PRIVACY-PRESERVING USER CLUSTERING IN A SOCIAL NETWORK

Zekeriya Erkin\textsuperscript{1}, Thijs Veugen\textsuperscript{1,2}, Tomas Toft\textsuperscript{3}, Reginald L. Lagendijk\textsuperscript{1}

\textsuperscript{1}Multimedia Signal Processing Group, Delft University of Technology, The Netherlands
\textsuperscript{2}TNO Information and Communication Technology, Delft, The Netherlands
\textsuperscript{3}Aarhus University, Denmark

ABSTRACT

In a ubiquitously connected world, social networks are playing an important role on the Internet by allowing users to find groups of people with similar interests. The data needed to construct such networks may be considered sensitive personal information by the users, which raises privacy concerns. The problem of building social networks while user privacy is protected is hence crucial for further development of such networks. K-means clustering is widely used for clustering users in a social network. In this paper, we provide an efficient privacy-preserving variant of K-means clustering. The scenario we consider involves a server and multiple users where users need to be grouped into K clusters. In our protocol the server is not allowed to learn the individual user data and users are not allowed to learn the cluster centers. The experiments on the MovieLens dataset show that deployment of the system for real use is reasonable as its efficiency even on conventional hardware is promising.

\textbf{Index Terms—} SEC-PRIV, SEC-INTE, CRY-ENCR

1. INTRODUCTION

Internet applications in which people are grouped based on personal preferences have become very popular. By grouping users, these applications provide personalized services as well as building social networks where people can find the opportunity to communicate with others who share similar interests. There are a vast amount of social networks now available for dating, traveling, reading, cultural activities and many more [1]. Most users tend to give privacy sensitive data to benefit from such applications. As in the case of dating sites, the users provide to the system their personality details along with their preferences for a candidate while in traveling networks, the users announce a list of dates and locations for their planned travels.

The very success of applications based on finding similar people depends on the accuracy of grouping users which is directly proportional to the amount of collected user data. Since the content of the data is mostly privacy sensitive, the protection of the data is a raising concern among users [2]. Many rely on the trustworthiness of the service provider that possesses all the data. Several incidents have shown that this assumption is not completely true [3]. Even if the service provider protects its database against a common security problem of identity theft, there is no guarantee to prevent information being passed on without consent. A possible solution to protect the privacy sensitive data is having a trusted third party that is fully trusted by both the user and the service provider that keeps the data and runs the algorithm instead of the service provider. Unfortunately, having a third party that is fully trusted and willing to do all bulky computations is not realistic. A genuine solution is deploying cryptographic protocols to protect the privacy sensitive data of the users. Assuming that the server and the users are semi honest, meaning that they follow the protocol steps but are curious to extract more information than they allowed to have by storing all previous messages, it is possible to have a secure system where no information is revealed except the result of the algorithm run. With such a design, identity theft and abuse of user data by the service provider will be unlikely without having the secret key that is used to secure the user preferences.

A closer look at the problem of grouping users in a social network leads us to a well-known problem of clustering data. A user can be attached to a group of users if the user shares a common taste as of the users in that group. In a social network, the preference of a user is represented by a vector in the feature space. Thus, finding similar users with the same taste is basically a problem of clustering these feature vectors. The goal of the secure system is, then, grouping users with the same taste while protecting their privacy by hiding their preference data or feature vector. At the same time, the server should protect sensitive information about the algorithm like cluster locations. A malicious user can place himself into a desired cluster if this information is known. At the end of the secure clustering protocol, a user should only obtain the label information which is in fact a pointer to the cluster he is in, and the server should not get any information on the feature vector of the users.

As a method of clustering data, the K-means algorithm is widely used because of its simplicity and ability to converge extremely quickly in practice. Hence, in [4, 5, 6, 7] the authors addressed cryptographic techniques for the privacy-preserving clustering protocols based on K-means algorithm. In these works, the authors apply secure multiparty computation techniques [8], which makes any two-party privacy-preserving data mining problem solvable mostly by using Yao’s secure circuit evaluation method [9]. Even though Yao’s method can be used to implement any function in a privacy preserving manner, heavy computation costs in such circuits make these solutions feasible only for small circuit sizes which is a difficult requirement in many application scenarios. In [4, 5, 6], the authors attempt to solve the clustering problem in a two-party setting which is suitable to deploy techniques based on secret sharing. [5] suffers from a problem during the clustering algorithm where a division operation is misinterpreted as multiplication by the inverse which is not correct. On the other hand, [7] has a multi-user 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 possess the intermediate centroid assignments,
meaning more information leakage. Therefore, these proposals are either not suitable for the problem of clustering users in a social network as they have different setting or not secure and efficient enough to deploy in practice.

In this paper, we provide a solution based on secure multi-party computation techniques in a semi-honest environment. Within this setting, our proposal groups people in a social network while protecting their privacy-sensitive data against the server and other users by means of encryption. A user gets a cluster identifier at the end of protocol but nothing more while the server obtains neither the identity of the users nor the content of user data. Our proposal provides a solution that is computationally efficient and scalable to a real life scenario of a centralized social network. We also show that communication cost of our protocol reaches the same performance of the most similar work in the field but achieving more privacy. The overall protocol was also implemented and tested exhaustively on the MovieLens dataset. Experimental results show that the algorithm proposed in this paper is both reliable and efficient for practical use.

2. PRIVACY-PRESERVING CLUSTERING

Data clustering is a common technique for statistical data analysis where data is partitioned into smaller subgroups with its members sharing a common property [10]. Particularly, each user is represented as a point in an R dimensional space and is clustered according to minimal Euclidean distance. As a very common clustering technique, K-means assigns each user \( P_i = (p_{i,1}, \ldots, p_{i,R}) \) to the closest cluster among \( K \) clusters \( C = \{C_1, \ldots, C_K\} \) where \( C_j = (c_{j,1}, \ldots, c_{j,R}) \). The algorithm starts with choosing the constant value \( K \) which is the number of clusters in the feature space. Each cluster is represented by its center (also named centroid) which is initially a random point in the space. In every iteration, the distance \( D_{i,j} \) between \( i^{th} \) user \( P_i \) and cluster center \( C_j \) for \( j = 1 \) to \( K \) are calculated and the user is assigned to the cluster with the minimal distance. Once every user is assigned to a cluster, centroid locations are recalculated by taking the arithmetic mean of the user locations within each cluster. These two steps are repeated until either a certain number of iterations is reached or centroid locations are more or less fixed.

In the privacy-preserving version of the K-means clustering algorithm (Algorithm 1), each step is implemented in the encrypted domain. To realize this system, the server is assumed to have key pairs for himself of the Paillier [11] and Damgård, Geisler and Kroigaard (DGK) [12, 13] cryptosystems. These cryptosystems are chosen as they possess a property called additive homomorphism that allows us to process data in the encrypted domain such that the product of two encrypted values \([a] \) and \([b] \), corresponds to a new encrypted message whose decryption yields the sum of \( a \) and \( b \) as \([a] \cdot [b] = [a+b] \).

As a consequence of this additive homomorphism any ciphertext \([a] \) can be raised to the power \( b \) to obtain the encryption \([a]^b = [ab] \).

In addition to the homomorphism property, Paillier and DGK cryptosystems are semantically secure implying that each encryption has a random element that results in different ciphertexts for the same plaintext. Throughout this paper we denote the Paillier encryption of a message \( m \) by \([m] \) and DGK encryption by \([m] \).

2.1. Computing Encrypted Distances

Assigning a user to the closest cluster requires Euclidean distance computations between a user \( P_i \) and centroid \( C_j \) in an R dimensional space as given in Equation 1. Regarding that the distance computations are only used for determining the minimum distance, taking the square root can be omitted.

\[
D^2_{i,j} = \|P_i - C_j\|^2 = \sum_{n=1}^{R} (p_{i,n} - c_{j,n})^2 \\
= \sum_{n=1}^{R} p_{i,n}^2 + \sum_{n=1}^{R} (-2p_{i,n}c_{j,n}) + \sum_{n=1}^{R} c_{j,n}^2. \tag{1}
\]

The implementation of the distance computation in the encrypted domain has two steps. First, the server encrypts \((-2\text{to})\) times its centroid locations with his public Paillier key to obtain \([-2c_{j,n}] \) for all \( j \) and \( n \), and publishes them. Then, the user calculates the encrypted Euclidean distance to each centroid as follows: the user computes the sum in first term in Equation 1 and encrypts it. In order to compute the encrypted second term, the additive homomorphism property of the Paillier cryptosystem is used. The user simply needs to raise each encrypted centroid value \([-2c_{j,n}] \) to the power of \( p_{i,n} \) being the user’s location in the \( n^{th} \) dimension. These values are then multiplied. Note that by receiving \([-2C] \) instead of \([C] \), the user spends less time for the costly exponentiation as he only uses \( p_{i,n} \) as the power rather than \(-2(p_{i,n}) \). The calculation of the last term requires encryptions of the squares of the centroids. As all needed values are known to the server, it simply supplies these encryptions, \([[\sum_{n=1}^{R} c_{1,n}^2], \ldots, [\sum_{n=1}^{R} c_{K,n}^2]] \), along with the encryptions of \([-2C] \). Finally, the user multiplies these values to obtain the encrypted distance \([D^2_{i,j}] \) as given in Equation 2.

\[
[D^2_{i,j}] = \prod_{n=1}^{R} [p_{i,n}^2] \cdot \prod_{n=1}^{R} [-2c_{j,n}]^{p_{i,n}} \cdot \prod_{n=1}^{R} [c_{j,n}^2]. \tag{2}
\]

At the end of this step, each user possesses the encrypted distances \([D_{i,1}], \ldots, [D_{i,K}] \) from his location \( P_i \) to the current K centroids.

2.2. Preparing user data

After having obtained the encrypted distances to each centroid \([D_{i,1}], \ldots, [D_{i,K}] \), the \( i^{th} \) user needs to find the minimum of these

---

**Algorithm 1 The Privacy-preserving K-means clustering algorithm.**

**Require:** The server sets parameter \( K \) and selects \( K \) random points as the initial centroids.

**Ensure:** A cluster pointer to the user.

1. The user computes encrypted distances to the \( K \) current centroids.
2. The server and the user run an interactive protocol to find the minimum distance of \( K \) encrypted distances to each centroid.
3. The server and all users jointly update the centroid locations.
4. Repeat step (1), (2) and (3) until the server finds that one of the termination conditions is reached.
5. The server and the user run a final protocol to reveal the cluster label to the user.
2.3. Updating Centroids

Once all users complete their calculation on forming their encrypted vector $[X_i]$ and encrypted matrix $[Y_i]$, they jointly start a protocol for updating the centroids. For this step, the server creates a user chain as illustrated in Fig. 1. We will explain the procedure for matrices $Y_i$, the accumulation of vectors $X_i$ will be similar, and even simpler because the elements of $X_i$ only take single bits.

Each user generates a random number $r$ for each value in the matrix $Y_i$, to be used as blinding factors so the server will not learn the value of individual matrices. Actually for each user $i$, $R_i$ is a $K$ by $R$ matrix of random values $(R_i)_{j,n}$. The matrix $R_i$ is sent to the left neighbour of the user chain. Each user computes $(R_i)_{j,n} - (R_{i-1})_{j,n}$ as a blinding value for $(Y_i)_{j,n}$. Since the server will compute the sum $Y_{j,n}^{sum} = \sum_{i=1}^{M} (Y_{i})_{j,n}$, these random values will eventually cancel out. Note that the first user computes $(R_1)_{j,n} - (R_{M})_{j,n}$, $M$ being the number of users.

Suppose that user data $p_{i,n}$ is at most $k$ bits. Then for each centroid $j$ and user data index $n$, the sum $Y_{j,n}^{sum}$ can maximally take $k' = k + \lceil \log M \rceil$ bits. In order to sufficiently blind (a subchain of) the matrix elements, the random numbers should also be of size $k'$. In order to keep the blinding factors uniformly distributed, each user should compute $R_0 - R_{i-1}$ modulo $2^{k'}$, before adding these to $Y_i$. Since the matrix $Y_i$ is encrypted, the user cannot compute $Y_i + (R_0 - R_{i-1})$ modulo $2^{k'}$. Therefore, an extra random number $r'$ (or actually an extra matrix $R'$ of random numbers) is needed to mask a possible overflow modulo $2^{k'}$. This extra random number should be $k$ bits where $k$ is a security parameter, to sufficiently mask the overflow to the server. Equation 4 shows the complete value $(Y'_i)_{j,n}$ that is sent to the server in encrypted form.

\[
(Y'_i)_{j,n} = ([Y_i]_{j,n} \cdot 2^{k'}(R'_i)_{j,n} + ((R_i)_{j,n} - (R_{i-1})_{j,n}) \mod 2^{k'}) \mod 2^{k'}, \quad (4)
\]

The server will compute the matrix $[Y'^{sum}]$ by adding all matrix elements $[(Y'_i)_{j,n}]$ over all users $i$. Although the server could first decrypt the matrix elements, it would be more efficient to use the homomorphic property of Pallier:

\[
[Y'^{sum}]_{j,n} = \prod_{i=1}^{M} ((Y'_i)_{j,n}), \quad (5)
\]

The server can simply compute the actual sum $Y^{sum}$ by decrypting $[Y'^{sum}]$ and computing $Y^{sum}$ modulo $2^{k'}$ as shown in Equation 5. This is the sum of all user points per cluster.

A similar procedure is followed for the server to obtain $X^{sum}$, which is the number of users per cluster. The sum simply counts the number of ones, i.e. the number of users assigned to each cluster. The server can then update the centroids by computing $\epsilon_{j,n} = Y^{sum}_{j,n} / X^{sum}_{j}$ and rounding the result to the nearest integer.

2.4. Termination Control and Getting User Labels

At the end of each iteration the server checks whether the predetermined termination condition is reached. Since centroids locations and the number of iterations are known to the server, this control is considered to be costless. Once the termination condition is reached, the label information of the user which is the index of the non-zero element in the encrypted vector $[X_i]$ should be revealed to the user. For this purpose, each user performs the following operation to obtain his cluster label information:

\[
[\text{Id}] = \prod_{j=1}^{K} (x_{i,j} \times j), \quad (6)
\]

where $\text{Id}$ represents the cluster number that the user belongs to. Next, the user additively blings this encrypted value with an uniformly random element $r$ of size $\log (K) + \kappa$ to get $[\text{Id} + r]$ and re-randomize it before sending it to the server to be decrypted. The user can easily obtain his corresponding cluster label by subtracting $r$ from the decrypted value sent by the server. This step completes the privacy-preserving K-means clustering algorithm.

3. COMPARISON PROTOCOL

The most important building block in our protocol for privacy preserving K-means clustering is a cryptographic protocol that com-
parses two encrypted $\ell$ bit values $[a]$ and $[b]$ and returns the minimum of these two values encrypted along with the result of the comparison $[\lambda]$ where $\lambda = 1$ if $a \geq b$ and 0 otherwise. Instead of using Yao’s garbled circuit approach [9], which is often used and generally computationally expensive, in [14] a specialized fine-tuned protocol for this task is developed.

Having the comparison protocol in [14] at the core of our protocol, we build a binary tree for the values to be compared (see Section 2.2) as illustrated in Figure 2. We assume that $K$ is a power of two. If this is not the case, dummy values can be added to the list of values to be compared. For $K$ values, $K - 1$ comparison results are stored by the user. When the comparisons are complete, each $[\lambda_{i,j}]$ is converted to DGK cryptosystem as this minor change improves the efficiency of the consequent computations considerably compared to using Paillier cryptosystem. For this conversion, the user computes a number $[\lambda_1]$ composed of $[\lambda_{i,j}]$ as follows:

$$[\lambda_1] = \prod_{j=1}^{K-1} [\lambda_{i,j}]^{2^j},$$

(7)

where $[\lambda_{i,j}']$ is either $[\lambda_{i,j}]$ or $[1 - \lambda_{i,j}]$ with probability 0.5 to hide the comparison results from the server. Upon receiving the value $[\lambda_1]$, the server decrypts it and encrypts every bit value $X_{i,j}$ using DGK cryptosystem to obtain $[\lambda_{i,j}']$ and sends them back to the user. The user then reverses the hiding procedure by comparing either $[\lambda_{i,j}]$ or $[1 - \lambda_{i,j}]$ to obtain the correct values. After this conversion between cryptosystems, the user assigns the values $[\lambda_{i,j}]$ and $[1 - \lambda_{i,j}]$ to the left and right branches of each node of the tree respectively (Figure 2). Next, the user traverses the tree from root to top to reach each leaf while adding up the branch values which corresponds to the multiplication of the encrypted values. At the end of this procedure, we obtain an encrypted value $[\sigma_{i,j}]$ for each leaf, thus $D_{i,j}$. For only the minimum $D_{i,j}$ the value $\sigma_{i,j}$ is zero since all the branch values in the path should be zero. For the others, $\sigma_{i,j}$ is a non-zero value.

$$[D_{i,1}] \quad [D_{i,2}] \quad \cdots \quad [D_{i,k-1}] \quad [D_{i,K}]$$

$$[\lambda_{i,1}] \quad [1 - \lambda_{i,1}] \quad [\lambda_{i,K/2}] \quad [1 - \lambda_{i,K/2}]$$

$$([\lambda_{i,1}, [\text{min}_{i,1}]] \quad ([\lambda_{i,K/2}, [\text{min}_{i,K/2}]]$$

$$([\lambda_{i,K-3}, [\text{min}_{i,K-3}], [\lambda_{i,K-2}, [\text{min}_{i,K-2}]]$$

$$[\lambda_{i,K-1}] \quad [1 - \lambda_{i,K-1}]$$

$$([\lambda_{i,K-1}, [\text{min}_{i,K-1}]]$$

Fig. 2: Binary tree used to form user vector $X_i$.

This set of values is then re-randomized, and permuted with a uniformly random permutation $\pi$ before sending them to the server. The server decrypts the values and creates a new vector that only has a one value at the same position of zero in the received vector and zeros everywhere else. This vector is then encrypted item-wise with Paillier public key and sent back to the user. After repermutation, it is used as the $X_i$ vector described in Section 2.2.

4. EXPERIMENTS

The privacy-preserving K-means clustering algorithm presented in the paper has been implemented in order to determine its performance and reliability. This has been done in C++ using the GNU GMP library version 4.2.1. The tests were performed on a computer with Intel Xeon 2.33 GHz processor and 32GB of RAM running SuSE 10.3 operating system. Both server and user, modeled as separate classes, run on the same machine, thus network latency was not considered in the test results.

The MovieLens dataset (www.grouplens.org) was used for our experiments. This contains 100,000 integer ratings in the range of [0, 5] for 1682 movies by 943 users. As the sparseness of the dataset is great (94%), a subset containing the movies rated by most users was considered. We filled the null entries of this subset with the user mean vote rounded to the nearest integer value for the corresponding row. The number of movies in this subset, represented by $R$, also determines the parameter $\ell$ which is the bit length of the values to be compared. The parameter $\ell$ should be big enough to hold the largest possible value which is the Euclidean distance squared between two user rating vectors in our case. Since our aim is to show the equivalent accuracy between the plain and privacy-preserved K-means clustering algorithm, we set the number of clusters $K$ to a single value. The parameters used for the experiments are given in Table 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>12</td>
</tr>
<tr>
<td>$K$</td>
<td>10</td>
</tr>
<tr>
<td>$\ell$</td>
<td>9 bits</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>112 bits</td>
</tr>
<tr>
<td>Paillier Encryption</td>
<td>2048 bits</td>
</tr>
<tr>
<td>DGK Encryption</td>
<td>1024 bits</td>
</tr>
</tbody>
</table>

4.1. Reliability

The privacy-preserving K-means clustering protocol is designed for an accuracy equivalent to the plain clustering algorithm. The only possible degradation is due to the integer arithmetic used for updating centroids. As the cryptosystem used in our protocol accepts only integer values, the location of the centroids in the space should also be represented by integers. This can be achieved by scaling and rounding the values, however, in our application scenario we experienced that using scaling does not introduce noticeable improvement that can effect the outcome of the clustering protocol. Therefore, we only round the values to the nearest integer.

4.2. Round Complexity

The distance computation and the updating of the centroids both require one round, while each comparison require four. As the minimum of $K$ values can be found in $O(\log(K))$ rounds by using binary tree approach as illustrated in Figure 2, the overall round complexity of one iteration of the privacy-preserving K-means clustering protocol is $2 + 4\log(K)$. In addition, one extra round is required at the end of the clustering protocol to send the final labels to the users. The round complexity of our work outperforms the comparable work of Vaidya and Clifton [7] which has a round complexity of $O(M + K)$ in their basic algorithm and $O(M)$ in optimized version.
4.3. Communication Complexity

Communication complexity is mainly determined by the amount of encrypted messages exchanged between the server and the users. Therefore, it is related to the size of the Pailler and DGK encryptions. The communication complexity is $O(K(R+\ell))$. Considering that $\ell = \log_2(R \times c)$ where $c$ is the possible maximum distance-squared between two rating vectors of size $R$, the overall communication cost can be written as $O(KR)$.

The amount of data sent by the server and the user is 48 kB each for one iteration using the parameters given in Table 1.

As [4, 5, 6] have proposals for a two-party setting based on secret sharing, we can compare our result only to [7] which has the same communication cost of $O(KR)$ bits. In addition to achieving the same level of communication cost, in our proposal, we keep intermediate centroid locations hidden from the users and have no need for particular non-colluding entities, meaning better privacy.

4.4. Computational Complexity

The computational complexity of our privacy-preserving K-means clustering protocol is mainly dependent on the Pailler and DGK cytosystems. In one iteration of the clustering, the total computational complexity is $O(KR)$ Pailler encryptions, exponentiations and multiplications and $O(K\ell)$ DGK encryptions and $O(K\ell^2)$ DGK multiplications for the user. The server needs to compute $O(K + R)$ Pailler encryptions, $O(KR)$ Pailler decryptions and $O(MKR)$ Pailler multiplications. In addition to that, the server also computes $O(K\ell)$ DGK encryptions and decryptions.

The running time of our implementation is given in Table 2 with a different number of users for 10 iterations. As given in the complexity analysis, for constant $K$ and $R$ the running time is linear in the number of users $M$ in the system and it only takes roughly 1 hour to cluster 943 users.

![Table 2: Computational Complexity (in minutes).](image)

The second column of Table 2 reflects the running time of the whole protocol without any optimization. The random values used for encryption and blinding and the exponentiations $r^n$, $h^r$ that are used in Pailler and DGK encryptions respectively can be generated in idle processor time or prior to the start of the protocol. The running time of the protocol under these optimizations is given in the third column of Table 2. These values will be much smaller in a real system where there are at least $M + 1$ processors and many operations can be realized asynchronously.

5. CONCLUSION

In this paper we present an efficient, privacy-preserving K-means clustering algorithm in a social network setting. In particular, we propose a protocol in which privacy sensitive data of the users is kept hidden from the server and the cluster locations in the user space are kept secret from the users. Our protocol mainly uses secure multiparty computation techniques, but instead of using generic solutions, it benefits from fine-tuned cryptographic protocols developed for higher efficiency. Our proposal achieves more privacy by hiding all sensitive user data from the server and the centroid locations from the users with the same level of communication cost of [7] which has a comparable multi-user setting. The implementation of the privacy-preserving K-means clustering algorithm with MovieLens dataset shows that the accuracy of the system is as reliable as the reference implementation in clear and the running time of the protocol is promising even on modest hardware platforms. The numbers we have obtained from the experiments show that the protocol presented in this paper is efficient enough to be deployed in practice.

Acknowledgements The authors would like to thank David Tax for discussions on clustering methods and Martin Franz for his permission to use his DGK cryptosystem implementation.

6. REFERENCES


