{\mathrm{pk}}(m_1) \cdot \mathcal{E}_{\mathrm{pk}}(m_2)\right) = m_1 + m_2,$$

where pk and sk are the public and secret keys, respectively. As a consequence of the additive homomorphism, any cipher text $\mathcal{E}_{\mathrm{pk}}(m)$ raised to the power c results in a new encryption of $m \cdot c$ as

$$\mathcal{D}_{\mathrm{pk}}\left(\mathcal{E}_{\mathrm{pk}}(m)^c\right) = m \cdot c\,.$$

The encryption of a message, $m \in \mathbb{Z}_n$, by using the Paillier scheme is defined as

$$\mathcal{E}_{\mathrm{pk}}(m) = g^m \cdot r^n \bmod n^2,$$

where n is a product of two large prime numbers, g generates a sub-group of order n and r is a random number in $\mathbb{Z}_n^*$. Note that the message space is $\mathbb{Z}_n$ and the cipher text space is $\mathbb{Z}_{n^2}^*$. For decryption and further details, we refer readers to [23].

In addition to the homomorphic property, Paillier is semantically secure. Informally, this means that one cannot distinguish between encryptions of known messages and random messages. This is achieved by having multiple possible cipher texts for each plain text and choosing randomly between these. This property is required as the messages to be encrypted in this paper are from a very small range compared to the message space of the cryptosystem. Throughout this paper, we denote a Paillier encryption of a message m by $[\![m]\!]_{pk}$ for the sake of simplicity.

3.4 Notation

We use uppercase and lowercase letters to represent a vector and its elements, respectively: $A_i = (a_{(i,1)}, a_{(i,2)}, \ldots, a_{(i,R)})$. We represent the packed version of a vector as $\widetilde{A}_i = a_{(i,1)}||a_{(i,2)}||\ldots||a_{(i,R)}$. We present the variables used throughout the paper in Table 1.

4 Privacy-preserving user clustering

In this section, we present a cryptographic protocol that clusters users in a social network setting using the K-means algorithm. Our aim is to hide the private data of the users from the service provider. We define the private data as the preference vectors P_i, distances between the P_i

Table 1 List of symbols

Symbol	Description
M	Number of groups created by the service provider
N_u	Total number of users in the social network
K	Number of clusters
P_i	Preference set of user i
$\widetilde{P}_{\Sigma_{(m,r)}}$	Packed sum of preferences of users in G_m
$p_{(i,r)}$	rth coordinate of P_i
$c_{(k,r)}$	rth coordinate of cluster k
$\widetilde{P}_i$	Packed preference of user i
$\widetilde{C}_r$	Packed centroid for dimension r
$\gamma_{(i,k)}$	Binary value for the kth cluster: 1 for the closest cluster, 0 otherwise
$\widetilde{\Gamma}_{\Sigma_m}$	Packed total number of users in each cluster for G_m
w	Bit length of $p_{(i,r)}$ and $c_{(j,r)}$
$\alpha, \beta, \phi, \psi, \rho$	Random values
$[\![\cdot]\!]_H$	Encryption of a plain text using the public key of H_m
$\widetilde{S}_{(i,r)}$	Packed partial input of user i for dimension r
N_g	Number of users in each group
R	Dimension of preferences
G_m	Group m
$\widetilde{P}_i$	Packed preferences of user i
$\widetilde{P}_{\Sigma_r}$	Packed sum of preferences of all users for dimension r
C_k	Cluster centroid k
$D_{(i,k)}$	Euclidean distance between P_i and C_k
$\widetilde{D}_{(i,k)}$	Packed Euclidean distance between P_i and C_j
Δ	Compartment size of $\widetilde{\Gamma}_i$ in bits
$\widetilde{\Gamma}_i$	Packed vector of γ's for user i
$\widetilde{\Gamma}_\Sigma$	Packed total number of users in each cluster
σ	Statistical security parameter
H_m	Helper user of group m
n	Paillier modulus
ℓ	Compartment size of $\widetilde{C}$ in bits

and the centroid C_k, and the result of the clustering algorithm, which is the final cluster assignment of each user. We define the following roles for our construction:

1. The *service provider* has a business interest in providing services to users in a social network.
2. A *user* participates in a social network and would like to find similar other users based on his/her preferences.
3. A *helper* is a user who helps the service provider with the computations.

We assume that each user has a public key pair with valid certificates. We anticipate that the service provider determines K points to be the initial cluster centroids. To cluster all users, the following steps are performed:

1. The service provider creates M groups of users and, for each group, G_m for $m \in \{1, \ldots, M\}$, selects a random user H_m for each iteration as illustrated in Figure 1. We assume that there are N_g users in each group and the total number of users is $N_u = N_g \cdot M$.
2. The service provider informs every user in G_m about the public key to be used for encryption in that iteration, which is the public key of H_m.
3. The service provider sends the encrypted cluster centroids to the users. Each user computes K encrypted Euclidean distances, one for every cluster centroid, and sends them to the service provider.
4. The service provider interacts with H_m to obtain an encrypted vector for each user, whose elements indicate the closest cluster to that user. Then, the service provider sends this vector to the user.
5. Each user computes his/her partial input for updating the centroid locations using the encrypted vector and sends it to the service provider.
6. The service provider aggregates the partial inputs from all users in G_m and interacts with H_m to obtain the clustering result of G_m in plain text.
7. Finally, the service provider combines the clustering results from all groups and obtains the new centroids for that iteration.

Steps 1 to 7 are repeated either for a certain number of iterations or up to a point where the cluster centroids do not change significantly. After the final iteration, the service provider runs a protocol with H_m to send the index of the closest cluster to each user. Hereafter, we describe the above procedure in detail for a single group.

4.1 Steps 1 and 2: grouping and key distribution

The first step of the K-means clustering algorithm is for the service provider to choose K points in an R-dimensional space as the initial cluster centroids. Next, the service provider creates M groups consisting of N_g users each and picks a random user from every group who will help the service provider with the computations for the current iteration. Note that in order for the helper users to be clustered, the service provider treats them as an ordinary user in a different group. Later, the service provider disseminates the public key of the helper user to the other users.

4.2 Step 3: computing the encrypted distances

In principle, the service provider and the users in G_m compute the Euclidean distances from each user's preference

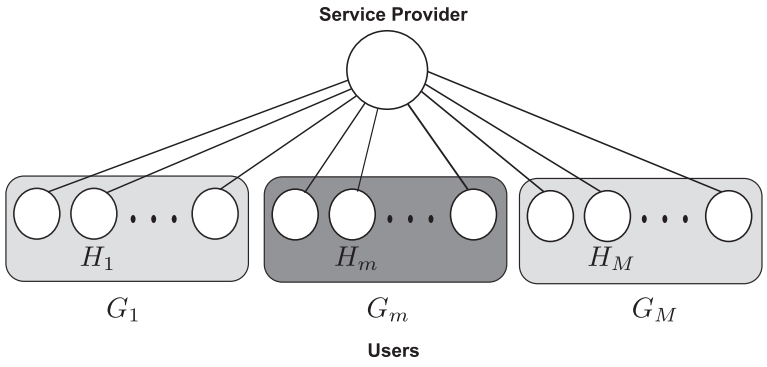

Figure 1 The user groups of the secure K-means clustering.

vector P_i to every cluster centroid C_k as given in Equation 1:

$$D^2_{(i,k)} = ||\mathbf{P}_i - \mathbf{C}_k||^2 = \sum_{r=1}^{R} p^2_{(i,r)} + \sum_{r=1}^{R}(-2p_{(i,r)}c_{(k,r)}) + \sum_{r=1}^{R} c^2_{(k,r)}\,, \tag{1}$$

where user i possesses P_i and the service provider holds cluster locations C_k, which are privacy sensitive and hence should be kept secret from the other party. To compute the Euclidean distance without revealing private data, the service provider and user i use the homomorphic property: given that encryption of C_k and sums $\sum_{r=1}^{R} c^2_{(k,r)}$ are provided by the service provider, the encrypted distance could be computed as follows [9]:

$$\left[\!\left[D^2_{(i,k)}\right]\!\right]_H = \left[\!\left[\sum_{r=1}^{R} p^2_{(i,r)}\right]\!\right]_H \cdot \prod_{r=1}^{R} [\![c_{(k,r)}]\!]_H^{-2p_{(i,r)}} \cdot \left[\!\left[\sum_{r=1}^{R} c^2_{(k,r)}\right]\!\right]_H . \tag{2}$$

The above approach to compute the encrypted distances in [9] uses the homomorphic property of the cryptosystem without any optimization and therefore introduces a considerable computational overhead for user i. More precisely, that computation requires $K(R+1)$ encryption by the service provider and one encryption, KR exponentiation and $K(R+1)$ multiplications by *each* user over mod n^2. Repeating these expensive operations for N_u users, the clustering algorithm becomes considerably expensive and thus impractical in real life.

To improve the efficiency in terms of communication and computation, we use data packing by following a similar approach as in [4,7,8]. Instead of computing separate Euclidean distances in the encrypted domain, users can compute a single *packed* value, which involves K distances, with the help of the service provider. For this purpose, the service provider creates the following packed values:

$$\begin{aligned}
\widetilde{C}_1 &= c_{(1,1)}||c_{(2,1)}||\ldots||c_{(K,1)}\\
\widetilde{C}_2 &= c_{(1,2)}||c_{(2,2)}||\ldots||c_{(K,2)}\\
&\vdots\\
\widetilde{C}_R &= c_{(1,R)}||c_{(2,R)}||\ldots||c_{(K,R)},
\end{aligned} \tag{3}$$

where $||$ denotes concatenation. Formally, the above values are calculated as follows:

$$\widetilde{C}_r = \sum_{k=1}^{K} c_{(k,r)} \cdot (2^\ell)^{k-1}, \text{for } r \in \{1,\ldots,R\}\,. \tag{4}$$

The bit length of each $\widetilde{C}_r$ is now $K \times \ell$ bits. We set the size of each compartment to $\ell = 2w + \lceil \log R \rceil$ bits, with w being the bit length of $c_{(k,r)}$, to accommodate the distances computed in the consequent steps, which are the sum of R positive numbers of size $2w$ bits.

In addition to the packed values above, the service provider prepares the following value:

$$\widetilde{C}^2 = \sum_{r=1}^{R} c^2_{(1,r)}||\sum_{r=1}^{R} c^2_{(2,r)}||\ldots||\sum_{r=1}^{R} c^2_{(K,r)}\,, \tag{5}$$

where the sums of squares are also packed in compartments of size ℓ-bits. Then, the service provider encrypts $\widetilde{C}^2$ and $\widetilde{C}_r$ for $r \in \{1\ldots R\}$ with the public key of H_m and sends the encrypted values to the users. Next, each user i of G_m computes the packed distances as follows:

$$\begin{aligned}
\left[\!\left[\widetilde{D}^2_i\right]\!\right]_H &= \left[\!\left[D^2_{(i,1)}||D^2_{(i,2)}||\ldots||D^2_{(i,K)}\right]\!\right]_H\\
&= \left[\!\left[\widetilde{C}^2\right]\!\right]_H \cdot \prod_{r=1}^{R} \left[\!\left[\widetilde{C}_r\right]\!\right]_H^{-2p_{(i,r)}} \cdot \left[\!\left[\widetilde{P}^2\right]\!\right]_H\,,
\end{aligned} \tag{6}$$

where

$$\widetilde{P}^2 = \underbrace{\sum_{r=1}^{R} p_{(i,r)}^2 || \ldots || \sum_{r=1}^{R} p_{(i,r)}^2 || \sum_{r=1}^{R} p_{(i,r)}^2}_{K \text{ times}} . \quad (7)$$

User i then sends $[\![\widetilde{D}_i^2]\!]_H$ to the service provider.

Remark 1. Each squared distance $D_{(i,k)}^2$ consists of at most ℓ-bits. We assume that $K \cdot \ell \ll n$, where n is the message space of the Paillier encryption scheme, meaning all of the K distances can be packed in one encryption. Note that $|p_{(i,r)} - c_{(k,r)}| \leq \max(p_{(i,r)}, c_{(k,r)})$, and thus, $D_{(i,k)}^2 \leq R \cdot 2^{2w}$.

Notice that due to the way we compute distances using data packing, there is a gain by a factor of K in the number of operations on the encrypted data compared to [9]. The computation of the packed encrypted distances only requires $R+1$ encryption by the service provider and one encryption, R exponentiation and $R+1$ multiplications by each user.

4.3 Step 4: finding the closest cluster

After having obtained $[\![\widetilde{D}_i^2]\!]_H$, the service provider interacts with H_m to find out the minimum distance, hence the closest cluster. To achieve this, the service provider sends the packed distances to H_m, who has the decryption key. H_m decrypts the cipher text and obtains the packed distances in clear. Note that H_m does not know the identity of the owner of the computed distances. After decryption, H_m unpacks the distances and creates a vector $(\gamma_{(i,1)}, \gamma_{(i,2)}, \ldots, \gamma_{(i,K)})$, where $\gamma_{(i,k)}$ is 1 if and only if $D_{(i,k)}^2$ is the minimum distance (so user i is in cluster number k), and 0 otherwise. Before sending these binary values to the service provider, H_m encrypts them using his/her public key.

Upon receiving the values $[\![\gamma_{(i,k)}]\!]_H$'s from H_m, the service provider packs them to reduce the bandwidth usage and to simplify the computations in the subsequent steps:

$$[\![\widetilde{\Gamma}_i]\!]_H = [\![\gamma_{(i,1)} || \gamma_{(i,2)} || \ldots || \gamma_{(i,K)}]\!]_H = \prod_{k=1}^{K} [\![\gamma_{(i,k)}]\!]_H^{2^{\Delta(k-1)}} = \left[\!\!\left[\sum_{k=1}^{K} \gamma_{(i,k)} \cdot 2^{\Delta(k-1)} \right]\!\!\right]_H , \quad (8)$$

where $\Delta = w + \lceil \log N_u \rceil$, N_u being the number of total users in the system. This gives one packed $\widetilde{\Gamma}_i$ with a compartment size of $w + \lceil \log N_u \rceil$ bits. The service provider, then, sends $\widetilde{\Gamma}_i$ to the users.

Remark 2. Notice that in the above procedure, H_m will learn how many users in his/her group belong to each cluster. To hide this information from H_m, the service provider uses a different permutation, π_i, independently chosen for each user to shuffle the order of clusters during the creation of the $\widetilde{C}_r$ values. The order is corrected when the service provider applies the inverse permutation, π_i^{-1}, on the received $[\![\gamma_{(i,k)}]\!]$'s. As this permutation is necessary and can only be done by the service provider, H_m cannot apply data packing himself, which would simplify the computations otherwise.

Remark 3. We assumed in Equation 8 that packing K $\gamma_{(i,k)}$'s, each within a compartment size of $w + \lceil \log N_u \rceil$ bits, is possible. This is a valid assumption in practical cases since the Paillier modulus, even for a weak security, is 1,024 bits. Given that $K = 10$ and $w = 3$, N_u can be as large as 2^{99}.

4.4 Step 5: computing partial inputs

To update K cluster centroids, the service provider needs to take the average of user preferences in each cluster under encryption. To achieve this, upon receiving $[\![\widetilde{\Gamma}_i]\!]$, user i computes

$$[\![\widetilde{S}_{(i,r)}]\!]_H = [\![\widetilde{\Gamma}_i]\!]_H^{p_{(i,r)}} = [\![\gamma_{(i,1)} \cdot p_{(i,r)} || \ldots || \gamma_{(i,K)} \cdot p_{(i,r)}]\!]_H, \quad (9)$$

for $r \in \{1, \ldots, R\}$. The result of this operation is R encryptions, each of which contains K packed values. Each compartment of the encryptions contains the multiplication of $\gamma_{(i,k)}$ and $p_{(i,r)}$ for $k \in \{1, \ldots, K\}$. It is clear that $K-1$ compartments consist of zeros and only one compartment that has the index of the closest cluster is exactly $p_{(i,r)}$. User i, finally, sends $[\![\widetilde{S}_{(i,r)}]\!]$ for $r \in \{1, \ldots, R\}$ to the service provider.

To update the cluster centroids, the service provider needs the number of users and the sum of preferences in each cluster. Since $[\![\widetilde{\Gamma}_i]\!]$'s are available to the service provider, it easily computes the number of users in each cluster for G_m under encryption:

$$[\![\widetilde{\Gamma}_{\Sigma_m}]\!]_H = \prod_{i \in G_m} [\![\widetilde{\Gamma}_i]\!]_H = \left[\!\!\left[\sum_{i \in G_m} \widetilde{\Gamma}_i \right]\!\!\right]_H = \left[\!\!\left[\sum_{i \in G_m} \gamma_{(i,1)} || \sum_{i \in G_m} \gamma_{(i,2)} || \ldots || \sum_{i \in G_m} \gamma_{(i,K)} \right]\!\!\right]_H . \quad (10)$$

Similarly, the service provider computes $P_{\Sigma(m,r)}$, the sum of user preferences of G_m, for each cluster as follows:

$$[\![\widetilde{P}_{\Sigma_{(m,r)}}]\!]_H = \prod_{i\in G_m} [\![\widetilde{S}_{(i,r)}]\!]_H = \left[\!\!\left[\sum_{i\in G_m} \widetilde{S}_{(i,r)} \right]\!\!\right]_H , \qquad (11)$$

for $r \in \{1,\ldots,R\}$. This results in R encryption, one for each dimension, and each of which has K packed sums of preferences of users in G_m.

4.5 Step 6: aggregating partial inputs

The next step for the service provider is to obtain the decryptions of $[\![\widetilde{\Gamma}_{\Sigma_m}]\!]_H$ and $[\![\widetilde{P}_{\Sigma_{(m,r)}}]\!]_H$. For this reason, the service provider interacts with H_m. As these values are also privacy sensitive, the service provider prevents H_m from accessing the content of the cipher text by applying masking: it generates two sets of random numbers α_m and $\beta_{(m,r)}$ that are $K\cdot\Delta+\sigma$ bits long, where σ is a statistical security parameter in the range of 40 to 100 bits. After that, the server blinds $\widetilde{\Gamma}_{\Sigma_m}$ and $\widetilde{P}_{\Sigma_{(m,r)}}$ by performing the following multiplications:

$$\begin{aligned} [\![\widetilde{\Gamma}_{\Sigma_m}+\alpha_m]\!]_H &= [\![\widetilde{\Gamma}_{\Sigma_m}]\!]_H \cdot [\![\alpha_m]\!]_H , \\ [\![\widetilde{P}_{\Sigma_{(m,r)}}+\beta_{(m,r)}]\!]_H &= [\![\widetilde{P}_{\Sigma_{(m,r)}}]\!]_H \cdot [\![\beta_{(m,r)}]\!]_H , \end{aligned} \qquad (12)$$

for $r\in\{1,\ldots,R\}$. Then, the service provider sends $[\![\widetilde{\Gamma}_{\Sigma_m}+\alpha_m]\!]_H$ and $[\![\widetilde{P}_{\Sigma_{(m,r)}}+\beta_{(m,r)}]\!]_H$ to H_m. After decrypting them, H_m could send $\widetilde{\Gamma}_{\Sigma_m}+\alpha_m$ and $\widetilde{P}_{\Sigma_{(m,r)}}+\beta_{(m,r)}$ to the service provider, who could remove the masking by subtracting the random values, but this would reveal sensitive information to the service provider about the distribution of users in each group. To avoid this information leakage, H_m also applies masking by adding random values, ϕ_m and $\psi_{(m,r)}$, which are computed as described below, and sends the resulting masked values to the service provider.

The random values ϕ_m and $\psi_{(m,r)}$ of size $K\cdot\Delta+\sigma$ bits are generated by a single helper user prior to the start of the iteration such that $\sum_{m=1}^{M}\phi_m=0$ and $\sum_{m=1}^{M}\psi_{(m,r)}=0$ for $r\in\{1,\ldots,R\}$. Each of these random values are then encrypted with the public key of the corresponding helper user and sent to the service provider, who passes them to the corresponding H_m. Finally, each helper user sends $\widetilde{\Gamma}_{\Sigma_m}+\alpha_m+\phi_m$ and $\widetilde{P}_{\Sigma_{(m,r)}}+\beta_{(m,r)}+\psi_{(m,r)}$ to the service provider.

4.6 Step 7: obtaining the new cluster centroids

After receiving masked values from all of the M groups, the service provider obtains the masked packed sums of preferences and the number of users in each cluster, separately for each group. The service provider adds the plain texts from all H_m's and obtains

$$\sum_{m=1}^{M}(\widetilde{\Gamma}_{\Sigma_m}+\alpha_m+\phi_m)\ ,\ \sum_{m=1}^{M}(\widetilde{P}_{\Sigma_{(m,r)}}+\beta_{(m,r)}+\psi_{(m,r)})\ . \qquad (13)$$

Recall that $\sum_{m=1}^{M}\phi_m=0$ and $\sum_{m=1}^{M}\psi_{(m,r)}=0$. Since the service provider knows $\sum_{m=1}^{M}\alpha_m$ and $\sum_{m=1}^{M}\beta_{(m,r)}$, he subtracts them from the total and obtains $\widetilde{\Gamma}_\Sigma=\sum_{m=1}^{M}\widetilde{\Gamma}_{\Sigma_m}$ and $\widetilde{P}_{\Sigma_r}=\sum_{m=1}^{M}\widetilde{P}_{\Sigma_{(m,r)}}$ for $r\in\{1,\ldots,R\}$. Notice that $\widetilde{P}_{\Sigma_r}=\sum_{i\in C_1}p_{(i,r)}||\ldots||\sum_{i\in C_K}p_{(i,r)}$ and $\widetilde{\Gamma}_\Sigma=\sum_{i\in C_1}\gamma_{(i,r)}||\ldots||\sum_{i\in C_K}\gamma_{(i,r)}$.

Finally, the service provider unpacks $\widetilde{\Gamma}_\Sigma$ and $\widetilde{P}_{\Sigma_r}$ and computes the new cluster centroids as follows:

$$c_{(k,r)}=\lceil\frac{\widetilde{P}_{\Sigma_r}(k)}{\Gamma_\Sigma(k)}\rceil , \qquad (14)$$

for $k\in\{1,\ldots,K\}$ and $r\in\{1,\ldots,R\}$, where $\Gamma_\Sigma(k)$ is the total number of users in cluster k and $P_{\Sigma_r}(k)$ is the rth coordinate of the sum of preferences in cluster k, obtained after unpacking. The result of this operation is then rounded to the nearest integer. The complete steps of the protocol for one iteration is given in Figure 2.

4.7 Termination control and obtaining the cluster index

The service provider checks whether the predetermined termination condition is reached at the end of each iteration. Since centroid locations and the number of iterations are known to the service provider in plain text, termination control is considered to be costless. Once the termination condition is reached, i.e. when a certain number of iterations is reached or when centroids do not move significantly, the cluster index of the user, which is the non-zero element in the encrypted vector $[\![\widetilde{\Gamma}_i]\!]_H$, should be delivered to the user in plain text. For this purpose, after the last iteration, the service provider sends $[\![\widetilde{\Gamma}_i]\!]_H$ to user i, who masks it with a random number ρ_i of size $\log(KR)+\sigma$ bits to get $[\![\widetilde{\Gamma}_i+\rho_i]\!]_H$, and sends it back to the service provider. The service provider sends the cipher text to H_m to be decrypted. After receiving the plain text, the service provider sends $\widetilde{\Gamma}_i+\rho_i$ to user i, who can easily obtain the cluster index by subtracting ρ_i from the decrypted value and checking the non-zero value in the compartments. Notice that all messages pass through the service provider instead of sending them directly to the helper user. This is unavoidable in our design as the users do not know the identity of the helper user. As an alternative approach, the encrypted random value ρ_i can also be sent directly to the service provider at the start of the protocol. Having this random number, the service provider can mask $\widetilde{\Gamma}_i$ and send it to the helper user to be decrypted. By this way, transmission of $\widetilde{\Gamma}_i$ to user i can be avoided.

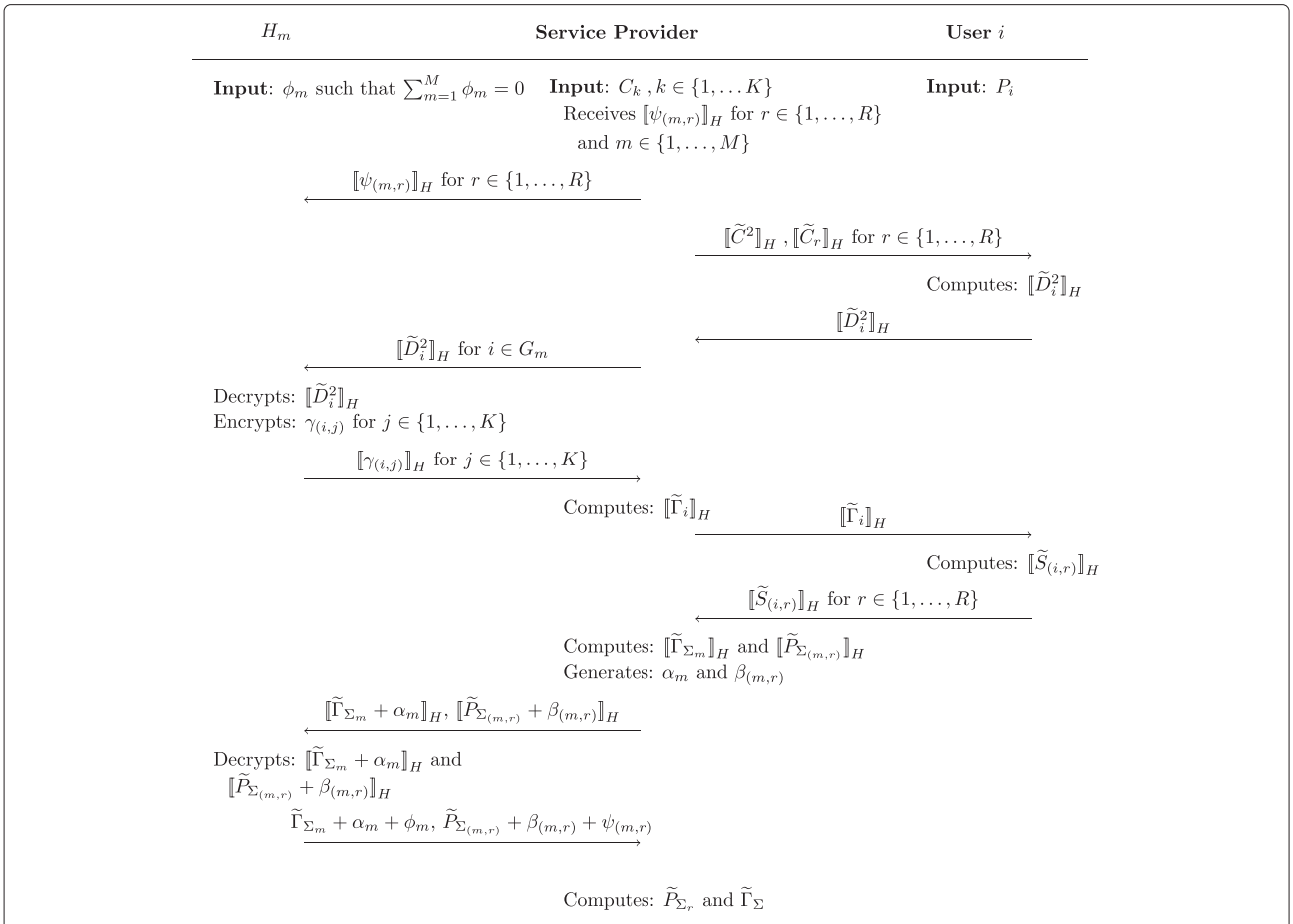

Figure 2 One iteration of the protocol.

5 Security discussion

In this section, we present arguments to show that our protocol is secure under the semi-honest security model. Recall that this model expects involved parties, namely the service provider, the helper users and all other users, to be honest in following the protocol steps. These parties are also assumed to be curious so they can keep previous messages to deduce more information than they are entitled to. This model does not consider corrupted parties. In this paper, we consider only one flavour of security threat: information leakage. In the following, we present an informal discussion on this issue.

Before discussing what information each party can derive from the received messages, we need to point out what information is allowed to be accessed. Remember that the number of clusters is public information. While the users to be clustered receive only the index of the cluster they are assigned to, the service provider does not obtain any information on the preferences of the users nor the final clustering results. However, a helper user obtains the distances between a user and the cluster centres. Even though the distances are permuted and the identity of the user is unknown to the helper user, the helper user can still acquire information about the distances between clusters and the number of assignments for each cluster. Note that enabling helper users to access permuted distances seems to be a reasonable compromise to achieve better performance in computation. Although not formally proved, we believe that the privacy risks created here will be rather harmless to the users, particularly when the number of clusters grows. Considering that in each iteration a different user will be assigned as a helper user, the amount of information on the distances between centres diminishes.

Recall that every user, including the helper users, only interacts with the service provider using a secured channel. The public key of the helper user is also delivered to the users in that group by the service provider. On the basis of this information, we analyze what information can be inferred from exchanged messages.

5.1 Service provider

The service provider receives from the helper users encrypted packed distances of the users $[\![\widetilde{D}^2_{(i,k)}]\!]_H$ and the encrypted binary values $[\![\gamma_{(i,k)}]\!]_H$ that show the closest

cluster for each user. As the Paillier cryptosystem is semantically secure, meaning that it is infeasible for a computationally bounded adversary to derive significant information about the plain text when its cipher text and the public key used are known, it is not feasible for the service provider from obtaining meaningful information from the cipher texts without the decryption key. However, the service provider also receives the sum of preferences and the number of users in each cluster within the group from the helper users in plain text. To prevent the service provider from accessing this information, the helper users mask their messages using random numbers, ϕ_m and $\psi_{(m,r)}$, in such a way that when these masked values are all added up, random values cancel each other out and the service provider gets the final result of the clustering for that iteration. The values received are statistically indistinguishable from random values with the same sum, but completely independent of the group sums. Therefore, the service provider does not have access to any information that might harm the users.

Notice that the way helper users perturb data prevents the service provider to obtain meaning information about each user group. This data perturbation technique will serve its purpose as long as the random numbers are generated accordingly, and helper users do not cooperate with the service provider. Within the semi-honest model, we assume that random number generation is performed properly. Selecting a number of helper users randomly for each iteration also reduces the risk of possible cooperation.

5.2 Helper user

In each group, all computations on the encrypted data are performed by using the public key of the helper user. As pointed out before, the helper user receives and sends data from and to the service provider only. To prevent the helper user from knowing the number of users in each cluster, the service provider applies a different permutation for each user during packing of the centroids. As a result of different permutations, the helper cannot observe the actual cluster with the minimum distance. Note that while the helper user learns the distances between users and K centroids, it is not possible to know the distance between a specific user and a certain cluster since both is kept hidden from the helper user. Furthermore, since in each iteration helper user changes, deducing meaningful information from computed distances becomes infeasible.

Hiding the sum of preferences in each group by applying permutation, on the other hand, is not possible. To hide this information, the service provider masks the values by adding random values, which guarantees that the helper user cannot infer meaningful information.

5.3 Users

Users receive encrypted messages from the service provider and without the decryption key, they cannot access the content. As a result, users cannot obtain information on the intermediate values of the clustering algorithm.

Although out of the scope of our semi-honest model, users are able to manipulate the clustering output by providing fake input data. As long as the size of the input is correct, such an attack would also not be prevented. However, since a user only obtains the index of his cluster, such an attack is not likely to lead to information leakage.

6 Performance

In this section, we present the complexity analysis of the privacy-preserving K-means clustering algorithm and experimental results on its performance.

6.1 Complexity analysis

The privacy-preserving version of the clustering algorithm presented in this paper has a number of disadvantages compared to the version in plain text. User preferences, which are usually small non-negative integers, grow to large numbers, e.g. 2,048 bits, after encryption. On top of that, addition and multiplication on the plain text become multiplication and exponentiation over mod n^2, which are computationally time-consuming. Moreover, transmission of the data from users to the service provider and vice versa requires more bandwidth than the plain version. Finally, realization of the algorithm involves interactive steps, requiring data exchange between the users and the service provider, which do not exist in the clustering algorithm with plain text data.

In Table 2, we present the complexity of our protocol in terms of expensive operations, namely encryption, decryption, multiplication and exponentiation, and communication cost for a single iteration and compare them with only [9]. Note that we assume the cost of operations on the plain text data is negligible compared to the ones on the encrypted data, and thus, we omit these operations.

As seen in Table 2, the privacy-preserving K-means clustering algorithm has linear complexity in the number of users similar to the original version on plain text. However, the cost of working in the encrypted domain has been significantly reduced compared to [9], which has a comparable complexity to [10]. The computational and communication gain come from the effective use of data packing, eliminating the need for an expensive secure comparison protocol in [9] and involving helper users in the computations.

The complexity analysis also shows that our proposal has lower complexity compared to the previous works in [2,12] and [14]. The communication complexity in [2] is $\mathcal{O}(N_u nKR)$ bits, and the computational complexity for the two-party setting is $\mathcal{O}(N_u KR)$ encryptions

Table 2 Communication and computational complexity for SP, H_m and user *i* for one iteration of the algorithm

	Our proposal			Algorithm in [9]	
	SP	H_m	**User**	**SP**	**User**
Encryption	$\mathcal{O}(N_u R)$	$\mathcal{O}(N_g K)$	$\mathcal{O}(1)$	$\mathcal{O}(N_u K(R+\ell))$	$\mathcal{O}(K(R+\ell))$
Decryption	-	$\mathcal{O}(N_g R)$	-	$\mathcal{O}(N_u K(R+\ell))$	-
Multiplication	$\mathcal{O}(N_u R)$	-	$\mathcal{O}(R)$	$\mathcal{O}(N_u KR)$	$\mathcal{O}(K(R+\ell^2))$
Exponentiation	$\mathcal{O}(N_u K)$	-	$\mathcal{O}(R)$	-	$\mathcal{O}(K(R+\ell^2))$
Communication	$\mathcal{O}(N_u(R+K))$	$\mathcal{O}(N_g(R+K))$	$\mathcal{O}(R)$	$\mathcal{O}(N_u K(R+\ell))$	$\mathcal{O}(K(R+\ell))$

In terms of Paillier encryptions. SP, service provider, H_m, helper user.

and multiplications for one party and $\mathcal{O}(N_u KR)$ exponentiations and multiplications for the other. [12] claims to have the same level of communication complexity with [2] but does not provide the computational complexity. [14], on the other hand, has a communication complexity of $\mathcal{O}(K^3 nR)$. The computational complexity is $\mathcal{O}(K^3 R)$ encryption and $\mathcal{O}(N_u K^3 R)$ multiplications for one party and $\mathcal{O}(K^3 R)$ exponentiation, $\mathcal{O}(K^2)$ encryption and $\mathcal{O}(N_u K^3 R)$ multiplications for the other party.

6.2 Performance analysis

To test its performance, we implemented the cryptographic algorithm in C++ using GMP Library version 4.2.1 and tested our implementation on a single computer with 8 cores and 16 GB of RAM. Table 3 gives the list of parameters used in the tests. The values for K, n and k were taken from [9] for a fair comparison of the two protocols. In our test, we only assume that random numbers that are required for encryption and blinding are generated in advance as they are not dependent on the user data. This off-line computation consists of generating random numbers and raising them to the power of n for the fast encryption afterwords. For the values in Table 3, the total time for these computations was 97 min.

Table 3 Parameters

Symbol	Value
N_u	100,000
N_g	1,562
M	64
R	12
K	10
n	1,024 bits
ℓ	10
w	3 bits
σ	40 bits

However, note that these values for better on-line performance can be generated in the idle time of the service provider.

Figure 3 shows the run-times of our proposal and the protocol from [9], for ten iterations. To compare both protocols fairly, we took $M = 1$, meaning that there is only one helper user in the system. The run-time of the protocol in [9] for $N_u > 973$ users has been estimated. It is clear that the performance difference between the two protocols is drastic: the protocol in [9] takes approximately 110 h, while our proposal takes only 2.3 h for 100,000 users. This significant difference in performance is a result of employing data packing, which reduces the number of operations on the encrypted data, and avoiding the use of an expensive secure comparison protocol, thanks to using a helper user.

While our protocol outperforms the protocol from [9] even in the case of only two helper users, a drastic performance boost is achieved when there are multiple helper users in the system. Figure 4 shows the run-time of our protocol for different numbers of helper users and a varying number of users in each group. For every choice of M, the total number of users in the system is set to 100,000. While with two helpers, it takes approximately 80 min, with 64 helper users, it takes 26 min to cluster 100,000 users. Figure 4 shows that as the number of helper users increases, the total run-time decreases first sharply and later gradually. This means that for a fixed number of users in the system, increasing the number of helper users introduces a significant gain in performance. However, after a certain number of helper users, each having a negligible amount of work to accomplish, the performance of the overall system is determined by the tasks of the service provider. This fact is clearly seen in Figure 5, where for different numbers of helper users in the system, the work share between the helper users and the service provider is given by a percentage, omitting the users' participation, which is negligible compared to the helper users and the service provider. It is clear from the figure that the number of helper users should be chosen accordingly since the computational and

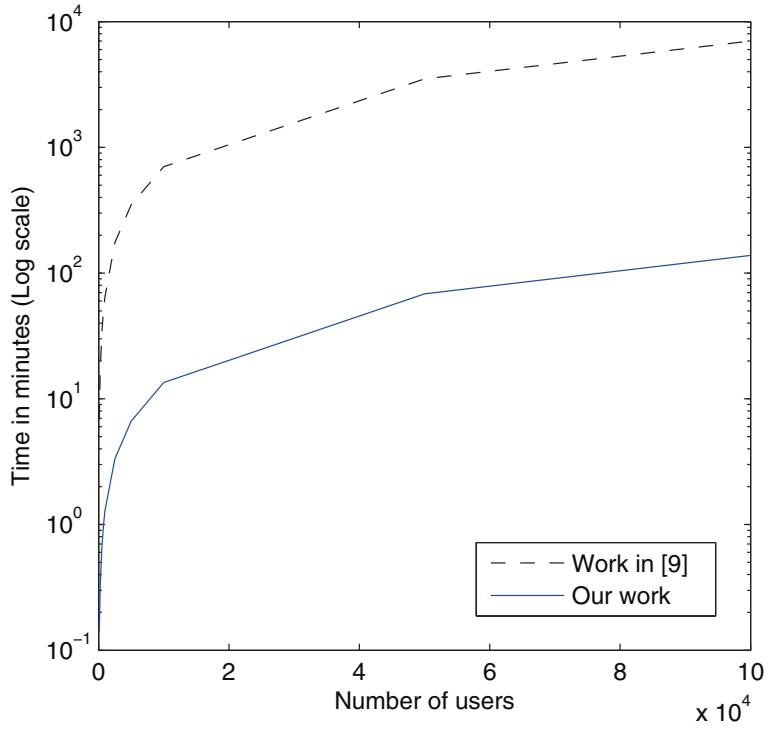

Figure 3 Performance comparison of our proposal (M=1) with [9].

communication loads become heavy with fewer helper users.

As for the bandwidth, we only consider the transmitted encrypted messages. For the parameters given in Table 3, the service provider sends and receives 1.215 GB of data, while the amount of data transmitted for a helper user is 9.4 kB. An ordinary user sends and receives only 6.8 kB of data. For the same set of parameters, the work in [9] requires the service provider and each user to transmit 7.8 GB and 82 kB of data, respectively. The significant difference in the amount of transmitted data is a result of data packing, as shown in the complexity analysis.

7 Discussion

Our proposal in this paper outperforms the most related protocol given in [9], which is also based on cryptographic tools within the semi-honest security model. Note that the privacy-preserving protocol in [10] that hides the cluster centroids from the service provider has a complexity comparable to [9]. Even though the numerical results on a data set of 100,000 show that our protocol is promising to be deployed in real life, we believe the performance of our proposal in a real implementation can be improved further for the following reasons. Firstly, an appropriate number of helper users can be determined by assessing the number of users in the system and the users' resources in terms of bandwidth and computation. This leads to a number of groups, in which the helper user can process encrypted data without disrupting the user's other activities. Secondly, after choosing the optimum number of helper users based on the aforementioned criteria, the overall performance of the privacy-preserving clustering algorithm will be determined by the performance of the service provider. Note that all operations by the service provider can completely be performed in parallel. Since a multiple server model, or a cloud, is widely used in business, the overall runtime of the privacy-preserving clustering algorithm is expected to be within reasonable boundaries in real life.

With respect to bandwidth usage, our protocol employs data packing to the fullest extent. Note that using the Paillier cryptosystem, we face data expansion by a factor of 64, assuming that 32-bit numbers become 2,048-bit cipher text after encryption. We reduce this expansion considerably by deploying data packing.

A major aspect to be considered in deploying the privacy-preserving K-means clustering algorithm in real life is the security assumptions. Our model assumes the honest participation of all parties. While the semi-honest security model can be considered too simplistic, it is still good enough for real-world applications where the service provider and the users have incentives to act according to the protocol as seen in the sugar beet auction system

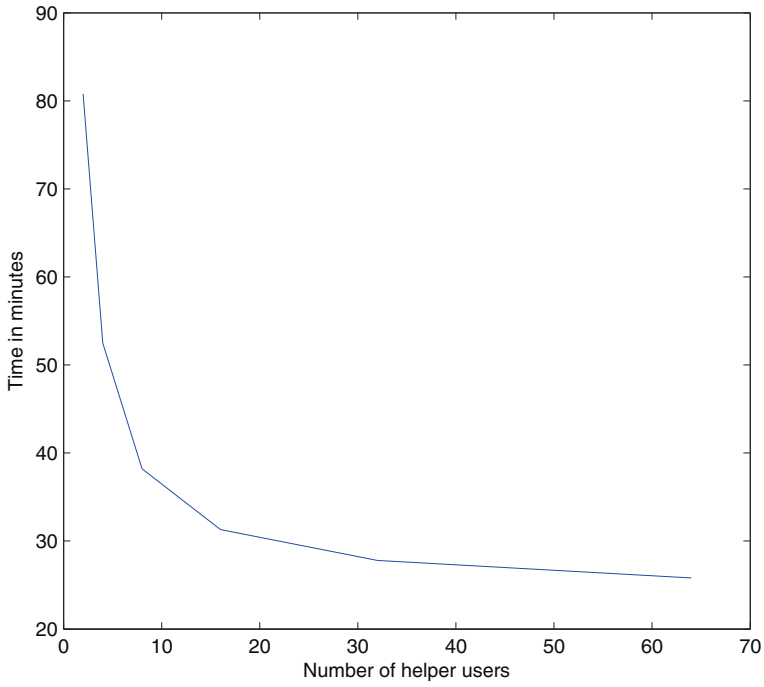

Figure 4 Run-time of the privacy-preserving algorithm with respect to different numbers of helper users.

[24]. Note that a protocol with 'proper measures against malicious parties' will be *much more expensive* computationally and hence impractical for large-scale deployment. A protocol with less strict but still realistic security guarantees is therefore preferred. To that end, we distribute the trust of the system between multiple parties, preventing a single malicious party to learn sensitive data. In a distributed model consisting of independent database owners, security risks are smaller because a collusion between the service provider and one of the helper users

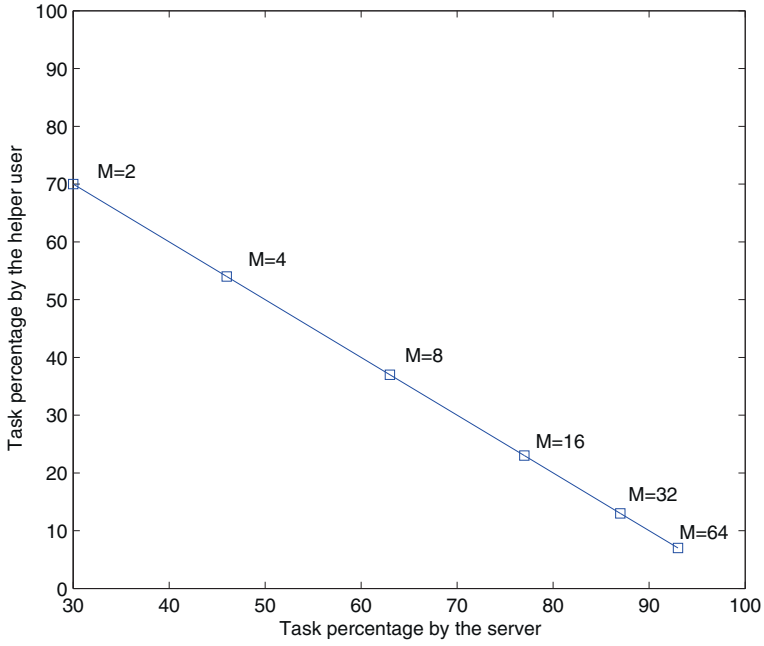

Figure 5 Work share between the service provider and different numbers of helper users.

will be less likely. A second aspect to consider is the active participation of all users in the system. It is our conclusion that without introducing (semi) trusted third parties, users' data cannot be processed without their participation. Fortunately, due to our construction, only the helper user needs to be on-line during the clustering procedure. Once the encrypted data are sent to the service provider, users can go off-line for the rest of the computation. If the same helper users are to be used, other users can stay off-line not only during that iteration but during the whole clustering; however, this would lead to minor changes in the protocol such as the encrypted distances should be computed by the service provider. Note that using the same helper users will lead to a similar setting to [10] with dedicated key holders. However, it is our motivation to distribute trust among multiple random helper users in each iteration for privacy protection, which requires helper users to be on-line during each iteration.

8 Conclusion

In this paper, we present an efficient, privacy-preserving K-means clustering algorithm in a social network setting. We present a mechanism where the private data of the users, sensitive intermediate values and the final clustering assignments are protected by means of encryption. The service provider, who does not have the decryption key, can still perform clustering without being able to access the content of private data. While the approach of processing encrypted data presents a concrete privacy protection for the users, it also introduces performance drawbacks compared to the version with plain text due to data expansion after encryption and expensive operations on the encrypted data. Previous work has shown different approaches to reduce the complexity of privacy-preserving K-means clustering such as using semi-trusted third parties. In this work, we build a mechanism on the common server-client model and reduce the costs by employing data packing. By this way, we reduce the number of encryption by a factor of K, thus introducing a considerable gain in terms of communication and computation. We also avoid interactive protocols such as secure comparison by exploiting the distributive setting. We also distribute trust among multiple random users for each iteration of the protocol, which introduces a computational gain proportional to the number of such users. The resulting cryptographic protocol is significantly more efficient compared to previous work in the semi-honest security model. We also analyze the effects of different choices of parameters on the performance of the cryptographic protocol. Experimental results support our claim on the feasibility of privacy-preserving K-means clustering such that it takes 26 min to cluster 100,000 users. This result, which can be improved further on a real system, encourages the deployment of privacy-preserving K-means clustering algorithms based on homomorphic encryption.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
This work is supported by the Dutch COMMIT programme.

Author details
[1]Department of Intelligent Systems, Delft University of Technology, Delft, 2628 CD, The Netherlands. [2]TNO, P.O. Box 5050, Delft, 2600 GB, The Netherlands. [3]Computer Science Department, Aarhus University, IT-Parken, Åbogade 34, Aarhus N 8200, Denmark.

Received: 23 July 2013 Accepted: 31 October 2013
Published: 9 November 2013

References

1. Y Lindell, B Pinkas, Privacy preserving data mining, in *CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology* (Springer, London, 2000), pp. 36–54
2. G Jagannathan, Wright R N, Privacy preserving distributed K-means clustering over arbitrarily partitioned data, in *KDD '05: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining* (ACM, New York, 2005), pp. 593–599
3. R Lagendijk, Z Erkin, M Barni, Encrypted signal processing for privacy protection: encrypted signal processing for privacy protection. IEEE Signal Process. Mag. **30**(1), 82–105 (2013)
4. Z Erkin, T Veugen, T Toft, RL Lagendijk, Generating private recommendations efficiently using homomorphic encryption and data packing. IEEE Trans. Inf. Forensics Secur. **7**(3), 1053–1066 (2012)
5. K Fukunaga, *Introduction to Statistical Pattern Recognition* (Academic, San Diego, 1990)
6. O Goldreich, *Foundations of Cryptography II* (Cambridge University Press, Cambridge, 2004)
7. JR Troncoso-Pastoriza, S Katzenbeisser, MU Celik, AN Lemma, A secure multidimensional point inclusion protocol, in *ACM Workshop on Multimedia and Security* (ACM Dallas, 2007), pp. 109–120
8. T Bianchi, A Piva, M Barni, Composite signal representation for fast and storage-efficient processing of encrypted signals. IEEE Trans. Signal Process. **5**(1), 180–187 (2009)
9. Z Erkin, T Veugen, T Toft, R Lagendijk, Privacy-preserving user clustering in a social network, in *1st IEEE Workshop on Information Forensics and Security (WIFS09)* (IEEE, London, 2009), pp. 96–100
10. M Beye, Z Erkin, R Lagendijk, Efficient privacy preserving k-means clustering in a three-party setting, in *IEEE Workshop on Information Forensics and Security (WIFS '11)*. (Foz do Iguaçu, 29 Nov–2 Dec 2011)
11. R Agrawal, R Srikant, Privacy-preserving data mining, in *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Volume 29(2)* (ACM, New York, 2000), pp. 439–450
12. P Bunn, R Ostrovsky, Secure two-party k-means clustering, in *Proceedings of the 14th ACM Conference on Computer and Communications Security* (ACM, New York, 2007), pp. 486–497
13. C Clifton, M Kantarcioglu, J Vaidya, Tools for privacy preserving distributed data mining. ACM SIGKDD Explorations **4**(2), 28–34 (2003)
14. G Jagannathan, K Pillaipakkamnatt, R Wright, A new privacy-preserving distributed k-clustering algorithm, in *Proceedings of the Sixth SIAM International Conference on Data Mining*. (Bethesda, 20–22 Apr 2006)
15. S Oliveira, O Zaiane, Privacy preserving clustering by data transformation, in *Proceedings of the 18th Brazilian Symposium on Databases*. (Manaus, 6–10 October 2003, pp. 304–318)
16. S Oliveira, O Zaiane, Achieving privacy preservation when sharing data for clustering, in *Secure Data Management,* ed. by W Jonker, M Petković. Proceedings of the VLDB 2004 Workshop, SDM 2004, Toronto, Canada, August 30, 2004. Lecture Notes in Computer Science, vol. 3178 (Springer, Berlin, 2004), pp. 67–82

17. ACC Yao, How to generate and exchange secrets (extended abstract), in *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science* (IEEE, Toronto, 1986), pp. 162–167
18. Z Huang, W Du, B Chen, Deriving private information from randomized data, in *SIGMOD '05: Proceedings of the 2005, ACM SIGMOD International Conference on Management of Data* (ACM, New York, 2005), pp. 37–48
19. H Kargupta, S Datta, Q Wang, K Sivakumar, On the privacy preserving properties of random data perturbation techniques, in *Proceedings of the ICDM 2003* (IEEE, Melbourne, 2003), pp. 99–106
20. V Kolesnikov, AR Sadeghi, T Schneider, Improved garbled circuit building blocks and applications to auctions and computing minima, in *Cryptology and Network Security,* ed. by JA Garay, A Miyaji, A Otsuka. Proceedings of the 8th International Conference, CANS 2009, Kanazawa, Japan, December 12–14, 2009. Lecture Notes in Computer Science, vol. 5888 (Springer, Berlin, 2009), pp. 1–20
21. D Kononchuk, Z Erkin, JCA van der Lubbe, RL Lagendijk, Privacy-preserving user data oriented services for groups with dynamic participation, in *Computer Security – ESORICS 2013,* ed. by J Crampton, S Jajodia, K Mayes. Proceedings of the 18th European Symposium on Research in Computer Security, Egham, UK, September 9–13, 2013. Lecture Notes in Computer Science, vol. 8134 (Springer, Berlin, 2013), pp. 418–442
22. N Doraswamy, D Harkins, *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks* (Prentice-Hall, Upper Saddle River, 1999)
23. Paillier P, Public-key cryptosystems based on composite degree residuosity classes, in *Advances in Cryptology — EUROCRYPT '99,* ed. by J Stern. Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999. Lecture Notes in Computer Science, vol. 1592 (Springer, Berlin, 1999), pp. 223–238
24. P Bogetoft, DL Christensen, I Damgård, M Geisler, TP Jakobsen, M Krøigaard, JD Nielsen, JB Nielsen, K Nielsen, J Pagter, MI Schwartzbach, T Toft, Multiparty computation goes live. IACR Cryptology ePrint Arch. **2008**, 68 (2008)

doi:10.1186/1687-417X-2013-4
Cite this article as: Erkin *et al.*: Privacy-preserving distributed clustering. *EURASIP Journal on Information Security* 2013 **2013**:4.