World Wide Web (2015) 18:1481–1517
DOI 10.1007/s11280-014-0290-4

Privacy preserving graph publication in a distributed environment

Mingxuan Yuan · Lei Chen · Philip S. Yu · Hong Mei

Received: 26 September 2013 / Revised: 23 February 2014 /
Accepted: 10 March 2014 / Published online: 1 June 2014

Abstract Nowadays, more and more people join different social networks to share or comment on their daily activities. Along with the popular usage of social networks, people's privacy becomes a big concern. Therefore, recently, many works studied how to publish privacy preserving social networks for "safely" data mining or analysis. These works all assume that there exists a single publisher who holds the complete graph. While, in real life, people join different social networks for different purposes. As a result, there are a group of publishers and each of them holds only a subgraph. Since no one has the complete graph, it is a challenging problem to generate the published graph in the distributed environment without releasing any publisher's local content. In this paper, we propose an SMC (Secure Multi-Party Computation) based protocol to publish a privacy preserving graph in a distributed environment. We prove that our scheme can publish a privacy preserving graph without leaking the local content information and meanwhile achieve the maximum graph utility. We show the effectiveness of the protocol on real social networks under different distributed storage cases.

Keywords Soical network · Privacy

M. Yuan
Huawei Noah Ark Lab, Hong Kong, China

M. Yuan · L. Chen (✉)
The Hong Kong University of Science and Technology, Hong Kong, China
e-mail: leichen@cse.ust.hk

P. S. Yu
University of Illinois at Chicago, Chicago, USA

H. Mei
Peking University, Bei Jing, China

1 Introduction

Recently, privacy preserving graph publication has become a hot research topic since information published on the Web can be combined together to break the privacy in a graph. As a result, many privacy preserving graph models have been proposed to protect the privacy of the graph [4, 5, 9, 11, 15, 20, 25, 27–29]. All these models assume there is a trustable centralized data source, which has a complete original graph G, and can directly generate the privacy preserving graph G' from G.

However, in reality, people join different social networks due to different interests or purposes. For example, a person uses Facebook to share his information with his classmates and coworkers. At the same time, he may also build his blog on a blog server to share his interests with others. As a result, his connection information is stored in two social networks. A consequence of this joining preference is that each social network website only holds partial information (a subgraph) of the complete social network G. We call such a social network website as a data agent. Consider a social graph as shown in Figure 1 where each person in the graph has two labels and people are involved in different kinds of interactions, the graph can be stored on three data agents, A_1, A_2 and A_3 separately as shown in Figure 2.

Although people join different networks, it is often necessary to obtain a privacy preserving graph generated from the complete graph for criminal investigation [19] or mining useful patterns/influential persons [11, 13, 19]. This requests that the distributed agents should cooperatively generate a privacy preserving graph G'. There are three potential approaches [17] to jointly publish privacy preserving relational data in a distributed system. For graph data, similar approaches can be implemented as shown in Figure 3.

A Third-Party approach (Figure 3a) is to let all the data agents send their local contents to a trustable third party agent. This third party agent generates the published graph by integrating all the data. However, since the data of each site is its most valuable asset, no one is willing to share its data with others, finding such a trustable third party data agent is not feasible in the real world [17]. A crash or a compromise on the third party agent by attackers could lead to a complete privacy loss. Thus, the Third-Party approach is unsuitable to generate a privacy preserving graph in a distributed environment.

The naive approach (Figure 3b) is to let each data agent generate a privacy preserving graph based on its local content and *securely combine* these graphs into a large privacy preserving graph.[1] Secure combination means no local content is released during the process. However, this approach encounters two problems. The first is that it results in the low quality of the published data since the graph is constructed only based on local information instead of the complete graph. The second problem is that most graph protection models [4, 7, 20, 28, 29] need to consider the connections between nodes. A privacy preserving graph generated only by local information may violate the privacy requirement when globally more connection information is provided. The secure combination needs to delete some connections to ensure the final published graph satisfies the privacy requirement. Therefore, the published graph will have incomplete information of the original graph.

Another approach (Figure 3c) is that each data agent participates in a protocol to produce a privacy preserving graph. The privacy preserving graph is generated just as there virtually exists an agent who has the integrated data. During the computation, except the content

[1] The overlapping between subgraphs should be considered. [12] showed it is not safe even when each publisher publishes privacy preserved data independently if their data is overlapping. A solution based on naive approach can be found in Section 5.1. Here we use Figure 3 to show the basic workflow of the naive approach.

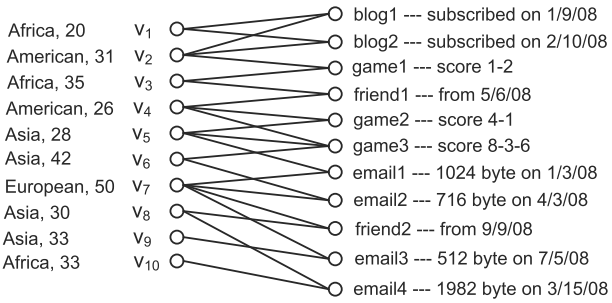


Figure 1 The complete graph

derived from the final published graph, the protocol controls no additional local content of a data agent is released. This solution is known as the famous Secure Multi-Party Computation (SMC) [8, 10, 17, 22]. SMC deals with a problem where a set of parties with private data wish to jointly compute a function of their data. A lot of SMC protocols have been proposed for different computation problems [8, 10, 17, 22], but not for privacy preserving graph publishing in a distributed environment. SMC has two basic requirements: 1) Correctness requirement: the computation is performed in a distributed environment just the same as doing it on an agent who holds the integrated data; 2) Security requirement: each data agent should not know the local information of other agents even with the intermediate results passing through each other. Due to the Correctness requirement, the SMC approach guarantees the published data has the same quality as the Third-Party approach. However, it is not trivial to implement such a protocol. The Correctness requirement requests the computation must be implemented on the integrated information from all data agents. While the Security requirement requires the participants who conduct the computation cannot know the local information of others. These two requirements conflict to each other. An SMC protocol needs to correctly solve this conflict based on the characteristics of the corresponding problem.

In this paper, we follow the SMC approach and design a secure protocol SP to allow the data agents to cooperately generate a graph G' based on a recently proposed protection model [4], called S-Clustering. Through clustering methods, S-Clustering publishes a graph G' that only contains super nodes (clusters) where each super node represents multiple nodes in G. We call a published graph which satisfies S-Clustering as the S-Clustering graph. We select this model due to its three advantages [4]: (1) Unlike previous works

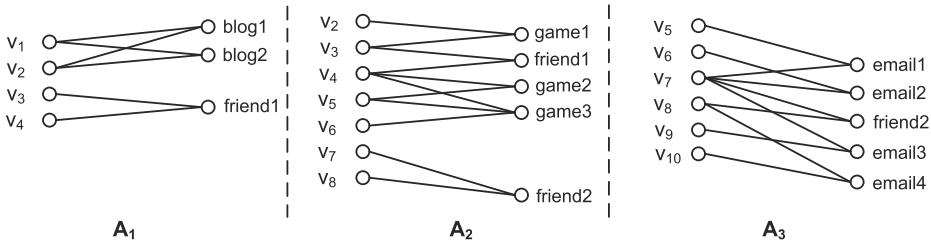
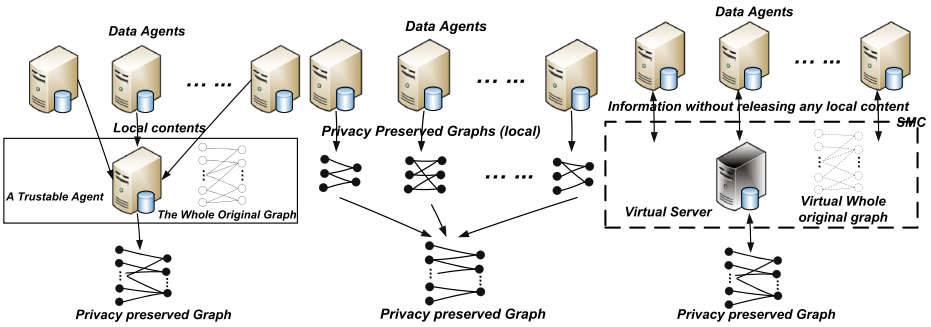


Figure 2 The distributed storage



(a) Third-Party Approach (b) Naive Approach (c) SMC Approach

Figure 3 Architectures for privacy preserving data publishing

[5, 9, 11, 15, 20, 25, 27–29] that only prevent node re-identification in the published graph,[2] S-Clustering provides protections for both nodes and linkages. (2) An attacker's background knowledge is not limited to structural or label. It can be a combination of any structural information and label. (3) Unlike other protection models, S-Clustering supports a flexible representation of rich interaction graphs which is capable of encoding multiple types of interactions between entities, including interactions which can involve large numbers of participants (not just pairs). Nevertheless, in this paper, we propose a protocol to generate a S-Clustering graph in a distributed environment.

We refer the algorithm which generates the S-Clustering graph in a centralized environment as the centralized algorithm. The distributed version of the centralized algorithm, SP, should work the same as running the centralized algorithm on the complete original graph. For any data agent, SP protects its local information when generating the published graph. We propose novel solutions in SP based on random lock, permutation, and Millionaire Protocol [24] and prove that our protocol satisfies all the requirements of SMC.

To demonstrate the effectiveness of SP protocol, we implement a Relaxed Secure Protocol (RSP) (Section 5.1) which is based on the naive approach (Figure 3b). We compare the graphs generated by RSP and SP. The result shows that the graph generated by SP has much higher utility than the one generated by the naive approach. We also test the communication cost of SP on a real cloud computing platform.

Finally, we demonstrated how to extend the basic design idea of SP to another protecting model for simple graphs. The results showed that the protocol design proposed in this paper can be used for different clustering based graph protecting models.

The rest of this paper is organized as follows: we discuss the related works of privacy preserving graph publishing and SMC in Section 2. We define the problem and introduce the S-Clustering [4] in Section 3. In Section 4, the protocol SP is introduced in detail. We demonstrate the effectiveness of SP in Section 5 on a real data set. Section 6 discusses how the proposed design idea can be adapted to other graph protection models on simple graphs and evaluates the effectiveness. Finally, we summarize this work in Section 7.

[2]Node re-identification refers to find a node with certain background knowledge in an anonymized graph

2 Related work

When publishing an anonymized social network, the unique patterns such as node degree, subgraph, or distance to special nodes can be used to re-identify the nodes/links [15]. This kind of attack is called the "structure attack". A lot of works [4, 5, 9, 15, 20, 25, 27–29] have been conducted on how to publish privacy preserving graphs to avoid the "structure attack". The basic methods to generate a privacy preserving graph include clustering and edge editing. Clustering [4, 9, 15, 27] is to cluster "similar" nodes together and publish clusters instead of the original nodes. Edge editing [20, 25, 27–29] is to change the graph's structure by adding/deleting edges.

Hay [15] proposed a clustering model to prevent privacy leakage from vertex refinement, subgraph, and hub-print queries. Zheleva [27] discussed how to use clustering and edge editing to prevent the sensitive link leakage by mining observed links in published networks. Campan [5] discussed how to implement k-anonymous when consider both node labels and structure information by clustering. Cormode [4, 9] introduced (k, l)-clustering model for bipartite graph and S-Clustering for social networks to do the protection respectively. Liu [20] defined and implemented k-degree anonymous model by edge editing, that is for a published network, for any node in it, there exists at least $k-1$ other nodes have the same degree as this node. Zhou [28] considered a stricter model: for every node there exist at least $k-1$ other nodes share isomorphic neighborhoods when taking node labels into account. Zou [29] proposed a k-Automorphism protection model: A graph is k-Automorphism if and only if for every node there exist at least $k-1$ other nodes do not have any structure difference with it. Cheng [7] designed a k-isomorphism model to protect both nodes and links: a graph is k-isomorphism if this graph consists at least k disjoint isomorphic subgraphs. Ying [25] studied how random deleting and swapping edges changes graph properties and proposed an eigenvalues oriented random graph change algorithm. All these works assume there is a trustable data agent who has a full original graph and directly generate the privacy preserving graph G' from G.

Frikken [11] designed a protocol which allows a group of agents to generate an integrated graph. However, the graph generated by this protocol cannot provide the protections against the "structure attacks" proposed in recent works [4, 5, 9, 15, 20, 25, 27–29]. Kerschbaum [19] designed a protocol to generate an anonymized graph from multiple data agents. Each edge in the anonymized graph has a digital signature which can help trace where this edge comes from. While, simple removing the identifiers of nodes in a graph cannot resist an attack which aims to re-identify the nodes/links [15]. Therefore, it is essential to investigate protocols that support the stronger protection models [4, 5, 9, 15, 20, 25, 27–29] in a distributed environment.

Secure multi-party computation (SMC) targets on designing protocols to make a set of participants jointly finish some computations with private inputs. Each participant should not learn the private inputs of others. [18] targeted on set union computation. [1] designed protocol to compute the k^{th} element. [17] proposed a protocol to securely compute the k-anonymous table of the tabular data. They assume the tabular data is horizontally partitioned and stored on multiple agents. [21] designed protocols to compute the k-means of vertically partitioned tabular data. Most SMC works suppose the participants are semi-honest. The semi-honest means all the participants follow the protocol as requested but are eligible to make induction based on the intermediate results and final published result. Jiang [16] discussed how to extend a secure two party computation framework for semi-honest users to malicious users. However, they claimed in their paper [16] "most practical algorithms

developed have only been proven secure under the semi-honest model. While not a proof, this certainly gives evidence that achieving security against a malicious adversary adds significant complexity and expense". In this paper, we suppose all data agents are semi-honest and leave the malicious case as our interesting future work.

3 Problem description

3.1 Graph and storage model

An online social network with rich interactions can be represented as a bipartite graph $G(V, I, E)$ [4, 14, 23], where each node in V represents an entity in the social network and each node in I stands for an interaction between a subset of entities in V. An edge $e(u, i) \in E$ ($u \in V, i \in I$) means entity u participates in the interaction i. Each entity has an identity and a group of attributes. Without loss of generality, each entity's identifier can be represented as a unique id within the range $[1, |V|]$. In the rest part of this paper, we refer entity u as the same as the entity with id u. Each interaction also has an identity and a set of properties as shown in Figure 1. An interaction can involve more than two entities, such as the interaction "game3". Two entities can also be involved in different interactions at the same time. For example, in Figure 1, v_1 and v_2 participate in "blog1" and "blog2" simultaneously.

In a distributed environment, the graph G is distributively stored on l different data agents. That is, each agent A_i holds a portion of the graph $G_i(V_i, I_i, E_i)$ such that $V = \cup_{i=1}^{l} V_i$, $I = \cup_{i=1}^{l} I_i$ and $E = \cup_{i=1}^{l} E_i$. The interactions that an entity participates may be stored on different agents. Note each V_i must contain all the entities that participate in every interaction in I_i. In other words, each interaction stored on an agent must be complete. The G_i held by different data agents may overlap, such that the intersection between two graphs stored on different agents might not be an empty set.[3] That is, $\cap_{i=1}^{l} V_i \neq \phi$ and $\cap_{i=1}^{l} I_i \neq \phi$.

Figure 2 shows an example of a distributed storage of Figure 1, where both entities and interactions have overlaps in different agents. For example, both A_1 and A_2 store node v_2 and interaction $friend1$. In the rest of this paper, we use **node** to specifically represent an entity in V and use **interaction** to represent an item in I. When we say two nodes u, v have a **connection**, it means u and v are involved in a same interaction i in I (In the graph, we have two edges $e(u, i)$ and $e(v, i)$).

3.2 Problem definition

In this paper, we propose a protocol which securely generates a S-Clustering graph [4] for interaction based graphs in a distributed environment. S-Clustering [4] assumes that an attacker can know the id, label and any connection information of a node. An attacker uses the information he knows about some users to analyze the published graph in order to learn more about these users. For example, an attacker knows that two users have an interaction, which is blog1 subscribed on 1/9/08. When he read a published graph such as Figure 1, he can learn the two users must be v_1 and v_2. Then these two users' age, nationality as well as the information that they also have another interaction blog2 subscribed on 2/10/08

[3] When V_is do not have overlap ($\cap_{i=1}^{l} V_i = \phi$), G is a disconnected graph since each data agent only holds a subgraph that is isolated from other parts.

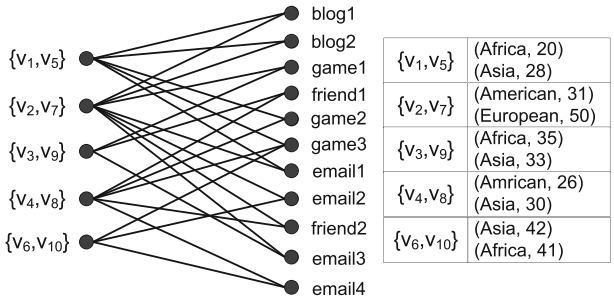


Figure 4 A clustered graph

are disclosed to the attacker. Given a constant k, S-Clustering publishes a clustered graph (S-Clustering graph) which guarantees the following three *Privacy objectives*:

Objective 1: For any node u, an attacker has at most $\frac{1}{k}$ probability to recognize it.

Objective 2: For any interaction i, an attacker has at most $\frac{1}{k}$ probability to know a node u involves in it.

Objective 3: For any two nodes u and v, an attacker has at most $\frac{1}{k}$ probability to know they have a connection (u and v participate in a same interaction).

The problem we solve in this paper is:

1. ***Input***: A graph G which is distributed stored on l data agents and a constant privacy parameter k.
2. ***Output***: A S-Clustering graph G' of G under k.
3. ***Constraints***:
   (a) Correctness: G' is computed just the same as generating it on an agent who holds G;
   (b) Security: The three *Privacy objectives* of S-Clustering are guaranteed even when each participant gets the intermediate results during the computation.[4]

3.3 S-clustering model

Given a graph $G(V, I, E)$, to satisfy the three privacy objectives, a S-Clustering graph $G'(CV, I, CE)$ is published, where CV is a super node set in which each super node represents a group of entities in V. These super nodes are also called clusters. We use c to denote a cluster in CV and $|c|$ to denote the cluster size, which is the number of entities in c. For each interaction i in I, if c contains an entity which participates in i, there is an edge $e(c, i)$ in G'. Figure 4 is a clustered graph of Figure 1.

To guarantee the three privacy objectives, the S-Clustering graph G' must satisfy:

– Each cluster represents at least k entities. This guarantees the Objective (1).
– The *Clustering Safety Condition (CSC)* [4]:

[4]The local information of each agent which violates the *Privacy objectives* of S-Clustering is protected.

Definition 1 A division (clustering) of nodes V into clusters satisfies the **Clustering Safety Condition (CSC)** if for any node $v \in V$, v participates in interactions with at most one node in any cluster $S \subset V$, that is:

- $\forall e(v,i), e(w,i), e(v,j), e(z,j) \in E : w \in S \wedge z \in S \Rightarrow z = w$;
- $\forall e(v,i), e(w,i) \in E : v \in S \wedge w \in S \Rightarrow v = w$;

This condition guarantees Objectives (2) and (3).

Algorithm 1 Clustering with CSC

```
1  Sort(V);
2  CV=null;
3  for v ∈ V do
4      flag = true;
5      for cluster c ∈ CV do
6          if SIZE(c) <k and CSC(c,v) then
7              Insert(c,v);
8              flag = false;
9              break;
10     if flag then
11         Create a New Cluster c and add c to CV;
12         Insert(c, v);
```

The CSC requires that any two nodes in a cluster cannot be connected or connect to the same node. Figure 4 is a clustered graph which satisfies CSC. For any two nodes that appear in the same cluster of Figure 4, they do not have a connection or connected to the same third node. For any two nodes u and v, if they satisfy (or do not satisfy) the CSC, we denote this as $CSC(u, v)$ (or $\neg CSC(u, v)$). Similarly, for a cluster c and a node u, if u satisfies (or does not satisfy) CSC with all the nodes in c, we denote this as $CSC(c, u)$ (or $\neg CSC(c, u)$).

Algorithm 1 [4][5] is used to generate a published graph. Firstly, the nodes in V are sorted by a pre-given sorting rule, for example, by node degrees. Then, each node v is sequentially added into an existing cluster c if $CSC(c, v)$. This is tested in line 6. If there is no such a cluster, a new cluster which only contains v is created.

It is suggested to sort nodes on one or more node attributes and degree before clustering in order to achieve good utility [4]. The selected attributes for sorting is based on queries that will be operated on the published graph. So the sorting rule is set as sorting nodes by some attributes and the degree. However, since all the node attributes are published as shown in Figure 4, it is not safe when the sorting rule contains node attributes even without publishing the sorting rule. Assume we publish a clustered graph where each node has two labels, location and age. A portion of the graph is shown in Figure 5. Without any prior background knowledge, we may observe that location is a parameter to do the sorting when the graph is large. This is because nodes with same location tend to be grouped together. There are one interaction between clusters (1,8,9) and (2,5,7), thus, we conclude there are

[5]When Algorithm 1 finishes, it is possible that there exists several groups with size less than k. The nodes in these groups can be assigned to other groups under CSC similar to lines 5-9. In this paper, same as [4], we describe the algorithm by ignoring this part.

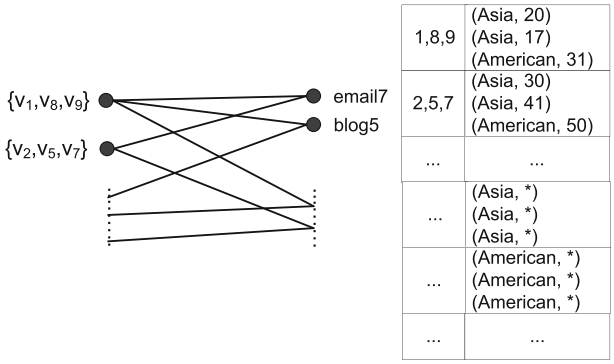


Figure 5 A clustered graph

edges $e((American, 31), email7)$ and $e((American, 50), email7)$.[6] Therefore, in order to guarantee the privacy, the sorting rule should not contain any node attribute. Another problem is that all the privacy preserving graph publication works [4, 5, 9, 11, 15, 20, 25, 27–29] assume the future mining tasks or queries on the published data cannot be known in advance. Otherwise, the mining results or query results instead of a graph can be directly published. Therefore, in this paper, we sort the nodes by degree. It should be emphasized that this is not the constraint to our protocol. Our protocol can support sorting by attributes. Since globally each attribute value can be represented by a number, the sorting on any attribute is the same as sorting on degree.

4 Secure protocol (SP)

4.1 Overview

We proposed SP to generate a S-Clustering graph in a distributed environment as shown in Algorithm 2. We call a computation that satisfies the Security requirement as a Secure Computation. Thus, SP contains four stages such that each stage conducts one Secure Computation in lines 1, 7, 15, and 16, respectively. Stage 1 sorts nodes without releasing any degree information. Stage 2 clusters nodes by correctly checking $CSC(c, v)$ without releasing any connection information. Stage 3 and Stage 4 generate the interactions on clusters and attributes for clusters respectively without disclosing any interaction-node mapping and node-attribute mapping.

When the computations in lines 1, 7, 15, and 16 of Algorithm 2 can be securely operated, the only intermediate information is the result of CSC checking (line 6 & line 7) and the computation order of nodes (line 1 & line 3). Unfortunately, a participant can potentially break the Security requirement if he knows these information.

Suppose during the computation, a participant knows nodes u and v does not satisfy CSC ($\neg CSC(u, v)$) (i.e, the result of CSC checking). Moreover, in the final published graph, u and v are in two clusters which connect each other through interactions and do not connect

[6]The soring rule is either "R_1(American, Asia)" or "R_2(Asia, American)". For R_1, the connection must be between two Americans. Otherwise, these two Americans should be put into the same cluster. For R_2, it cannot be the sorting rule. Otherwise, at least one group should contain three nodes only with attribute "Asia".

Table 1 A violation testing sequence

Step	CSC Result	Current Clusters
1	$\neg CSC(1,2)$	({1}, {2})
2	$\neg CSC(1,3)$	({1}, {2})
3	$\neg CSC(2,3)$	({1}, {2}, {3})
4	$\neg CSC(1,4)$	({1}, {2}, {3})
5	$\neg CSC(2,4)$	({1}, {2}, {3})
6	$CSC(3,4)$	({1}, {2}, {3, 4})
7	$CSC(1,5)$	({1, 5}, {2}, {3, 4})
8	$CSC(2,6)$	({1, 5}, {2, 6}, {3, 4})

to the same third cluster. Then this participant can conclude that u and v have a connection without any background knowledge. In the above example, if u is replaced by a cluster c, this participant can conclude for any node u' in c, the probability that u' and v have a connection $p' \geq \frac{1}{|c|}$. Since $|c| < k$ when checking CSC in Algorithm 2, $p' \geq \frac{1}{k}$. This violates one of the privacy protection objectives.[7] As a result, the computation information that contains any $\neg CSC$ between nodes should not be revealed to any participant.

Algorithm 2 Clustering with CSC in distributed environment

```
1  Securely Sort(V) by degree;                                          /* Stage1 */
2  CV=null;                                                        /* Begin Stage2 */
3  for v ∈ V do
4      flag = true;
5      for cluster c ∈ CV do
6          if SIZE(c)<k then
7              Distributed computing r = CSC(c, v);
8              if r then
9                  Insert(c,v) without generating/updating edges between clusters;
10                 flag = false;
11                 break;
12     if flag then
13         Create a New Cluster c and add c to CV;
14         Insert(c, v);                                          /* End Stage2 */
15 Compute interactions from distributed agents;                         /* Stage3 */
16 Publish a graph with node attributes only links to clusters;          /* Stage4 */
```

To hide the computation information that contains $\neg CSC$, the computation order of nodes should also be hidden. For example, we would like to achieve our privacy objectives with $k = 2$, and the sorting order of the nodes is 1, 2, 3, 4, 5, 6. If the final clustering result is {{1, 5}, {2, 6}, {3, 4}}, according to Algorithm 2, the computation information as in Table 1 can be deduced. This information contains the results of CSC checking, which may also cause the violation of the Security as analyzed in the previous paragraph.

To summarize, we must avoid the releasing of the following information:

1. Degree information (from Stage 1);

[7] The combination of intermediate results and the final result should not break any privacy protection objective of the S-Clustering model.

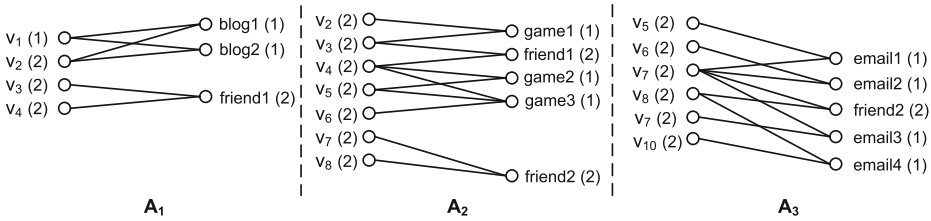


Figure 6 The distributed storage with weights

2. Specific node-attribute mapping (from Stage 4);
3. Specific connection information, including node-interaction mapping and connection between nodes (from Stage 2 & 3);
4. Any $\neg CSC$ result and the computation order of nodes (from Stage 1 & 2).

Next, we introduce our design of SP for each stage in detail and prove the Correctness and Security of our protocol. Before presenting SP, we assume each node/interaction has a weight that represents how many duplicated copies of this node/interaction have been stored in the system. We call these weights *duplicated weights*. Duplicated weights of nodes/interactions can be computed by passing each node/interaction's id through all data agents to query how many duplicates are stored. Since only node/interaction ids are passed, this process does not violate the Security requirement. Details of the method to generate duplicated weights is shown in Appendix A. Figure 6 shows the distributively stored graphs with duplicated weights. For example, v_2 has weight 2 since two agents A_1 and A_2 store it. $friend2$ has weight 2 since both A_2 and A_3 store it. During the process of our protocol, when counting an attribute or an interaction with duplicated weight w, we multiply the attribute value with $\frac{1}{w}$ to eliminate the overlapping.

4.2 Stage 1: Secure sorting sub-protocol (SSSP)

4.2.1 Protocol design

In this stage, we sort the nodes without revealing any degree information and the computation order of nodes (the sorting order of nodes on real ids) information. The basic idea is to do the sorting on permuted ids with their corresponding "locked" degree values. We use the random number to lock the real degree of each node. That is, we make one agent hold a key (a random number) and another agent hold the locked degree value (real degree plus this random number). Then, we sort all nodes on permuted ids through the cooperation of these two agents. During the sorting, the agent who has the locked degree values cannot learn any key value and the agent who holds the keys cannot learn any locked value. The Secure Sorting Sub-Protocol (SSSP) works as follows (Figure 7):

1. A_1 generates a random real number vector D with size $|V|$. A_1 constructs a variable vector $D' = D$. Then, for each node u stored on A_1, A_1 adds $\sum_{e(u,i)} \frac{1}{w_i}$ to $D'[u]$ (w_i is the duplicated weight of the interaction i). Finally, A_1 sends D' to A_2;
2. When A_2 receives D', for each node u stored on A_2, we add $\sum_{e(u,i)} \frac{1}{w_i}$ to $D'[u]$. Then, A_2 sends D' to A_3;
3. Each A_i ($i > 1$) does the same operation as A_2 and sends D' to A_{i+1} if A_{i+1} exists;
4. A_l generates a random permutation function π, passes π to A_1 and sends $\pi(D')$ to S_1

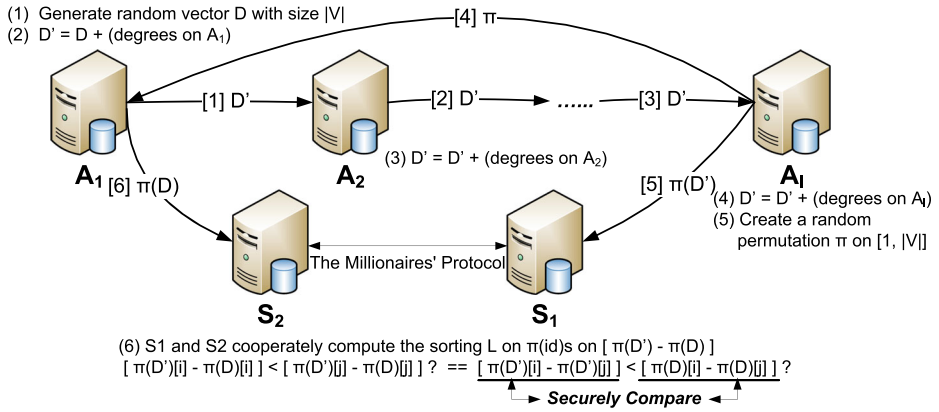


Figure 7 Stage 1: The Secure Sorting Sub-Protocol (SSSP)

5. A_1 sends $\pi(D)$ to S_2.
6. S_1 and S_2 cooperately sort all the nodes. During the sorting, each time when S_1 needs to compare two values $\pi(D')[i] - \pi(D)[i]$ and $\pi(D')[j] - \pi(D)[j]$, S_1 and S_2 uses the Millionaires' Protocol [24][8] to securely get the result by comparing the value $(\pi(D')[i] - \pi(D')[j])$ on S_1 and $(\pi(D)[i] - \pi(D)[j])$ on S_2.

Theorem 1 *SSSP sorts all nodes on $\pi(id)$s without violating the Security requirement.*

The detailed proof of Theorem 1 can be found in Appendix B.

4.3 Stage 2: Secure clustering sub-protocol (SCSP)

4.3.1 Protocol design

In Stage 2, we need to cluster nodes based on the sorted order L without revealing any connection information. Since S_1 holds L after Stage 1, in $SCSP$, we continue to make S_1 generate clusters on $\pi(id)$s. Before S_1 can do clustering on $\pi(id)$s, the connection information between nodes should be mapped on $\pi(id)$s firstly. So, $SCSP$ contains the following three steps:

1. Generate a noise matrix MR on A_1 and a noised adjacent matrix MR' of G on A_I. For any two nodes u and v, $MR'[u][v] - MR[u][v] > 0$ is equivalent to u, v has a connection;

[8]The Millionaires' Problem is: there are two numbers a and b hold by two people, they want to know the inequality $a > b$ or $a < b$ without revealing the actual values of a and b. The Millionaires' Protocol [24, 26] can securely compare a and b based on the techniques such as Homomorphic Encryption. When we want to compare two degree values d_1 and d_2 in case that S_x knows r_1, r_2 and S_y knows $(r_1 + d_1), (r_2 + d_2)$, we can use the Millionaires' Protocol to compare the value $((r_1 + d_1) - (r_2 + d_2))$ on S_x and the value $(r_1 - r_2)$ on S_y. Since $((r_1 + d_1) - (r_2 + d_2)) > (r_1 - r_2)$ equals to $d_1 > d_2$ and vice versa, the comparison of r_1 and r_2 is correctly performed.

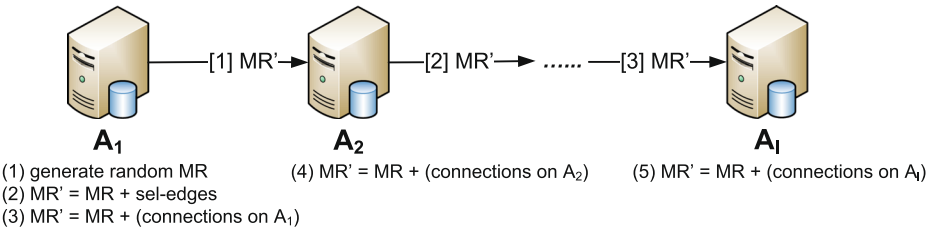


Figure 8 Stage 2: The Secure Clustering Sub-Protocol (SCSP) 1

2. Create a matrix NMR on A_1 and NMR' on A_l based on MR and MR', respectively. For any two nodes u and v, $NMR'[u][v] - NMR[u][v]$ is a [0, 1] matrix where $NMR'[u][v] - NMR[u][v] = 1$ indicates that u, v have a connection;
3. Do clustering on $\pi(id)$s with $\pi(NMR')$, $\pi(NMR)$ through the cooperations among S_1, S_2 and S_3.

The first two steps map the connection information on $\pi(id)$s (The [0, 1] character of $NMR' - NMR$ will be used for securely clustering). The last step does the clustering on the connection information between $\pi(id)$s. During the computation, to satisfy the Security requirement, we make the agent who holds MR, NMR or $\pi(NMR)$ cannot get the noised content MR', NMR' or $\pi(NMR')$ and vice versa.

Figure 8 shows the first step's working process to compute MR and MR' securely:

1. A_1 generates a random real number matrix MR with size $|V| \times |V|$. A_1 constructs a variable matrix $MR' = MR$. Then, a self-edge is added for each node (The self-edge will be used to determine the CSC). For each u, A_1 sets $MR'[u][u] = MR'[u][u] + a_1$ (a_1 is a random positive number). For any interaction i and any two nodes u and v that participate in i, A_1 sets $MR'[u][v] = MR'[u][v] + a_2$ (a_2 is a positive random number), and then A_1 sends MR' to A_2;
2. After A_2 getting MR', for any interaction i and any two nodes u and v that participate in i, A_2 sets $MR'[u][v] = MR'[u][v] + a_3$ (a_3 is a positive random number), and then A_2 sends MR' to A_3;
3. Each A_i ($i > 1$) does the same operation as A_2 and sends MR' to A_{i+1} if A_{i+1} exists;

After the first step, A_l gets a noised matrix MR'. For any nodes u and v, $MR'[u][v] - MR[u][v] > 0$ is equivalent to u and v have a connection.

The second step securely computes the matrices NMR and NMR' as shown in Figure 9:

1. A_l generates a random permutation function π_2 on $|V| \times |V|$ numbers and sends π_2 to A_1. A_l sorts all the numbers in MR' row by row to obtain a vector L_2' whose length is $|V| \times |V|$;
2. A_1 sorts MR row by row to obtain a vector L_2;
3. A_1 sends $\pi_2(L_2)$ to S_3 and A_l sends $\pi_2(L_2')$ to S_3;
4. S_3 computes two random number vectors LN_2 and LN_2' which satisfy

$$LN_2'[i] - LN_2[i] = \begin{cases} 0 & \pi_2(L_2')[i] - \pi_2(L_2)[i] = 0 \\ 1 & \pi_2(L_2')[i] - \pi_2(L_2)[i] > 0 \end{cases}$$

Then, S_3 passes LN_2 to A_1 and LN_2' to A_l;

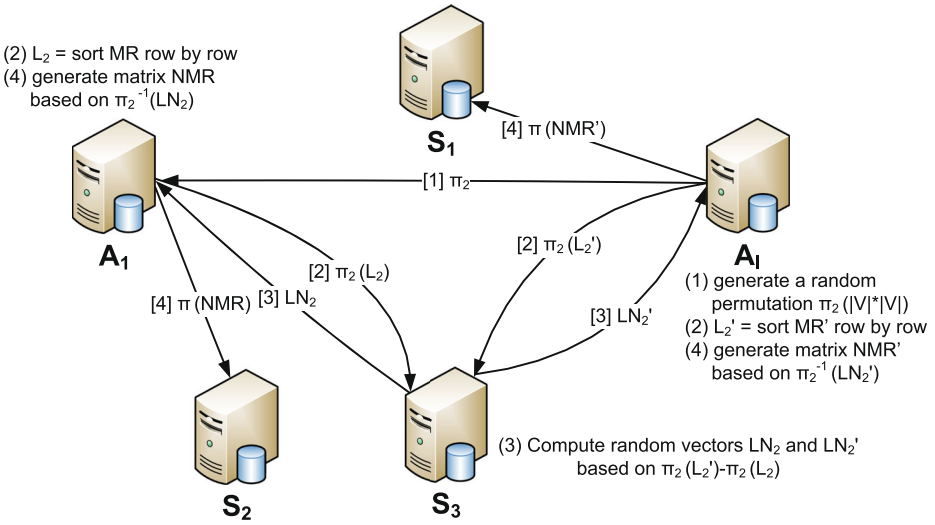


Figure 9 Stage 2: The Secure Clustering Sub-Protocol (SCSP) 2

5. A_1 computes $\pi_2^{-1}(LN_2)$ and converts the results to a $|V| \times |V|$ matrix NMR. A_1 passes $\pi(NMR)$ to S_2; Similarly, A_l generates the corresponding matrix NMR' based on $\pi_2^{-1}(LN_2')$ and passes $\pi(NMR')$ to S_1;

The third step is to do the clustering on $\pi(id)$s. Before introducing how S_1 conducts the clustering, we prove a property we use in the next computation. For a node u, we call E_u ($|E_u| = |V|$) the *connection vector* of u. E_u is a [0, 1] vector and $E_u[u] = 1$. If nodes u and v have a connection, $E_u[i] = 1$, otherwise $E_u[i] = 0$. For example, node v_1's connection vector in Figure 1 $E_1 = [1, 1, 0, 0, 0, 0, 0, 0, 0, 0]$. For a cluster of nodes C, we call $E_C = (\sum_{\forall u \in C} E_u)$ as C's *connection vector*. The connection vector has the following property:

Theorem 2 *For any cluster of nodes C, if there exists a t where* $E_C[t] > 1$, *then C does not satisfy CSC; otherwise, C satisfies the CSC.*

The detailed proof of Theorem 2 is shown in Appendix C.1.

For a cluster of nodes C on $\pi(id)$s, it is obvious $E_C = \sum_{\forall u \in C}(\pi(NMR')[u] - \pi(NMR)[u])$. Based on the property of connection vector, when S_1 needs to check whether v can join a cluster c based on CSC, the protocol works as (Figure 10):

1. S_1 computes a connection vector $ER_C = \sum_{\forall u \in C} \pi(NMR')[u]$. Where $C = c \cup \{v\}$. The size of this vector is $|V|$.;
2. S_1 generates a random permutation pattern π' and sends (C, π') to S_2;
3. S_2 computes a vector $R_C = \pi'(\sum_{\forall u \in C} \pi(NMR)[u])$. S_2 continues to generate a random vector R' with size $|V|$ and sends R' to S_1. S_2 sends $R'_C = R_C + R'$ to S_3;
4. S_1 computes $ER'_C = \pi'(ER_C) + R'$ and passes ER'_C to S_3;
5. S_3 computes $E_C = ER'_C - R'_C$. If E_C contains a number bigger than 1, it returns "cannot cluster" to S_1. Otherwise, it returns "can cluster" to S_1.

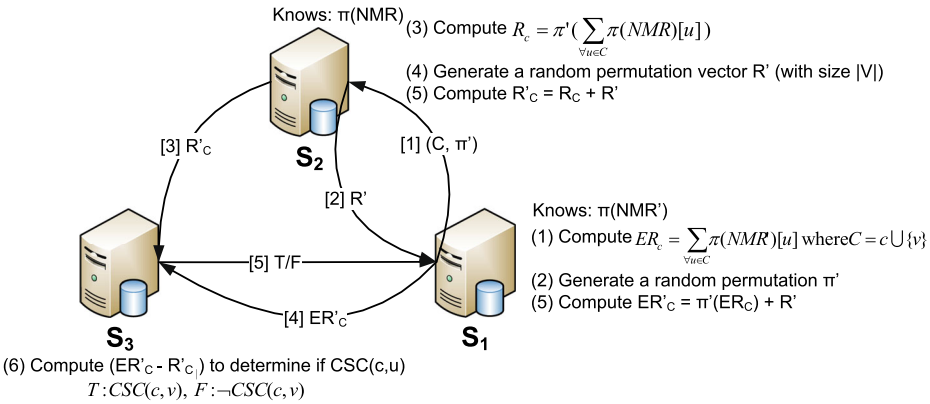


Figure 10 Stage 2: The Secure Clustering Sub-Protocol (SCSP) 3

S_1 does the clustering on L and uses the above method to test CSC. S_1 finally gets the cluster set CV' on $\pi(id)$s and passes CV' to A_1. A_1 computes the cluster set on real ids $CV = \pi^{-1}(CV')$.

Theorem 3 *SCSP clusters all nodes without violating the Security requirement.*

The detailed proof of Theorem 3 can be found in Appendix C.2.

4.4 Stage 3: Secure edge generation sub-protocol (SESP)

4.4.1 Protocol design

In Stage 3, we generate the interactions among clusters. After A_1 gets the cluster set CV on real ids, A_1 reports CV to all A_is one by one. Each A_i generates the interactions between clusters and sends the results to A_{i+1}. Finally A_l gets a clustered graph with correct interactions. The working process of $SESP$ is shown in Figure 11. The above process only passes the connection information through interactions without the attribute information. Each A_i directly sends the attribute of each interaction to A_l since all the interactions will be clearly published.

Since the interactions between super nodes reported by each A_i is a sub-set of the final result, the middle results are already included in the published graph. Therefore the Security requirement is satisfied.

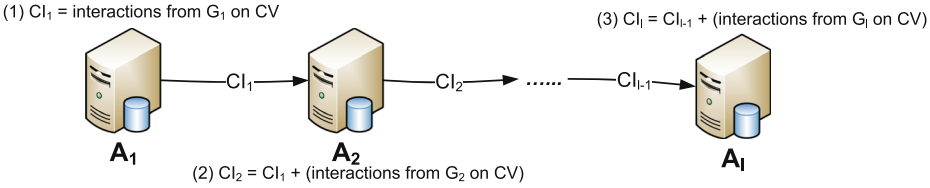


Figure 11 Stage 3: Secure Edge Generation Sub-Protocol (SESP)

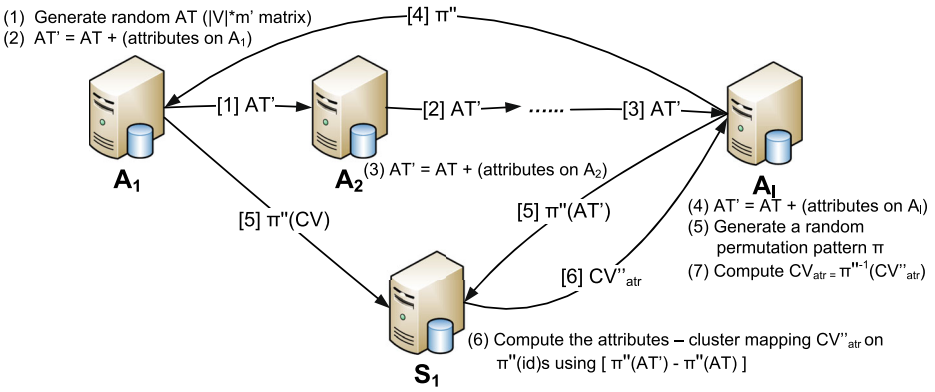


Figure 12 Stage 4: Secure label generation sub-protocol (SLSP)

4.5 Stage 4: Secure label generation sub-protocol (SLSP)

4.5.1 Protocol design

In Stage 4, we generate the node attributes for each cluster without disclosing any specific node-attribute mapping. Suppose there are m' attributes for each node. The Secure Label Generation Protocol (SLGP) works as follows (Figure 12):

1. A_1 generates a random real number matrix AT with size $|V| \times m'$. A_1 constructs a variable matrix $AT' = AT$. Then for each node u stored on A_1, for any attribute x of node u, if u's duplicated weight is w_u and the attribute value is a, $AT'[u][x] = AT'[u][x] + \frac{1}{w_u}a$;
2. When A_2 receives AT', for each node u stored on A_2, for any attribute x of node u, if u's duplicated weight is w_u and the attribute value is a, $AT'[u][x] = AT'[u][x] + \frac{1}{w_u}a$. A_2 sends AT' to A_3;
3. Each A_i $(i > 1)$ does the same operation as A_2 and sends AT' to A_{i+1} if A_{i+1} exists;
4. A_l generates a random permutation function π'' (a new permutation function), A_l passes π'' to A_1 and sends $\pi''(AT')$ to S_1;
5. A_1 sends $\pi''(AT)$ and $\pi''(CV)$ to S_1
6. S_1 computes $\pi''(AT') - \pi''(AT)$, then generates the node attributes for each cluster in $\pi''(CV)$. Suppose the result is CV'_{atr};
7. S_1 sends CV''_{atr} to A_l, A_l computes $\pi''^{-1}(CV''_{atr})$ to get the node attributes for each cluster.

Theorem 4 *SLSP assigns node attributes to each cluster without violating the Security requirement.*

We prove this theorem by showing for any participant, no node-attribute mapping is disclosed. The analysis is similar with $SSSP$, we ignore this part.

Based on the above step by step illustration of Algorithm, we conclude that the whole SP satisfies the Correctness and Security requirements.

Corollary 1 *SP exactly implements Algorithm 2 and satisfies the Security requirement.*

Algorithm 3 The algorithm running on data agent A_i

```
Receive a super graph G'_{i-1}(CV_{i-1}, CI_{i-1}, CE_{i-1});
V'_i = V_i − {u|∀c ∈ CV_{i-1}(u ∈ c)} ;
Sort(V'_i) based on G_i;
CV_i = {};
for v ∈ V'_i do
    flag = true;
    for cluster c in CV_i do
        if (CSC(c,v) on G_i) and SIZE(c)<k then
            Insert(c,v);
            flag = false;
            break;
    if flag then
        Insert(CreateNewCluster(), v) to CV_i;
Compute interaction set (CI_i, CE_i) based on (I_i, E_i) without violating CSC;
Compute cluster-attributes mapping of the clusters in CV_i;
Pass G'_i = (CV_{i-1} ∪ CV_i, CI_{i-1} ∪ CI_i, CE_{i-1} ∪ CE_i) to A_{i+1};
```

It is obviously that SP implements Algorithm 2 step by step. We can observe that SP has the following property: the intermediate results of one stage do not influence other stages since computation contents are different. Since each stage satisfies the Security requirement, the whole SP also satisfies the Security requirement.

5 Experiment

To demonstrate the effectiveness of SP protocol, we implement a Relaxed Secure Protocol (RSP) which is based on the naive approach (Figure 3b). We compare the graph generated by RSP and SP. We also test the communication cost of SP on a real cloud computing platform.

5.1 Relaxed secure protocol (RSP)

We implement RSP by clustering nodes on local data agents and deleting some cross agent interactions. By doing this, the Security can be guaranteed. A *cross agent interaction* is an interaction whose participants contain at least two nodes which have duplicates in the system. For example, the interaction $game1$ in Figure 2 is a cross agent interaction since both v_2 and v_3 are stored more than 1 once. We only delete the cross agent interactions which break or have the potential risk to break the CSC.

RSP runs Algorithm 3 on each A_i. For each A_i, a super graph $G'_{i-1}(CV_{i-1}, CI_{i-1}, CE_{i-1})$ is passed from A_{i-1} to it. A_i first computes a node set V'_i which contains all the nodes in V_i that have not been involved in CV_{i-1}. Then A_i sorts and clusters the nodes in V'_i based on the local information G_i. The clusters composed by the nodes in V_i is stored in set CV_i. For an interaction i, if it satisfies the following conditions:

1. It does not connect two clusters in CV_{i-1} which already have a connection in G'_{i-1};
2. It does not create connection within a cluster in CV_{i-1};
3. It does not violate the CSC combining with already selected interactions.

A_i selects interaction i, otherwise A_i deletes it. For case 1, when two clusters in CV_{i-1} already have a connection, interaction i may connect two nodes in the same cluster to the same node. It has the potential to violate the CSC. Therefore, we delete interaction i. For

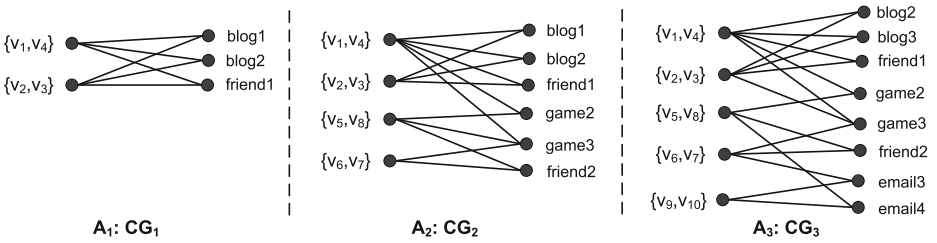


Figure 13 A running example of RSP

case 2, since agents A_1, ..., A_{i-1} only cluster nodes using their local information, it is possible A_i finds two nodes in a cluster in CV_{i-1} have a connection with G_i. In this case, A_i directly deletes the interaction which causes this connection. For case 3, since the clusters in CV_{i-1} are only based on local information, A_i may find two clusters in CV_{i-1} does not satisfy CSC. In this case, A_i only remains the interactions that do not violate CSC.

Finally A_i generates a new clustered graph G'_i and passes G'_i to data agent A_{i+1}. When A_l finishes the computation, A_l gets the published graph. A_1 starts with an empty graph G'_0.

Figure 13 shows an example of RSP on Figure 2. A_1 generates a clustered graph G'_1 with two clusters $\{\{v_1, v_4\}, \{v_2, v_3\}\}$. A_2 creates two new clusters $\{\{v_5, v_8\}, \{v_6, v_7\}\}$. For the interaction "game1", it connects nodes v_2 and v_3. Since v_2 and v_3 are put in the same cluster in G'_1, "game1" is deleted. After A_2 generates clustered G'_2, it passes G'_2 to A_3. A_3 generates the cluster $\{v_9, v_{10}\}$. Since interaction "email2" connects two nodes in the same cluster in G'_2, it is deleted. "email1" is deleted since the two super nodes $\{v_5, v_8\}$ and $\{v_6, v_7\}$ already have connections in G'_2. Finally, A_3 constructs the final published graph. Here due to the sparsity of the social networks, we make the same assumption as [4] that each data agent can find clusters that satisfy CSC.

Theorem 5 *RSP generates a clustered graph which satisfies the three privacy requirements of S-Clustering without violating the Security requirement.*

The proof of this theorem is straightforward since each participant only does the computation on its local content.

5.2 Criteria

The analysis in our protocol description part proves the Correctness and Security of SP. Since SP exactly follows the centralized algorithm which by default gets the best utility, we can use SP as the baseline and compare with RSP. We focus on comparing two aspects to show the benefit of designing an SMC protocol: the information loss and the utilities.

The naive approach RSP does the clustering with local knowledge and can only delete the interactions which may violate the CSC. We should estimate the information loss of RSP comparing with SP (SP does not delete any interaction). Assume del is the number of interactions that are deleted by RSP and the original complete graph contains $|I|$ interactions, we use the ratio of deleted interactions ($\frac{100del}{|I|}\%$) to represent the information loss.

Utility is used to estimate the quality of the published graph. For the clustering-based protection models [4, 5, 15, 27], the utility testing is operated by drawing sample graphs from the published one, measuring the utility of each sample and aggregating utilities across

samples. Most previous works [5, 9, 11, 15, 20, 25, 27–29] used the difference between certain statistic graph characteristics of the original graph and the published graph as the utility. Some works [4, 28] also used the average query errors of some randomly selected aggregate queries as the utility. In our experiment, we follow the above selections and test the following measures:

1. Degree Distribution
   Degree Distribution[20, 29] is a basic graph character which is considered by nearly all social network analysis works. Suppose the sorted degree sequence of the original graph G is D_G and the sorted degree sequence of a sampled graph G_c is D_{G_c}. The different between the degree distributions of G and G_c can be represented as the Euclidean Distance between D_G and D_{G_c}:

$$ED_{DD}(D_G, D_{G_c}) = \sqrt{\frac{1}{|V|}\sum_{i=1}^{|V|}(D_G[i] - D_{G_c}[i])^2}$$

   We compute the average ED_{DD} of 30 sampled graphs for each protocol. Suppose the SP's result is $ED_{DD,SP}$ and RSP's result is $ED_{DD,RSP}$, we use $\frac{ED_{DD,RSP}-ED_{DD,SP}}{ED_{DD,SP}} \times 100\ \%$ to compare these two protocols.
2. A group of randomly selected queries
   We test three aggregate queries as the same as [4]. We compute the average query error between the sampling graphs and the original graph. Suppose SP's result is $error_{SP}$ and RSP's result is $error_{RSP}$, we use $\frac{error_{RSP}-error_{SP}}{error_{SP}} \times 100\ \%$ to compare them. The three aggregate queries include:

   (a) Pair Queries: how many nodes with certain attribute interact with nodes with another attribute. For instance, how many users from American are friends with users from Asia?
   (b) Trio Queries: how many two hop neighbors. For example, how many Americans are friends with Asians who are also friends with Africans?
   (c) Triangle Queries: how many triangles. For instance, how many Americans are friends with Asians and Africans who are also friends?

   As the same as [4], we select a random set of queries of each type to do the testing. For each type, we select 20 queries and compute the average query error between the sampling graphs and original graph.

5.3 Data set and distributed storage

Our experiment is operated on a real life social network from ArXiv (arXiv.org). ArXiv(arXiv.org) is an e-print service system in Physics, Mathematics, Computer Science, Quantitative Biology, Quantitative Finance and Statistics. We extract the co-author graphs in Computer Science. Each node denotes an author, and each interaction means a unique coauthor paper. The ArXiv provides 37 categories in Computer Science to search the papers. Since most people work in multiple sub-fields and each paper also belongs to multiple sub-fields, the authors of papers in different categories are overlapped and finally form a large graph. We set each author's research field as his/her node label. The research field is set as the category of papers he/she published most. There are totally 37 different values of node attribute. We divide the 37 categories to 6 sets and store the crawled graph of each category

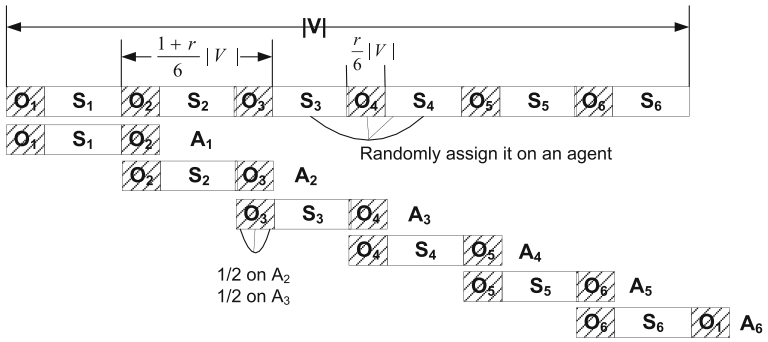


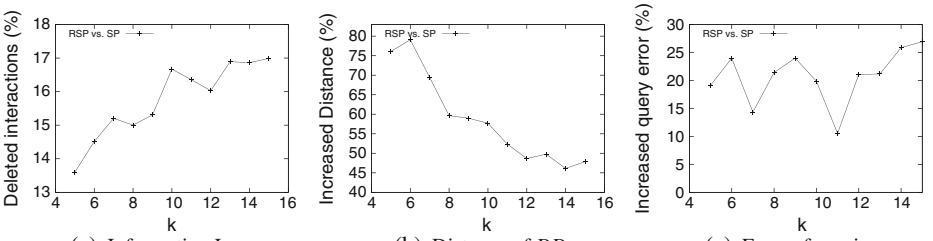
Figure 14 Generate the distributed storage

set on one data agent. This can be seen as different data agents maintain the relationship of people in different fields. The number of nodes stored in the 6 data agents are 6371, 6370, 7705, 5923, 4676 and 7876 respectively. There are $r = 34$ % nodes stored multiple times on different agents. The integrated graph of these 6 subgraphs contains 28868 nodes and 23290 interactions. On average, each interaction is involved by 2.4 nodes.

We consider the following two distributed storage cases:

1. Real Distributed Case
   The 6 subgraphs we crawled in different research fields can be seen as 6 data agents maintain the relationship of people in different research fields. The number of nodes stored in the 6 data agents are 6371, 6370, 7705, 5923, 4676 and 7876 respectively. There are $r = 34$ % nodes stored multiple times on different agents. We call r the node overlapping ratio.
2. Simulated Distributed Case
   We manually divide the integrated graph G to 6 parts that have similar node numbers with different node overlapping ratios (rs). When we want to generate the distributed storage with node overlapping ratio around r, we use the following method to divide the graph:
   (a) We randomly sort all the nodes in G;
   (b) We put the nodes to 6 overlapping sets, each set with size $\frac{1+r}{6}|V|$ as shown in Figure 14. Each node set is assigned to an agent. Then agent A_i holds the nodes from position $(i-1)\cdot\frac{1+r}{6}|V|+1$ to position $i\cdot\frac{1+r}{6}|V|$ in the sorting list. The nodes from position $i\cdot\frac{1+r}{6}|V|$ to $i\cdot\frac{1+r}{6}|V|$ are stored both on A_i and A_{i+1}. For any two adjacent agents A_i and A_{i+1}, there are $\frac{r}{6}|V|$ nodes stored on both of them. $r|V|$ nodes are stored on multiple agents.
   (c) We assign interactions to data agents. For the interactions that their participants appear in both A_i and A_{i+1}, we randomly assign $1/2$ of them on A_i and assign the left $1/2$ on A_{i+1}. For an interaction that no data agent holds all its participants, we randomly assign it to an agent who has at least one of its participants by adding its other participants to that agent.

   It is obvious that this method generates a distributed storage with node overlapping ratio at least r. In our experiment, we change r from 5 % to 80 % to test the three criteria.



(a) Information Loss (b) Distance of *DD* (c) Error of queries

Figure 15 Real distributed case

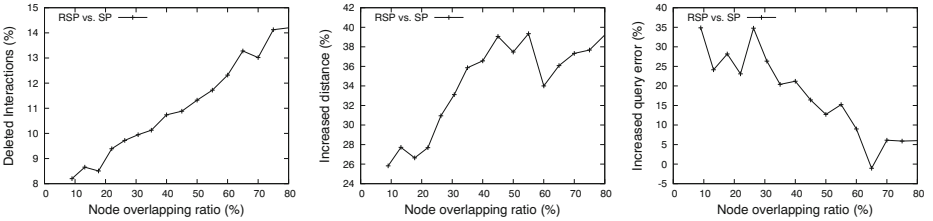
5.4 Result

5.4.1 Result on Real Distributed Case

We compare the performance of SP and RSP under different privacy parameter ks. Figure 15a shows the result of information loss. From the result we can see, RSP deletes 13 % to 17 % interactions. This is a fairly significant information loss. The published graph by RSP fails to correctly represent the original graph. Another phenomenon is that RSP deletes more interactions with the increasing of k. This is because larger k means stronger protections, which needs stronger protection conditions. Thus RSP needs to omit more interactions to satisfy the clustering safety condition CSC. Figure 15b shows the comparison of degree sequence distance. RSP performs 45 % to 80 % worse than SP. The published graph by RSP is much worse than SP when estimating by the degree distribution. With the increasing of k, the published graph represents less and less information of the original graph. Compared with RSP, the benefits of using SP will decrease when k increases. Figure 15c shows the results of queries. In most cases, the RSP performs 10 % to 25 % worse than SP. Since the random sampling and random query selections, the result vibrates a lot. In most cases, SP gets a much better graph than RSP.

5.4.2 Result on Simulated Distributed Case

For the Simulated Distributed Case, we set $k = 5$ and compare the performance of SP and RSP with different node overlapping ratio rs. Figure 16a shows the number of deleted interactions. Larger r means more overlapping between data agents and more chances to delete an interaction. In most cases, RSP deletes 10 % to 20 % interactions. Figure 16b shows the comparison of degree sequence distance. RSP performs 25 % to 45 % worse than



(a) Information Loss (b) Distance of *DD* (c) Error of queries

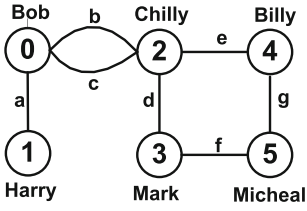
Figure 16 Simulated distributed case

SP. Figure 15c shows the results of queries. In most cases, the RSP performs 5 % to 35 % worse than SP. SP generates a graph with higher utilities than RSP. With the increasing of node overlapping ratio r, RSP will omit more interactions. Thus the "information loss" (Figure 15a) and "increased distances" (Figure 15b) rises with the increasing of r. The clustering based method used in this paper imports a lot of fake interaction information (From the clustering safety condition CSC, the number of interactions between two clusters is limited by at most k. Other possible $k^2 - k$ interactions are all fake ones.). In this case, deleting some interactions may reduce the three connection queries error which is caused by information losing and graph clustering. So compared with RSP the benefits of using SP decreases when k increases on error of queries.

From the testing results, we can find SP exhibits benefits on both information completeness and the utilities of the published graphs. Firstly, the RSP deletes roughly 13 %-19 % interactions, publishing a graph which loses such a large portion of information is not acceptable. While SP guarantees no information loss in the published graph. Secondly, the utilities of the graph generated by SP are much better than RSP. It is necessary to design an SMC protocol such as SP for the privacy preserving graph publication problem. Actually, since SP has the same effect as running the state-of-art centralized graph construction algorithm on the complete original graph, the published graph generated by SP can be seen as the best.

6 Adaptation to simple graph protecting model

The proposed protocol design idea can be easily adapted to other graph protection models on simple graphs (The graph only contains vertices, edges and vertex attributes). In this section, we demonstrate how to design an efficient algorithm and the corresponding SMC protocol for the clustering protection model [15]. We select this model because it can avoid the node re-identification with any background knowledge (i.e., it allows an attacker to use arbitrary subgraph to do the attack), which provides the strongest protection to nodes. For a simple graph, by clustering, the published graph G' only contains super nodes (clusters) where each super node represents multiple nodes in the original graph. Figures 19a and 19b are two clustered graphs of Figure 17. Each super node in these two clustered graphs represents three original nodes in Figure 17. The weight on each super edge in Figures 19a and 19b means the number of original edges this edge represents. For example, in Figure 19b, super node A contains original nodes $\{0, 1, 2\}$, since there are three edges $a(0, 1)$, $b(0, 2)$ and $c(0, 2)$ between nodes $\{0, 1, 2\}$, A has a self-edge with weight 3. It is easy to see when a clustered graph is published, if the minimum cluster size is k, with whatever background knowledge, an attacker can only re-identify an individual with probability at most $\frac{1}{k}$. When



Node ID	Node Attributes
0	Africa, Bachelor
1	American, Master
2	European, Bachelor
3	Asia, Phd
4	American, Phd
5	Asia, Bachelor

Edge ID	Edge Attributes
a	email: 1024 Byte on 01/03/10
b	comment: 936 Byte on 11/08/10
c	email: 524 Byte on 12/07/10
d	blog: subscribed on 15/06/10
e	blog: subscribed on 02/08/10
f	email: 776 Byte on 23/07/10
g	comment: 1246 Byte on 04/08/10

Figure 17 The original graph

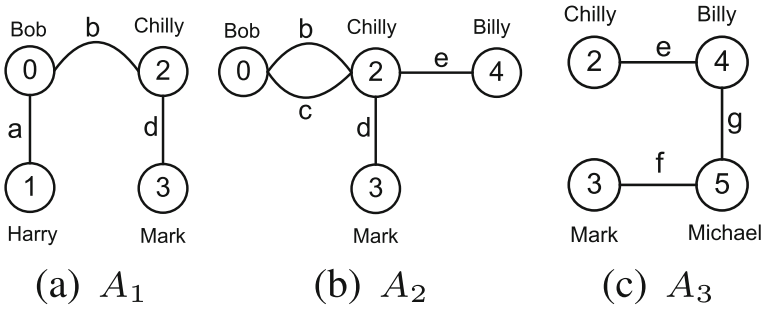


Figure 18 The distributed storage

using a clustered graph, a customer can sample a graph which is consistent with the clustered one. Here, consistent means the edges between nodes in the sampled graph must match the weights on the edges in the clustered graph. Figure 20 shows three sampled graphs of Figure 19b. Normally, when a user wants to do a computation on the clustered graph, in order to get an accurate result, he samples a group of graphs and compute the average result on these graphs.

So the protection model targets to generate a clustered graph with the minimum number of possible sampling graphs when giving the minimum cluster size k [15]. The number of possible sampling graphs is called the number of possible worlds. Suppose the super node set of a clustered graph G' is V', for the heterogeneous graph model, the number of possible worlds $|W(G)|$ can be computed as: $\prod_{X\in V'}(\frac{1}{2}|X|(|X|-1))^{d(X,X)}\prod_{X,Y\in V'}(|X||Y|)^{d(X,Y)}$, where $d(X,Y)$ denotes the weight on edge (X,Y). For example, $|W$ (Figure 19a)$| = (3)^2(9)^5 = 3^{12}$ and $|W$ (Figure 19b) $| = (3)^3(3)^2(9)^2 = 3^9$. Figure 19b is much better than Figure 19a since $|W$ (Figure 19b)$|$ is $\frac{1}{27}$ of $|W$ (Figure 19a)$|$.

Hay [15] used a stochastic method to partition nodes into clusters. They designed a simulated annealing algorithm (SA). However, the stochastic algorithms like SA need to compute too many steps to achieve an acceptable solution. Besides the efficiency problem, the long computation sequence provides many information. This characteristic makes the stochastic

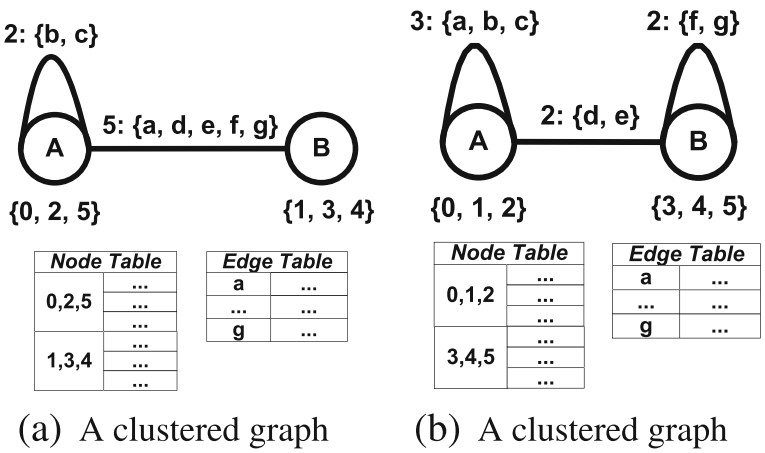


Figure 19 Examples of the clustering model

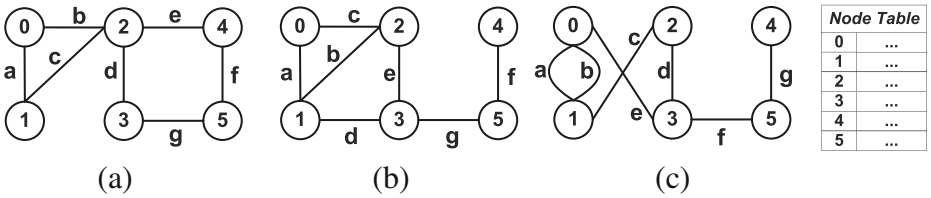


Figure 20 Example of sampled graphs

algorithms unsuitable for the SMC Problem. The server who runs the algorithms can easily find out certain information stored on other servers through the computation logs. For example, one basic operation of the SA algorithm designed by [15] is to move one node from one cluster to another cluster. If after several steps, the clusters are changed from Figure 19a to Figure 19b, the connection information such as there exist edges $f(3, 5)$ and $g(4, 5)$ can be deduced.[9]

6.1 Problem definition

The problem to be solved here for simple graphs is:

Problem 1 Given a distributively stored graph $G(V, E)$ which is hold by m different data agents, and a constant number k, generate a clustered graph $G'(V', E', AV', AE)$ from G that satisfies:

- Correctness:
  - Privacy: $\forall u_s \in V', |u_s| \geq k$
  - Utility: $|W(G')|$ is the smallest among all clustered graphs which satisfy the Privacy requirement.
- Security: The intermediate results any participant gets during the computation do not provide any more information than the final published graph.

Problem 1 can be proved to be an NP Hard problem which does not have a polynomial time approximate algorithm with constant factor from the classic Balanced Graph Partition Problem (BGP Problem) [3].

6.2 Algorithm design

We observed the following two properties to minimize $|W(G)|$:

- The size of clusters should be as small as possible.
- The fewer cross cluster edges, the better a clustered graph is.

From the first property, the clusters should have as small size as possible. So for the optimal solution, for most clusters, $\frac{1}{2}|X|(|X| - 1) = \frac{1}{2}k(k - 1)$ and $|X||Y| = k^2$. Since $k^2 > \frac{1}{2}k(k - 1)$, fewer cross cluster edges are preferred. We should design a clustering

[9] In Figure 19a, there's no edge between {1, 3, 4}. In Figure 19b, there are two edges between {3, 4, 5}. Then, the two edges must be (3, 5) and (4, 5).

algorithm which makes each cluster as small as possible and makes the number of cross cluster edge as few as possible to minimize the number of possible worlds. The above requirements are similar to the BGP Problem. We design our heuristic clustering algorithm HEU_CLUSTERING based on the Fiduccia-Mattheyses Algorithm [2, 3]. The details is shown in Algorithm 4. HEU_CLUSTERING is a top-down recursive partition algorithm which starts with the largest cluster which contains all the nodes in G. The input of this algorithm is a cluster X, the constant privacy parameter k and the current partition (cluster) set C.

Algorithm 4 Heuristic Clustering Algorithm: HEU_CLUSTERING

```
   Input: Cluster X, constant k, cluster set C
1  if |X| < 2k ;                                                    /* Stage 1 */
2  then
3      C = C ∪ X;
4  integer size_1 = (⌈|X|/2⌉ mod k == 0)? ⌈|X|/2⌉ : ⌊|X|/2k⌋ × k;
5  Randomly Split X into two clusters X_1 and X_2 with |X_1| = size_1;
6  Compute the gain of each node;
7  while true ;                                                     /* Stage 2 */
8  do
9      u_1 = the node in X_1 with the maximum gain that has not been moved;
10     u_2 = the node in X_2 with the maximum gain that has not been moved;
11     if (u_1.gain ≥ u_2.gain) ∧ (u_1.gain > 0) then
12         Algorithm 5 (X_1, X_2, u_1);
13     if (u_2.gain > u_1.gain) ∧ (u_2.gain > 0) then
14         Algorithm 5 (X_2, X_1, u_2);
15     else
16         break;
17     Recompute the gains of nodes in X_1 and X_2;
18 HEU_CLUSTERING(X_1, k, C) ;                                       /* Stage 3 */
19 HEU_CLUSTERING(X_2, k, C);
```

The algorithm works as follows. We first check whether X's size is less than $2k$. If X's size is less than $2k$, X cannot be split into two clusters. Therefore, we add X into C. Otherwise we randomly split X into two new clusters X_1 and X_2. In order to split X as balance as possible and generate as more clusters as possible, we set X_1's size as the closest value to $\frac{|X|}{2}$ which can be divided by k. This guarantees the final clusters we computed at most has one cluster with size bigger than k, which maximizes the number of clusters. After splitting X into two new clusters X_1 and X_2, for each node u, we compute the gain of moving this node, which is the number of edges between X_1 and X_2 that can be reduced if moving u from the current cluster to the other cluster.[10] For example, in Figure 21a, nodes 0, 1, 3 are put in one cluster and nodes 2, 4, 5 are put in the other cluster, the number next to each node is its gain. For node 5, it has one cross cluster edge and one inner cluster edge. So moving node 5 to the other cluster does not reduce the number of cross cluster edges. Node 5's gain is 0. For node 2, it has three cross cluster edges and one inner cluster edge, so node 2's gain is +2. Since moving a node with the positive gain helps to reduce the number of cross cluster edges, Algorithm HEU_CLUSTERING switches nodes between X_1 and X_2 in line 6-18 until no more benefit can be gotten. Each time,HEU_CLUSTERING moves one node with the maximum gain that has not been moved in its corresponding cluster. Since we need to keep the size of X_1 and X_2, whenever we move a node from X_1 to X_2, we move one

[10]Gain represents the benefit when moving u from the current cluster to the other cluster.

Algorithm 5 Switch two nodes

Input: Cluster X_a, Cluster X_b, node u
Move u from X_a to X_b;
Recompute the gains of nodes in X_b;
u' = *the node in* X_b *with the maximum* $gain$ *that has not been moved*;
Move u' from X_b to X_a;

node back from X_2 to X_1 in the next step. When a node is moved, the gains of the nodes connecting with this node should be re-computed. An example is shown in Figure 21b, when node 3 is moved, the gains of nodes 2,3 and 5 are changed.

Each time, when a cluster X is split into two clusters X_1 and X_2, only the nodes with edges between X_1 and X_2 may be moved. In the worst case, HEU_CLUSTERING runs with $O(|E|)$ complexity. Since social networks are sparse, the time complexity of HEU_CLUSTERING is $O(|V|)$. In the experiment part, we will show this heuristic algorithm generates quite well results by comparing it with the SA algorithm [15] through testings both on real datasets and synthetic datasets.

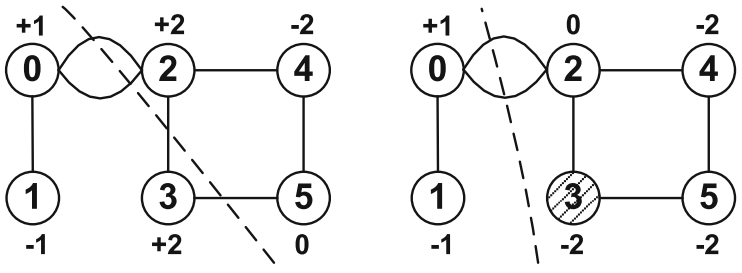
Algorithm HEU_CLUSTERING stops switching when no positive gain can be gotten. When Algorithm HEU_CLUSTERING splits a cluster X into two clusters X_1 and X_2, for the edges in X, at most $\frac{1}{3}$ of them may cross X_1 and X_2. So the number of cross cluster edges is at most $\frac{1}{3}|E| + \frac{1}{3}\frac{2}{3}|E| + \frac{1}{3}(\frac{2}{3})^2|E| + ... + \frac{1}{3}(\frac{2}{3})^{(log_k^{|V|}-1)}|E| = (1-(\frac{2}{3})^{log_k^{|V|}})|E|$. In the clustered graph G', if $|V|$ can be divided by k, the number of G''s possible world is at most $(\frac{1}{2}k(k-1))^{(\frac{2}{3})^{log_k^{|V|}}|E|}(k^2)^{(1-(\frac{2}{3})^{log_k^{|V|}})|E|} \leq (\frac{1}{2})^{(\frac{2}{3})^{log_k^{|V|}}|E|}k^{2|E|}$.

6.3 Protocol design

In this section, we show how a protocol using the similar idea as Section 4 can be developed. We firstly introduce the protocol design strategy in Section 6.3.1. Then we give the details of our protocol design in Section 6.3.2.

6.3.1 Strategies

As shown in Problem 1, the protocol should guarantee the correctness (i.e. Privacy and Utility) and the Security of the computation. To guarantee the correctness, the protocol should exactly implement the HEU_CLUSTERING algorithm. To guarantee the Security, the protocol should make sure for each participant, the intermediate information it gets during the



(a) Gain computation example (b) Gain change example

Figure 21 Examples of HEU_CLUSTERING

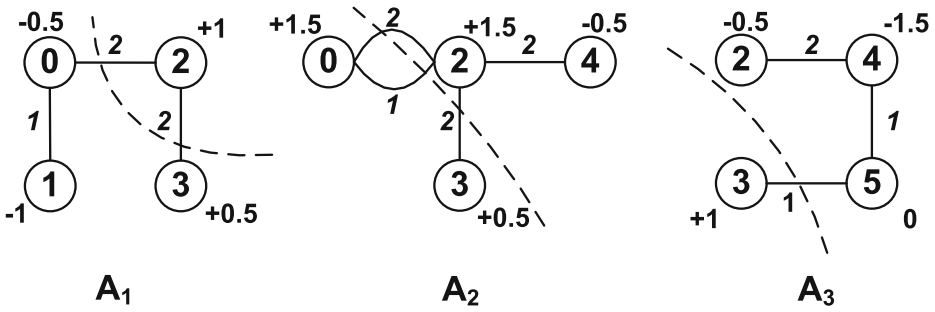


Figure 22 The distributed storage with duplicated weights

computation should not beyond the final published graph. The computation and comparison of the node gain values are two elementary operations in Algorithm 4. These operations may leak local connection information of data agents. In order to make sure that no local connection information is released, we have to implement secure computing and comparing of the node gain values (Fig. 21).

Gain Value Computation on Local Information We can make each data agent compute the gain values only based on its local connection information. It is easy to see the gain value of each node is the sum of local gain values if the edges do not have duplicate copies. However, it is possible that the edges may be stored multiple times on different data agents. When combining the local computed results, the duplicate computation on one edge should be removed. We assume each edge and node has a weight that represents how many duplicate copies of this node/edge have been stored in the system. We call these weights *duplicated weights*. Duplicated weights can be generated using the protocol in Appendix D. Figure 22 shows an example of Figure 18 where the number on each edge is this edge's duplicated weight.

Then, when computing the gains based on local information, we add/reduce the reciprocal of each edge's duplicate weight to avoid the duplicated computation. In Figure 22, the number beside each node is the gain of this node based on the corresponding local information. If we add the gains of a node on different data agents, the result is exactly the gain of this node.

Secure Gain Value Computation The above method makes sure the computation of gains does not release any local connection information. While, it is still not enough to achieve the security requirement since the exact gain values are still available to the participant who adds all the local gains. When a node u is moved, only the gains of nodes that connect with u change. The participant who knows the gains can easily get the connection information. For example, in Figure 21, at the beginning, node 2's gain is $+2$ and node 5's gain is 0. After node 3 is moved, node 2's gain becomes 0 and node 5's gain becomes -2. The participant who gets the above information can immediately conclude there exists an edge $(2, 3)$ and an edge $(3, 5)$. So to guarantee the security, the exact gain values should also be hidden.

The mechanism we use is adding noise numbers. Instead of doing the computation on real value x, one participant p_x generates a random number r and another participant p_y does the computation on the value $x + r$. We make sure the computation can be finished by letting p_x only know r and p_y only know $x + r$.

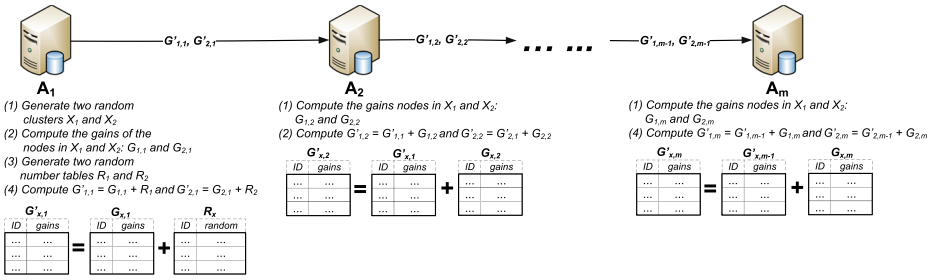


Figure 23 Protocol: Stage 1

Secure Gain Values' Comparison In our algorithm, the computation on the gain values is to select the node u with the maximum gain value and determine if u's gain value is bigger than 0 or not. So the only computation is the comparison between gain values. We use Yao's Millionaires' Protocol as a black box when computing the node with the maximum gain and determining whether this maximum gain is bigger than 0. When we want to compare two gain values x_1 and x_2 when p_x knows r_1, r_2 and p_y knows $(r_1 + x_1)$, $(r_2 + x_2)$, we can use Yao's Millionaires' Protocol to compare $((r_1 + x_1) - (r_2 + x_2))$ and $(r_1 - r_2)$. By doing this, the comparison computation can be finished without reveal the realing values of x_1 and x_2.

6.3.2 Protocol details

Clustering In Algorithm HEU_CLUSTERING, the lines 6,9,10,13,15,17 need the cooperation of all data agents and may violate the security. We divide Algorithm HEU_CLUSTERING into three stages as shown in Algorithm 4. In this section, we introduce how our protocol works for the first two stages respectively since Stage 3 is the recursive invoking of Algorithm HEU_CLUSTERING. Before the protocol is invoked, we assign unique ids to all nodes/edges using the protocol in Appendix E without releasing any real node/edge identifier information. When ids are assigned, the node number is known by each agent.

Stage 1 No specifically, we use data agent A_1 to control the schedule of Algorithm HEU_CLUSTERING. The working process of Stage 1 is shown in Figure 23:

(i) Suppose the current cluster is X, the data agent A_1 checks whether X's size is less than $2k$. If X's size is equal to or bigger than $2k$, A_l randomly divides X into two clusters X_1 and X_2 as the same as lines 3 and 4 in Algorithm 4.

(ii) For each node in X_1, A_1 generates a random number and stores these random numbers in table R_1. A_1 also generates random numbers for the nodes in X_2 and stores them in table R_2. The node ids in table R_1 and R_2 are sorted.

Algorithm 6 Find the node with the maximum $gain$ in C_x on A_1

```
1 pos_max = 0;
2 value_max = G'_{x,m}[0];
3 for i = 1; i < |G_{x,m}|; i++ do
4     A_1 and A_m compare the gains of the two nodes at positions pos_max and i using the Millionaires'
      Protocol (By comparing G'_{x,m}[i] − G'_{x,m}[pos_max] and R_x[i] − R_x[pos_max]);
5     if The node at position i has bigger gain then
6         pos_max = i;
7         value_max = G'_{x,m}[i];
```

(iii) A_1 computes the gains of the nodes in X_1 based on its local storage and stores the results in table $G_{1,1}$. The gains of the nodes in X_2 are computed and stored in table $G_{2,1}$. The node ids in table $G_{1,1}$ and $G_{2,1}$ are also sorted. Then A_1 computes two tables $G'_{1,1} = R_1 + G_{1,1}/G'_{2,1} = R_2 + G_{2,1}$ and sends $G'_{1,1}/G'_{2,1}$ to A_2.

(iv) Each data agent A_i does the same operations as A_1 in step 3. A_i sends the computed two tables $G'_{1,i}$ and $G'_{2,i}$ to the next data agent A_{i+1} if A_{i+1} exists.

After finishing the above steps, data agent A_m gets two tables $G'_{1,m}$ and $G'_{2,m}$. For a node u and any table T, we use $T(u)$ to represent u's corresponding value in T. For each node u in X_1, u's $gain$ is $G'_{1,m}(u) - R_1(u)$ and for each node v in X_2, v's $gain$ is $G'_{2,m}(v) - R_2(v)$.

Stage 2 In this stage, we need to switch nodes between X_1 and X_2 until no benefit can be gotten from the moving. A_1 controls the whole moving process. It is important to securely find the node with the maximum gain in each cluster and update the gains of nodes. As discussed in Section 6.3.1, we use the Millionaires' Protocol to compare two nodes' gains. We use Algorithm 6 to find a node in a cluster X_x with the maximum $gain$. The Millionaires' Protocol is used to compare two nodes' gains in Algorithm 6. To determine whether the current maximum $gain$ is bigger than 0 in Algorithm HEU_CLUSTERING line 10, we use the Millionaires' Protocol to check whether $G'_{x,m}[pos_{max}]$ is bigger than $R_x[pos_{max}]$.

After moving a node, the gains of nodes should be re-computed. The updating of gains follows the same operations as the 2-5 steps in Stage 1. During updating, we only compute the gains of nodes that have not been moved.

Super edge and attribute generation After running Algorithm 4, A_1 gets the cluster set C. The next step is to generate super edges and attributes for super nodes. This step is the same as Section 4.5.

It is clear that the protocol we designed exactly implements Algorithm 4. So the protocol satisfies the correctness requirement.

Theorem 6 *The above protocol satisfies the Security requirement.*

We prove the Security of our protocol by analyzing the intermediate information each participant gets. The details can be found in Appendix F.

6.4 Evaluation

In this section, we use experiments to answer two questions:

- How good is algorithm HEU_CLUSTERING: We show the effect of algorithm HEU_CLUSTERING by comparing its results with the SA algorithm [15] on several real datasets and synthetic datasets.
- The advantage of using distributed computation to generate the published graph: If there is no protocol to support the distributed computation, we can use the following method to construct a published graph. We can request each data agent to generate a clustered graph only based on its local content and finally combine the clustered graphs to a large published graph G_d (The vertices set different data agents work on are not overlapped). We show the published graph generated by the distributed computation is much better than G_d.

We test four real datasets: Cora (www.cs.umd.edu/projects/linqs/projects/lbc/index.html, 2708 nodes and 5429 edges), DBLP (kdl.cs.umass.edu/data/dblp/dblp-info.html, 6000 nodes and 31985 edges), Arnet (arnetminer.org, 6000 nodes and 37848 edges) and ArXiv

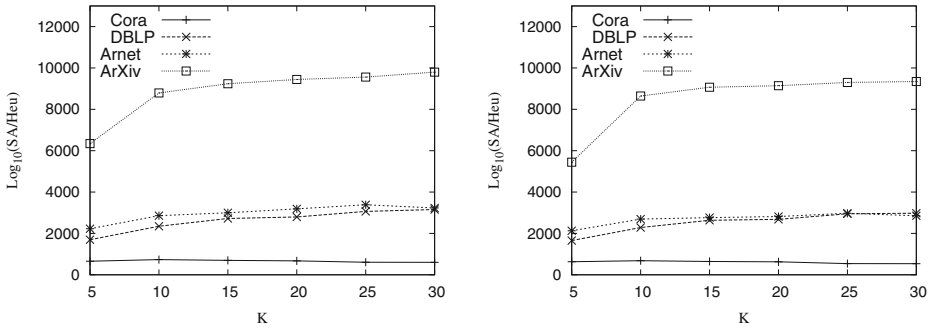


Figure 24 The effect of Algorithm HEU_CLUSTERING on real datasets

(arXiv.org, 28868 and 74086 edges). We also tested several synthetic datasets, we use the R-MAT graph model [6] with the same parameters as Zhou's paper [28] to generate 3 synthetic datasets with 1000, 2000 and 3000 vertices respectively.

6.4.1 The effect of algorithm HEU_CLUSTERING

For a problem which does not have an approximation algorithm with constant factor, normally either heuristic algorithms or stochastic algorithms are used. Hay [15] designed a simulated annealing (SA) algorithm to find a clustered graph with the minimum number of possible worlds. We would like to use this method as the baseline. Suppose the graph generated by Algorithm HEU_CLUSTERING is G'_{heu} and the graph generated by SA [15] is G'_{SA}, we compare G'_{heu} and G'_{SA} with different ks by showing the value $log_{10}(\frac{|W_{G'_{SA}}|}{|W_{G'_{heu}}|})$. [15] showed the SA can get "good enough" result with at most $100|V|$ steps ($|V|$ is the number of nodes in the graph). We use the results of SA when it runs $100|V|$ steps and $150|V|$ steps to do the comparision respectively.

Figure 24 shows the testing results on the real datasets. Figure 25 shows the testing results on the synthetic datasets. From the results we can see, for all the graphs, even after SA

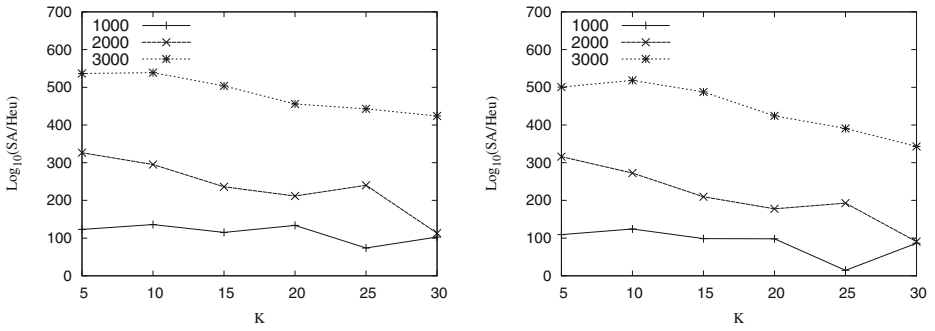


Figure 25 The effect of Algorithm HEU_CLUSTERING on synthetic datasets

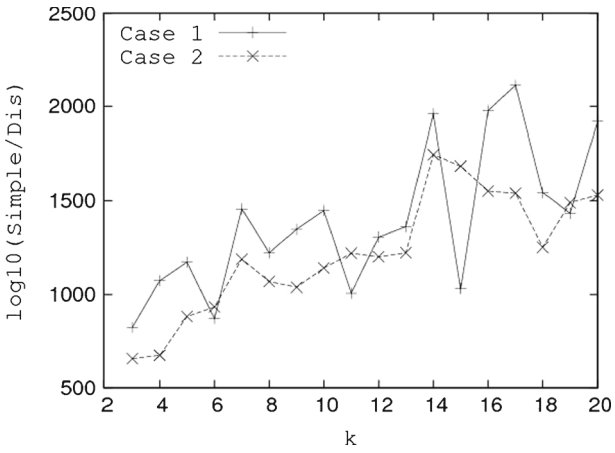


Figure 26 The effect of the distributed computation

algorithm running many steps, the results are still much worse than the heuristic algorithm. Algorithm HEU_CLUSTERING performs quite well.

6.4.2 The effect of distributed computation

If there is no distributed computation protocol, we can use the following naive approach to generate a published graph which lets each data agent do the computation only based on its local content.

- A_1 generates a clustered graph G'_1 using its local content and passes G'_1 to A_2;
- A_2 clusters all the nodes stored on it but not covered by G'_1 and generates a new clustered graph G'_2 by adding the new super nodes into G'_1. A_2 passes G'_2 to A_3;
- Each A_i does the same operations as A_2 and finally A_m gets the clustered graph which covers all the nodes.

Suppose the published graph generated by the above method is G'_{simple} and the graph generated by our protocol is G'_{dis}, we show $log_{10}(\frac{|W_{G'_{simple}}|}{|W_{G'_{dis}}|})$ under different ks.

We tested the two distributed storage cases for ArXiv dataset as the same as Section 5.3. To avoid the influence of the computation sequence among A_1 to A_6, we try all the possible computation sequences and use the best one as G'_{simple}. The testing results are shown in Figure 26. From the result we can see for both two cases and all ks, $|W_{G'_{dis}}|$ is only $\frac{1}{10^{600}}|W_{G'_{simple}}|$ to $\frac{1}{10^{2000}}|W_{G'_{simple}}|$, a large benefit can be obtained by using our distributed computation protocol.

7 Conclusion

In this paper, we target on the secure multi-party privacy preserving social network publication problem. We design two protocols SP and RSP for the latest clustering based graph protection model. The first protocol SP follows the same logic as the centralized algorithm.

Which makes SP generates a published graph with the same quality as the one generated in the centralized environment. RSP reduces the communication cost of SP a lot by publishing a graph with less utility than the one generated in the centralized environment. We show the cost-utility trade-off between these two protocols on a real data set in the experiment. SP is designed for small size social networks or the distributed system which has a high bandwidth. RSP is designed for large size social networks. As far as our knowledge, this is the first work on SMC privacy preserving graph publication which against the structure attack. In our extension work, we'll study how to enhance the current solution to handle the malicious or accessory agents. How to design SMC protocols for the editing based graph protection models will be another interesting future work.

In this paper, we target on the secure multi-party privacy preserving social network publication problem. We design an SMC protocol SP for the latest clustering based graph protection model. SP can securely generate a published graph with the same quality as the one generated in the centralized environment. As far as our knowledge, this is the first work on SMC privacy preserving graph publication against the "structure attack". In the future, one interesting direction is to study how to enhance the current solution to handle the malicious or accessory agents. How to design SMC protocols for the editing based graph protection models will be another interesting direction. Another interesting direction is to design more efficient protocols since the current solution passes $|V| \times |V|$ matrix between agents. The communication cost is high. One possible method is to use the compression techniques such as delta-compression to reduce the size of message to be passed.

Acknowledgment This work is supported in part by the Hong Kong RGC Project MHKUST602-12, National Grand Fundamental Research 973 Program of China under Grant 2012-CB316200, Microsoft Research Asia Gift Grant and Google Faculty Award 2013.

Appendix

A Method for duplicated weights computation

Since each node and interaction has a unique id, A_1 generates two random vectors RV_{dup} and RI_{dup}. RV_{dup} has size $|V|$ and RI_{dup} has size $|I|$. Then A_1 generates two variable vectors V_{dup} and I_{dup} with $V_{dup} = RV_{dup}$ and $I_{dup} = RI_{dup}$. $V_{dup}[i] - RV_{dup}[i]$ will be used to represent the duplicated weight of node i. $I_{dup}[i] - RI_{dup}[i]$ will be used to represent the duplicated weight of interaction i. Each A_i does the following operations:

- For each node i stored by A_i, $V_{dup}[i] = V_{dup}[i] + 1$;
- For each interaction i stored by A_i, $I_{dup}[i] = I_{dup}[i] + 1$;
- Passes V_{dup} and I_{dup} to A_{i+1} if A_{i+1} exists.

After doing this, A_l sends the V_{dup} and I_{dup} to A_1. Finally, A_1 gets the duplicated weights of all nodes and interactions by computing $V_{dup} - RV_{dup}$ and $I_{dup} - RI_{dup}$. A_1 passes the result to each A_i and A_i stores the duplicated weights for the nodes and interactions in G_i.

B Security of SSSP

Proof It is obvious that the sorting is based on node degrees. Since in this step, only node degrees are passed, we just need to check whether specific degree values or sorting order on real ids is disclosed or not. During the computation, only S_1 and S_2 have the sorting list L on $\pi(id)$s. Since S_1 and S_2 do not know π, they cannot get the sorting list on real ids. So no participant gets the sorting order on real ids. Next we show for each participant, no node degree is disclosed.

- For agent A_1, it only knows π, D and the degrees of the nodes stored on it. So no node degree information is disclosed to it;
- For any data agent A_i $(1 < i < l)$, A_i has an intermediate result of D'. Since D is not available for any A_i all the time, A_i does not have the information in $D' - D$. Thus A_i cannot obtain any node degree information;
- Agent A_l knows the final result of D' and π. Since it does not have D, it gets no node degree information;
- The server S_1 has $\pi(D')$, it computes L based on $\pi(D') - \pi(D)$. Since the Millionaires' Protocol is used to do the sorting, any information in $\pi(D)$ is not released to S_1. S_1 gets no node degree information;
- The server S_2 has $\pi(D)$, it helps S_1 to compute L based on $\pi(D') - \pi(D)$. Since the Millionaires' Protocol is used to do the sorting, any information in $\pi(D')$ is not released to S_2. S_2 gets no node degree information;

So $SSSP$ sorts all nodes on $\pi(id)$s without violating the Security requirement. □

C Security of SCSP

C.1 Proof of Theorem 2

Proof We prove this theorem by analyzing all the connection cases.

- If any two nodes u and v in C have a connection, then $E_u[v] = 1$ and $E_v[u] = 1$. Since $E_u[u] = 1$ and $E_v[v] = 1$, we get $E_C[u] > 1$ and $E_C[v] > 1$.
- If there is a node x which connects two nodes u and v in C, then $E_u[x] = 1$ and $E_v[x] = 1$. Thus $E_C[x] = E_u[x] + E_v[x] + ... > 1$.
- If there is no connection between any two nodes u and v, and moreover, u and v do not connect to the same node, then for any $t \in [1, |V|]$, there exists at most one node which satisfies $E[t] = 1$. Thus, E_C contains only 0 and 1. There is no $E_C[t] > 1$.

From the above analysis under all cases, the condition in Theorem 2 can correctly check the violation of CSC. At the same time, the satisfaction of CSC is also correctly checked. □

C.2 Security of SCSP

Before proving the Security of $SCSP$, we give a lemma to show it is secure to let a participant know the computation information on $\pi(id)$s. Then the $SCSP$'s Security can be proven based on this lemma.

Lemma 1 *For any participant p, p knows the computation information on $\pi(id)$s does not violate the Security requirement.*

Proof p knows the computation information on $\pi(id)$s, then p may know the clusters on $\pi(id)$s as well. In the worst case, by comparing with the published graph, a cluster c' on $\pi(id)$s is exactly mapped to one and only one cluster c in the published graph. Suppose for any two nodes u' and v' in $\pi(id)$s, the clusters they belong to are $c'_{u'}$ and $c'_{v'}$ respectively. The mapping clusters of $c'_{u'}$ and $c'_{v'}$ in the published graph are c_u and c_v. Even when an attacker finds $\neg CSC(u', v')$, all the nodes in c_u are the candidates for u' and all the nodes in c_v are the candidates for v'. This does not violate any privacy objective of the S-Clustering model. So p knows the computation information on $\pi(id)$s does not violate the Security requirement. □

Next, we prove that $SCSP$ clusters all nodes without violating the Security requirement.

Proof The property of the connection vector guarantees that all nodes are correctly clustered. Next we show for each participant, no specific connection information is released to it. For the agents who do clustering, we prove they at most can learn the computation information on $\pi(id)$s, then based on Lemma 1, the Security is proven. We analyze the participants one by one as follows:

- For agent A_1, in the first step's computation, it knows π, MR and MR'_{A_1}. Since MR' is not available to A_1, no connection information is released to it. In the second step, it receives π_2 and LN_2, since the numbers in LN_2 are randomly generated by S_3, A_1 gets no relationship between LN_2 and MR' to guess MR', no connection information is released at this step too;
- For agent $A_i(1 < i < l)$, A_i has an intermediate result of MR'. Since MR is not available for A_i all the time, A_i does not have the information in $MR' - MR$. Thus A_i cannot learn any connection information;
- For agent A_l, its Secure Computation in the first step can be analyzed as the same as A_i $(1 < i < l)$. In the second step, A_l knows π_2 and LN'_2. Since the number in LN'_2 is randomly generated by S_3, LN'_2 provides no information of MR to A_l. So A_l gets no connection information;
- For the agent S_1, it knows $\pi(NMR')$, since both π and $\pi(NMR)$ are not available for S_1, $\pi(NMR')$ does not provide any connection information to S_1. During the clustering, S_1 gets the whole computation results on $\pi(id)$s, as shown in Lemma 1, this does not violate the Security. Since S_3 only returns "cannot cluster" or "can cluster" to S_1, S_1 gets no more connection information between $\pi(id)$s besides the computation information on $\pi(id)$s;
- S_2 gets $\pi(NMR)$, since it does not know π and $\pi(NMR')$, $\pi(NMR)$ does not release any connection information. During the computation in the third step, S_2 only provides the corresponding noise vector. At most S_2 can know some computation information on $\pi(id)$s;
- S_3 receives $\pi_2(L_2)$ and $\pi_2(L'_2)$ in the second step and generates two random vectors according to them. Since S_3 does not know π_2 and each $\pi_2(L'_2)[i] - \pi_2(L_2)[i]$ is either 0 or a random positive number, $\pi_2(L'_2)[i] - \pi_2(L_2)[i]$ provides no specific connection information to S_3. In the third step, S_3 receives two vectors ER'_C and R'_C each time. The two vectors are permuted by a random pattern π' and a random vector R' is added

into ER'_C and R_C, too. Firstly, since the result is permuted by π', S_3 can only conclude "can clustering" or "cannot clustering", therefore, no specific connection information on $\pi(id)$s is released. Secondly, since each time a new random vector is added into ER'_C and R'_C, S_3 cannot get relationship between different rounds or relate them to the two random vectors S_3 generated in the second step. Thus S_3 cannot deduce any connection information through all the vectors it received.

So $SCSP$ clusters all the nodes without violating the Security requirement. □

D Protocol for duplicate weights computation in simple graph model

After each node and edge has a unique id, A_1 generates two random vectors RV_{dup} and RE_{dup}. RV_{dup} has size $|V|$ and RE_{dup} has size $|E|$. Then A_1 generates two variable vectors V_{dup} and E_{dup} with $V_{dup} = RV_{dup}$ and $E_{dup} = RE_{dup}$. $V_{dup}[i] - RV_{dup}[i]$ will be used to represent the duplicated weight of node i. $E_{dup}[i] - RE_{dup}[i]$ will be used to represent the duplicated weight of edge E. Each A_i does the following operations:

- For each node i stored by A_i, $V_{dup}[i] = V_{dup}[i] + 1$;
- For each edge i stored by A_i, $E_{dup}[i] = E_{dup}[i] + 1$;
- Passes V_{dup} and E_{dup} to A_{i+1} if A_{i+1} exists.

After doing this, A_m sends V_{dup} and E_{dup} to A_1. Finally, A_1 gets the duplicated weights of all nodes/edges by computing $V_{dup} - RV_{dup}$ and $E_{dup} - RE_{dup}$. A_1 passes the result to each A_i and A_i stores the duplicated weights for the nodes/edges in G_i.

E Protocol for node/edge id assignment

Nodes/edges can be assigned with unique ids using commutative encryption [19]. Each data agent A_i creates a private commutative encryption key K_i. For any node/edge identifier $name_x$ stored on A_i, A_i sends $K_i(name_x)$ to $A_{(i+1)\%m}$ ($K_i(name_x)$ is the encrypted $name_x$ with K_i). $A_{(i+1)\%m}$ encrypts $K_i(name_x)$ to $K_{(i+1)\%m}K_i(name_x)$ and sends the result to $A_{(i+2)\%m}$. Each agent does the same operation. Finally, A_i gets $K_{i-1}...K_{(i+1)\%m}(name_x)$. Since the encryption is commutative, $K_{i-1}...K_{(i+1)\%m}(name_x) = K_1K_2...K_m(name_x)$. A_i sends $K_1K_2...K_m(name_x)$ to all the other agents. After each agent does the same operations as A_i, each data agent gets the same encrypted node/edge identifiers by key $K_1K_2...K_m$. Nodes and edges can be assigned with unique ids based on these encrypted names. During this process, no real identifier is released. Each agent directly gets the node/edge number of the integrated graph.

F Security of the protocol for simple graphs

Proof We prove the Security of our protocol by analyzing the intermediate information each participant gets. In the working of Algorithm 4,

- A_1: A_1 controls the whole running process. Each time, A_1 knows the random number tables R_1 and R_2. A_l has tables $G'_{1,1}$ and $G'_{2,1}$, for any node u, since A_1 does not know $G'_{1,m}(u)$, A_1 cannot learn any connection information around u.

- $A_i (1 < i < m)$: Each time, A_1 knows tables $G'_{1,i}$ and $G'_{2,i}$, for any node u, since A_i does not know R_1 and R_2, A_i cannot learn any connection information of u;
- A_m: A_m has tables $G'_{1,m}$ and $G'_{2,m}$, for any node u, since A_m does not know R_1 and R_2, A_m cannot learn any connection information around u.

Since A_1 generates a new group of random numbers when computing or updating the gains each time, the gain tables in different steps do not have any relationship.

In the process of super edge generation, since the edges between super nodes reported by each A_i is a sub-set of the final result, the middle results are included in the published graph.

In the process of node attribute generation,

- For the agent A_1, it only knows π and AT. A_1 gets no more information than the final result;
- For any data agent A_i $(1 < i < m)$, A_i has a middle result of AT'. Since AT is not available for any A_i all the time, A_i does not have the information in $AT' - AT$. Thus A_i cannot obtain a node u's attributes;
- For the agent A_m, it knows the final result of AT' and π, since it does not have AT, it gets no node-attribute mapping information;
- The server S_1 gets $\pi(AT')$, $\pi(AT)$ and $\pi(C)$, it can only compute $\pi(AT') - \pi(AT)$. Since the vectors are permuted and S_1 does not know π, S_1 cannot relate a node with its attribute values.

Based on the above analysis, none of the processes violates the security requirement. Since the three processes' middle results are not related, the whole protocol satisfies the security requirement. □

References

1. Aggarwal, G., Mishra, N., Pinkas, B.: Secure computation of the kth-ranked element. In: Avdances in Cryptology - Proceedings of Eurocyrpt 04, pp. 40–55. Springer (2004)
2. Aho, A.V., Hopcroft, J.E.: The Design and Analysis of Computer Algorithms, 1st edn. Addison-Wesley Longman, Boston (1974)
3. Andreev, K., Räck, H.: Balanced graph partitioning. In: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '04, pp. 120–124. ACM, New York (2004)
4. Bhagat, S., Cormode, G., Krishnamurthy, B., Srivastava, D.: Class-based graph anonymization for social network data. Proc. VLDB Endow. **2**, 766–777 (2009)
5. Campan, A., Truta, T.M.: A clustering approach for data and structural anonymity in social networks. In: PinKDD'08 (2008)
6. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-mat: a recursive model for graph mining. In: SDM (2004)
7. Cheng, J., Fu, A.W.-c., Liu, J.: K-isomorphism: privacy preserving network publication against structural attacks. In: Proceedings of the 2010 International Conference on Management of Data SIGMOD '10, pp. 459–470. ACM, New York (2010)
8. Clifton, C., Kantarcioglu, M., Vaidya, J., Lin, X., Zhu, M.Y.: Tools for privacy preserving distributed data mining. SIGKDD Explor. Newsl. **4**, 28–34 (2002)
9. Cormode, G., Srivastava, D., Yu, T., Zhang, Q.: Anonymizing bipartite graph data using safe groupings. Proc. VLDB Endow. **1**, 833–844 (2008)
10. Du, W., Atallah, M.J.: Secure multi-party computation problems and their applications: a review and open problems. In: Proceedings of the 2001 Workshop on New Security Paradigms NSPW '01, pp. 13–22. ACM, New York (2001)
11. Frikken, K.B., Golle, P.: Private social network analysis: how to assemble pieces of a graph privately. In: Proceedings of the 5th ACM Workshop on Privacy in Electronic Society, WPES '06, pp. 89–98. ACM, New York (2006)

12. Ganta, S.R., Kasiviswanathan, S., Smith, A.: Composition attacks and auxiliary information in data privacy. CoRR (2008)
13. Garg, S., Gupta, T., Carlsson, N., Mahanti, A.: Evolution of an online social aggregation network: an empirical study. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, IMC '09, pp. 315–321. ACM, New York (2009)
14. Grujić, J.: Movies recommendation networks as bipartite graphs. In: Proceedings of the 8th International Conference on Computational Science, Part II ICCS '08, pp. 576–583. Springer, Berlin (2008)
15. Hay, M., Miklau, G., Jensen, D., Towsley, D., Weis, P.: Resisting structural re-identification in anonymized social networks. Proc. VLDB Endow. **1**, 102–114 (2008)
16. Jiang, W., Clifton, C., Kantarcioglu, M.: Transforming semi-honest protocols to ensure accountability. Data Knowl. Eng. **65**, 57–74 (2008)
17. Jurczyk, P., Xiong, L.: Distributed anonymization: achieving privacy for both data subjects and data providers. In: Proceedings of the 23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security XXIII, pp. 191–207. Springer, Berlin (2009)
18. Kantarcioglu, M., Clifton, C.: Privacy-preserving distributed mining of association rules on horizontally partitioned data. IEEE Trans. Knowl. Data Eng. **16**, 1026–1037 (2004)
19. Kerschbaum, F., Schaad, A.: Privacy-preserving social network analysis for criminal investigations. In: Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society, WPES '08, pp. 9–14. ACM, New York (2008)
20. Liu, K., Terzi, E.: Towards identity anonymization on graphs. In: SIGMOD'08, pp. 93–106 (2008)
21. Vaidya, J., Clifton, C.: Privacy-preserving k-means clustering over vertically partitioned data. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD '03, pp. 206–215. ACM, New York (2003)
22. Vaidya, J., Clifton, C.: Privacy-preserving data mining: Why, how, and when. IEEE Secur. Priv. **2**, 19–27 (2004)
23. Wasserman, S., G. Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press (1994)
24. Yao, A.C.: Protocols for secure computations. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science SFCS '82, pp. 160–164. IEEE Computer Society, Washington (1982)
25. Ying, X., Wu, X.: Randomizing social networks: a spectrum preserving approach. In: SDM'08 (2008)
26. Yonglong, L., Liusheng, H., Yang, W., Weijiang, X.: An efficient protocol for private comparison problem. Chin. J. Electron., 18 (2009)
27. Zheleva, E., Getoor, L.: Preserving the privacy of sensitive relationships in graph data. In: PinKDD'07, pp. 153–171 (2007)
28. Zhou, B., Pei, J.: Preserving privacy in social networks against neighborhood attacks. In: ICDE'08, pp. 506–515 (2008)
29. Zou, L., Chen, L., Özsu, M.T.: K-automorphism: a general framework for privacy preserving network publication. Proc. VLDB Endow. **2**, 946–957 (2009)