Secure Multi-party Graph Computation

Varsha Bhat, Harsimran Singh, and S.R.S. Iyengar

Department of Computer Science and Engineering,
Indian Institute of Technology Ropar,
Punjab, India.
{varsha.bhat, sharsimran, sudarshan}@iitrpr.ac.in

Abstract. In this paper, we present a protocol to compute a friendship network of n people without revealing the identities of the people involved. The final result is an unlabelled graph which doesn't disclose the identity of the parties. As part of the protocol, we present and make use of a technique to compute a random assignment of the numbers $\{1, 2, ..., n\}$ to n people. Our work has direct applications in the data collection stage of Social Network Analysis, where, it is of vital importance to compute the underlying network without compromising on the personal information or identity of the parties involved.

Keywords: Multiparty computation, social network analysis.

1 Introduction

Advancement in technology has allowed individuals to stay connected through various online social networking platforms such as Facebook, Twitter, LinkedIn, LiveJournal, etc. These networks can be visualised as graphs of social actors (individuals and/or organisations) seen as nodes, and the dyadic ties between them as edges. These edges in the network can represent a variety of relationships such as friendship, common interests, financial exchanges, email correspondence. Although this has led to the availability of copious amounts of data related to the social interactions, accessibility of this data, however, is not at par with its availability. This is primarily because of privacy issues of the involved individuals. With the emerging interest in the analysis of these social networks, the difficulty in accessing the underlying network has raised much concern.

Social Network Analysis (SNA) can be viewed as a three step process involving the collection of data (i.e. the network), its analysis and inferring the theories that explain the patterns observed in these structures [10]. The analysis of the underlying network is beneficial for various case studies such as the transmission of diseases [13, 14] classification of the influence or popularity of individuals [3], evolution of a social network [2], the flow of information in a social network [4], measuring influence of a publication [11]. It is known that the topology of the network affects the underlying dynamics such as information flow, social behaviour, small world phenomenon, structural properties such as

existence of communities, hubs, etc. However, the usage of SNA is subject to the availability of the underlying graph. That is, only if we have the social network at our disposal can we proceed with its analysis.
The availability of the network data depends on the willingness of social actors to share their private data. While the user data is made public for analysis in some cases, the fear of sensitive information being leaked prevents the users from sharing their data. For example, [13] builds the network of HIV infected people (and their partners) in Colorado Springs, USA. This network was constructed through the process of interviewing them individually. This would require the individuals to disclose their intimate relationships with other individuals, which they may be hesitant to reveal. The study in [13] relates the structure of the network to the epidemicity of the STD which would not have been possible without the data. Most of the sexual network data are collected through interviews, surveys and hospital records. This requires that a trusted third party collects the data and anonymizes it prior to making the network public for analysis. Anonymization is a process in which, only the interconnections between the social actors is retained. Every other information about the social actors is removed (i.e. name, email-address etc.). The idea is to keep the identity of the social actors private while researchers can still use the structure of the network for analysis.

Trusting a third party with sensitive information is what prevents individuals from sharing their data privately. Thus, there is a necessity for a method to generate the underlying network such that an external agent does not participate in the construction. Hence, the anonymized network must be computed by those individuals who themselves are a part of the network. Also, the process should be such that, no individual participating in the construction should be able to learn information related to any other individual. This is precisely what constitutes a multiparty computation.
Multiparty computation in general involves a set of parties interested in computing a function of their private data. The process must ensure that nothing but the final result is revealed. Here are a few instances where a multiparty protocol to compute an unlabelled graph would be helpful.

- Sexual networks, as noted above, can be easily computed by the concerned individuals themselves.
- The underlying network of people in an office that represents a like/dislike relationship can reveal a lot about the environment being healthy or not. In such a situation, employees may not be comfortable revealing their true feelings for his/her colleagues. Studying this network can reveal the stability of professional relationship among employees in a company.
- If the owner of a company wishes to determine the optimal number of managers that he needs to appoint from amongst his current employees, he should look at the underlying network of employees and see how many hubs are present.
- The underlying structure of a social network of the type PatientsLikeMe could prove very useful for the research on various diseases [5].

The anonymization defined above (removing labels from nodes and preserving the structure) is known as naive anonymization. [1] explains how naive anonymization is not enough, i.e. some of the nodes are still re-identifiable with little prior knowledge about the network. They describe passive and active attacks to re-identify some nodes in a naively anonymized network. [15] proposes a measure called *topological anonymity* which assesses the vulnerability of a network against node re-identification. The discussion in our paper will not concern this issue as [8, 9, 12, 7, 16] describe how to modify a naively anonymized network while preserving the usefulness of the network but rendering re-identification attacks useless to a certain extent. Once we have a naively anonymized network using our protocol, individuals can use the techniques mentioned in the references above to further ensure the privacy protection before releasing the network.

1.1 Our Contribution

In this paper, we provide a method for secure computation of a graph/network. Our contribution is two folds:

– We provide a protocol that the individuals of the network can follow to generate the underlying unlabelled network and thereby eliminating the need for a trusted third party
– As a part of the protocol, we also provide a technique using which n individuals assign themselves a unique number between 1 to n, such that each individual knows only his number and none others'. This also is done without a third party.

2 Preliminaries

There are a few terminologies that are necessary before we proceed further with the rest of the sections. These terminologies are essentially adapted from [6].

– **View:** The view of a party is every bit of data that it sees during the execution of the protocol.

– **Adversary:** There is a possibility of the parties being corrupt either by sharing their views or by deviating from the protocol (i.e. not following the instructions as per the protocol). This situation is modelled by assuming the existence of a centralised adversary who is capable of corrupting some parties. That is, the adversary is able to listen to the channel over which the party is communicating as well as tamper with the data being communicated.

– **Allowed influence and leakage:** When a party gives a wrong input to the protocol, we do not consider it a deviation from the protocol and is called *input substitution.* This is because we cannot stop the parties from changing their inputs. When a party provides wrong input, *inp* to the protocol, it is not considered as a corrupt party. Instead, it is considered as an honest party whose actual input is *inp*. Thus, input substitution constitutes *allowed influence* that an adversary can have on a party. Anything other than the allowed influence is considered as *actual influence* of an adversary over a party.

It is not possible that a party does not know its own input and output. We cannot deprive any party of this information. Therefore, the input and the output of a party constitutes the *allowed leakage* of the protocol to that party. Anything that the protocol leaks to a party during its execution, other than the allowed leakage, is called the *actual leakage* of the protocol to that party.

- **Security of a Multiparty Protocol:** When is a multiparty protocol secure? A protocol is said to be secure when a party cannot know anything about the inputs of the other parties based on its view alone. In the presence of corrupt parties, the definition of security changes as follows: if the set of corrupt parties cannot know anything about the inputs of the honest parties based on their combined views, then the protocol is said to be secure. Parties can be corrupted by an adversary in following ways:
  - **Passive and active corruption:** We say a party is passively corrupted by an adversary when the adversary is only listening to the conversation of the party. Let us say the adversary has passively corrupted any t number of parties. If the adversary is unable to get any information about the inputs and outputs of the honest parties, then we call the protocol secure against passive corruption under the threshold t.

    A party is actively corrupted by an adversary if it is taking all the decisions on behalf of the party. If the active corruption of any t parties does not lead to the leakage of honest parties' input and output, then we call the protocol to be actively secure against the corruption of size t.

  - **Static and dynamic corruption:** In the case of static corruption, the set of corrupt parties remains the same throughout the execution of the protocol. In the case of dynamic corruption, the set of corrupt parties may change at each step of the protocol. However, the number of corrupt parties never exceeds the threshold t.

A party can be modelled by an *interactive agent*. An interactive agent has some input ports and some output ports for communication with other interactive agents. It also has an internal state.

The interaction between parties is modelled by an interactive agent, π_{AT}, as defined in [6]. The proof for the security of the interactive agent π_{AT} is also given in [6]. The agent π_{AT} (Fig.1) ensures that the interaction between two parties is secure and information exchanged between them is not leaked to other parties. A multiparty protocol, say π_{prt}, is nothing but a set of interactions between parties. Hence it can be modelled using an interactive agent IP_i for each party P_i and an agent π_{AT} for modelling the interactions between these parties. Interactive agent IP_i has three input ports $prt.in_i$, $AT.infl_i$, $AT.out_i$ and three output ports $prt.out_i$, $AT.leak_i$, $AT.in_i$ as given in Fig.2. The ports $prt.in_i$ and $prt.out_i$ are for the input and output of the party respectively. The ports $AT.in_i$ and $AT.out_i$ are to interact with agent π_{AT} to talk to other parties. The output port $AT.leak_i$ is used to leak the view of a corrupt party P_i. The input port $AT.infl_i$ is used to deliver the instructions to the party P_i when it is actively corrupt.

Fig. 1. Agent for party IP_i

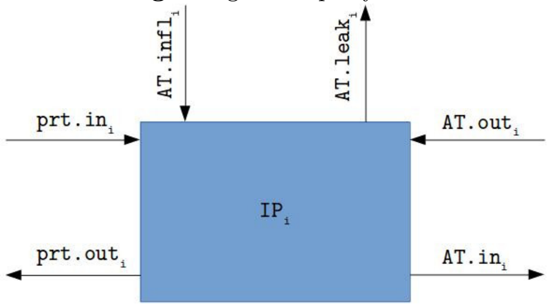
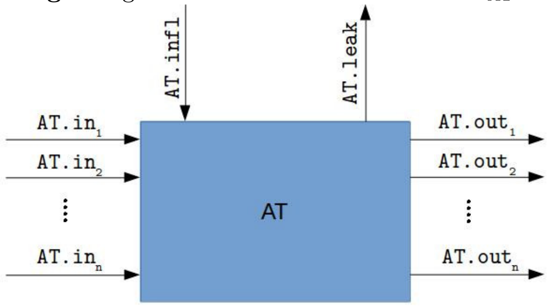
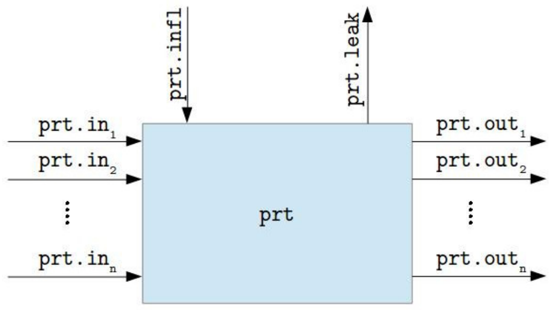


Fig. 2. Agent for Authenticated Transfer π_{AT}



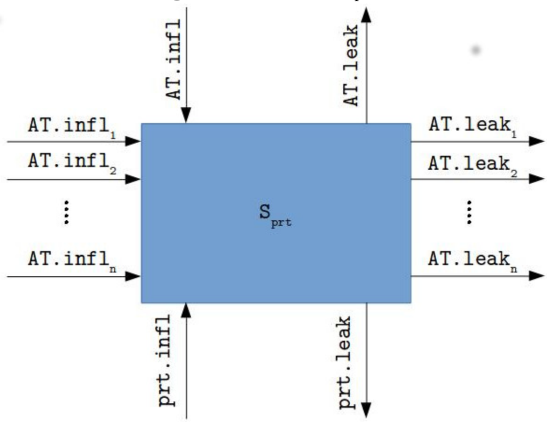
For each protocol π_{prt}, we construct an *ideal functionality* named F_{prt} (Fig.3). The ideal functionality is also an interactive agent. It has n input ports named $prt.in_i$ and n output ports named $prt.out_i$ where $i \in \{1, 2, \ldots, n\}$. $prt.in_i$ is for the input of the party P_i and $prt.out_i$ is for the output of the party P_i. The ideal functionality takes the input of each party and computes the desired function and delivers the output to the parties. The ideal functionality also has an input port $F.infl$ and output port $F.leak$. The output port $F.leak$ is to leak the views of all the corrupted parties. The input port $F.infl$ is to influence the actively corrupted parties.

Fig. 3. Ideal agent F_{prt}



We need to show that the protocol π_{prt} is as secure as the ideal functionality F_{prt}. To show this, we need to show the existence of a Simulator S_{prt} (Fig.4). S_{prt} is also modeled by an interactive agent. S_{prt} has input ports $prt.leak$, $AT.infl$ and $AT.infl_i$ where $i \in \{1, 2, \ldots, n\}$ and output ports $prt.infl$, $AT.leak$ and $AT.leak_i$ where $i \in \{1, 2, \ldots, n\}$.

Fig. 4. Simulator S_{prt}



If we compose the interactive agents F_{prt} and S_{prt} (i.e. if one interactive agent has an input port with same name as the output port of another interactive agent, then we join them together), we get an interactive system

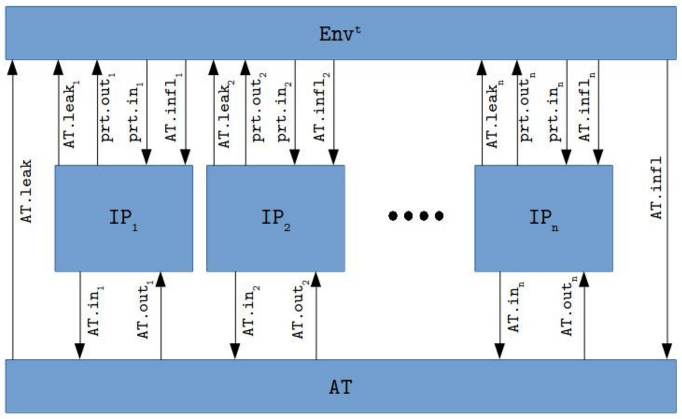
$F_{prt} \circ S_{prt}$. $F_{prt} \circ S_{prt}$ has same port structure as π_{prt}.

Now we define an agent for the adversary, Env^t, where t is the maximum number of parties that an adversary corrupts. The agent Env^t has input ports $AT.leak$, $prt.out_i$, $AT.leak_i$ and output ports $AT.infl$, $prt.in_i$, $AT.infl_i$ where $i \in \{1, 2, \ldots, n\}$. When we compose Env^t with $F_{prt} \circ S_{prt}$ or with π_{prt} (Fig.5 and 6), it becomes a closed interactive system, i.e. there are no open ports in this system.

To prove the security of the protocol π_{prt}, we play the following game with the adversary Env^t: Attach the adversary Env^t with either $F_{prt} \circ S_{prt}$ or π_{prt}. The adversary has to tell which interactive system it is attached to.

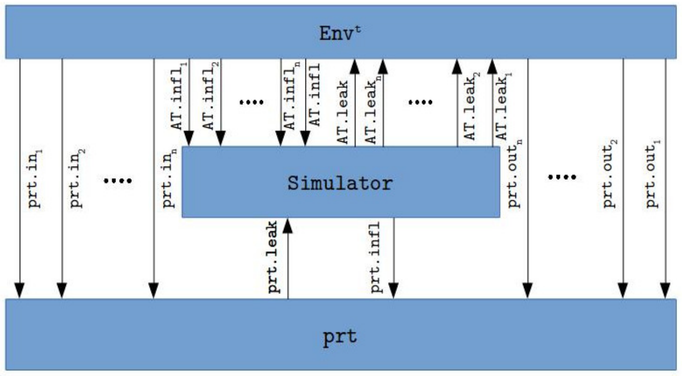
If we can build a simulator such that the adversary is not able to distinguish between the two interactive systems, then we can say that the protocol π_{prt} is as secure as the ideal functionality $F_{prt} \circ S_{prt}$. If both the systems π_{prt} and $F_{prt} \circ S_{prt}$ are behaving same with respect to adversary Env^t, then Simulator S_{prt} is essentially computing the actual leakage and influence from allowed leakage and influence. Therefore, the actual leakage and influence contains no more information than the allowed leakage and influence.

Fig. 5. Environment interacting with π_{prt} directly



- **Secret Sharing Protocol**
  A party P wants to share its secret with n parties (including P) such that each party has a share (some information) about the secret. The information contained in a share owned by a party is not enough to compute the secret. But, once all the parties (including P) combine their shares, then the secret can be computed.

Fig. 6. Environment interacting with $F_{prt} \circ S_{prt}$



Let $S \in \mathbb{Z}_q$ be the secret of party P which it wants to share. P chooses n random numbers $\{S_1, S_2, \ldots, S_n\}$ from $\mathbb{Z}_q$ such that:

$$S = \sum_{i=1}^{n} S_i \tag{1}$$

P gives each party exactly one share. This completes the protocol. Only when all the parties combine together they generate the secret S by summing up their shares.

– **Protocol for Secure Addition**
There are n parties, $\{P_1, P_2, \ldots, P_n\}$. Each party has a secret x_i. All the parties want to know the sum of their secrets, without revealing their secret to anyone. The required sum is:

$$Sum = \sum_{i=1}^{n} x_i \tag{2}$$

Following is the description of the protocol:

- Each party shares its secret x_i with all the parties using secret sharing protocol as mentioned above. The shares of secret x_i are denoted as $\{x_{i1}, x_{i2}, \ldots, x_{in}\}$. Therefore,

$$x_i = \sum_{j=1}^{n} x_{ij} \tag{3}$$

- Each party P_i has: $\{x_{1i}, x_{2i}, \ldots, x_{ni}\}$. P_i computes:

$$u_i = \sum_{j=1}^{n} x_{ji} \tag{4}$$

Each party makes the u_i public.

- All the parties now compute:

$$Sum' = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ji} = \sum_{j=1}^{n}\sum_{i=1}^{n} x_{ji} = \sum_{j=1}^{n} x_j = Sum \qquad (5)$$

3 The SMPGC Problem

The *secure multiparty graph computation* problem involves securely generating the graph of the underlying network of people such that no information apart from the structure is revealed. That is, the identity of the social actors is to be concealed, disallowing any sort of re-identification. We firstly give a formal definition of the problem followed by the protocol for solving it. We also show how the protocol is secure and no information is leaked.

3.1 Problem statement

Consider n parties $P_1, P_2 \dots P_n$ who are interested in generating the graph of their network. The constraint, however, is that the resulting graph must be unlabelled to prevent identification of the individual parties. Also, the graph generation process must be such that there is no risk of disclosing data that is private to a party. Thus, the problem of multiple parties securely computing their underlying graph is known as *secure multi-party graph computation.* Secure here means that no knowledge regarding the identity of nodes in the network is revealed to anyone (i.e. anonymized) and correctness of the network generated is guaranteed. These being the constraints, we cannot have any third party generating the graph.

3.2 Definitions

Throughout the execution of the protocol, the parties exchange data in the form of matrices. These matrices are generated in a specific way. It is therefore required that a few terms are defined that lay the groundwork to understand the protocol.

– *Column Replicated matrix* **(CRM)**: is an $n \times n$ matrix that is generated by replicating a random $n \times 1$ column vector n times. The column vector to be replicated is chosen uniformly at random from set of $n \times 1$ vectors in Z_q (where $q \in \mathbb{N}$ and Z_q is a group under addition) .

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 6 & 6 & 6 & 6 \\ 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 \end{pmatrix}$$

– *Mapped adjacency vector* (MA_i): The adjacency vector A_i of a party P_i is of the order $1 \times n$ and consists of 1's to indicate the presence of a tie and 0's otherwise. However, the parties generate this vector in accordance with an already agreed upon column labelling (i.e. which column corresponds to which party in the vector). In our protocol, column labelling is always such that column i corresponds to label P_i. We set a variable called *splitter* $(\mathbf{k})$ to a specific value in Z_q and map all the 0's in the adjacency vector to a random number greater than k, picked uniformly at random. Similarly we map all the 1's in the adjacency vector to a random number lesser than or equal to k picked uniformly at random. This modified adjacency vector is called a *mapped adjacency vector* (MA_i) of the party.

$$A_i = \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix} \xrightarrow{q=6,k=3} MA_i = \begin{pmatrix} 4 & 1 & 0 & 5 \end{pmatrix}$$

– *Mapped adjacency matrix* $(\mathbf{MA})$: The matrix obtained using mapped adjacency vectors of all the parties, where each row of the matrix is the MA_i corresponding to some party P_i, is called mapped adjacency matrix MA. Note that if the value of k is known we can easily reverse map the mapped adjacency matrix MA to adjacency matrix A.

$$MA = \begin{pmatrix} 1 & 2 & 0 & 3 \\ 4 & 4 & 1 & 1 \\ 1 & 5 & 4 & 0 \\ 2 & 1 & 4 & 5 \end{pmatrix} \xleftrightarrow{\text{q=6,k=3}} A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

3.3 SMPGC Protocol

In this section, we shall provide a protocol for generating the underlying graph on n parties $P_1, P_2, \ldots P_n$. The final output of the protocol will be the unlabelled adjacency matrix representing the underlying graph. The protocol involves the parties processing matrices of the order $n \times n$ which would finally morph into the desired adjacency matrix. All the matrices, vectors, as well as the operations performed on them are under modulo q in the following discussion.
The working of the SMPGC protocol assumes that certain functionality is possible. For instance, it should be possible for each party P_i to easily share his secret s such that the information of s is collectively held by parties and no single party apart from P_i has any information on s. It is also required that these n parties be capable of securely adding their secrets and revealing only the sum of all the secrets. Another function that is assumed possible is that each party can be securely assigned a unique number between 1 ton. The assignment is made unanimously, but no party is aware of the assignment of any other party. The first two functionalities are easily achievable through the protocol of secret sharing and secure addition respectively, as mentioned in Section 2 . The next functionality of unique integer assignment is discussed in detail in Section 4, along with its security and correctness are also shown. For now, however, we shall assume the working of this functionality as a black box for our protocol given below.

Step 1: Computing the *splitter* **k**

The choice of splitter is dependent on the density of the graph that is being constructed. This is easily computable once we know the sum of degree of all the nodes in the graph, i.e. the number of ties each party has in the network. Thus the parties can choose the secret as their degree in the network, given by the number of one's in their adjacency vector. This is shared using the Secret Sharing protocol. All the parties will add these secrets using the protocol for Secure Addition . This gives the total degree d of the network. The density of the graph D can be computed as:

$$D = \frac{d}{n^2} \tag{6}$$

Each party sets the value of k (the splitter used to map adjacency vector) to $D * q$, where q is the input space Z_q that was publicly chosen. This will result in the uniform distribution of the mapped adjacency vector under modulo q. However, the assumption made here is that all the adjacency vectors with given density are possible.

Step 2: **Hidden Permutation** generation

The next step is for the parties to unanimously assign oneself a distinct number between $[1, n]$ using the protocol for secure permutation. Let the number uniquely assigned to a party P_i be denoted by a_i. Thus, each of the a_i is unique and spans over the entire range $[1, n]$.

Step 3: Computing the **unlabelled mapped adjacency matrix** : This is the most important step of the protocol where we wish to compute the underlying graph represented in the adjacency matrix form. But, this step provides the graph in the form of mapped adjacency matrix which can later be mapped back to the adjacency matrix. Let the desired mapped adjacency matrix be MA_D.

– Now that the splitter **k** is known to all, each party P_i generates his mapped adjacency vector $\mathbf{MA}_i$ and also a column replicated matrix $\mathbf{CRM}_i$. The previous step provides a unique number a_i to each party. The generated $\mathbf{MA}_i$ is added to the a_i^{th} row of $\mathbf{CRM}_i$. This modified $\mathbf{CRM}_i$ would be each persons secret s_{i1}.

$$s_{i1} = \begin{pmatrix} \alpha_1 & \alpha_1 & \alpha_1 & \alpha_1 \\ \alpha_2 & \alpha_2 & \alpha_2 & \alpha_2 \\ \alpha_3 & \alpha_3 & \alpha_3 & \alpha_3 \\ \alpha_4 & \alpha_4 & \alpha_4 & \alpha_4 \end{pmatrix}_{CRM_i} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \ldots & \ldots & MA_i & \ldots \\ 0 & 0 & 0 & 0 \end{pmatrix} a_i$$

– All the parties add these secret matrices using Protocol for secure Addition allowing each of the parties to know the sum of all the $\mathbf{(CRM + MA)}'s$. Let this sum be denoted by the matrix M. It is clearly visible that the sum of all the CRM_i's is yet another column replicated matrix CRM'.The matrix M reveals nothing as the splitter in each row i of the underlying MA is masked by the random shift created by the addition with the value α_i which is the i^{th} row entry of the underlying matrix

CRM'. So, the reverse mapping of MA is not possible.

$$\begin{aligned} M &= \sum_{i=1}^{n} s_{i1} \\ &= \sum_{i=1}^{n} CRM_i + MA \\ &= CRM' + MA \end{aligned} \tag{7}$$

– Now we see that the row labels of M are hidden as each party knows only his row (a_i) in which resides his mapped adjacency vector MA_i masked by the corresponding row of the column replicated matrix CRM' . But, the column label is still known to all. Another point to be noted is that the row ordering of the obtained matrix M does not match the column ordering. Thus, the next attempt is to make the row ordering and the column ordering the same, without which no sensible data can be extracted. To accomplish this, each party P_i will generate its own random matrix R_i and copy the i^{th} column of M (i.e. i^{th} column corresponds to label P_i) and add this column to a_i^{th} column of R_i. Now, the column labelling of the matrix is hidden (as only P_i knows a_i) and is brought to its right place. Let Z_i be the matrix of each party whose all entries are zeroes except that its a_i^{th} column has entries of the i^{th} column of M. The matrix that will be each party's secret s_{i2} in next step is as given below.

$$s_{i2} = R_i + Z_i \tag{8}$$

– All the parties add these secret matrices s_{i2} using Protocol for secure Addition. Let the resulting sum matrix be M_1. Let M' be the resulting matrix obtained by reordering the matrix M such that row order respects the column ordering. Then we know M_1 can be given as follows:

$$\begin{aligned} M_1 &= \sum_{i=1}^{n} R_i + M' \\ &= R + M' \end{aligned} \tag{9}$$

$$M' = CRM' + MA_D \tag{10}$$

$$\therefore M_1 = R + CRM' + MA_D \tag{11}$$

– Now the parties perform secure addition to compute the sum of all their CRM_i's and $R_i's$. The sum of CRM_i and R_i be R_i' for each party which is his secret. Let all their sum be denoted by matrix M_2.

$$R_i' = CRM_i + R_i \tag{12}$$

$$
\begin{aligned}
M_2 &= \sum_{i=1}^{n} R'_i \\
&= \sum_{i=1}^{n} (CRM_i + R_i) \\
&= \sum_{i=1}^{n} CRM_i + \sum_{i=1}^{n} R_i \\
&= CRM' + R
\end{aligned}
\tag{13}
$$

– All the parties thus have access to both M_1 and M_2. Each party subtracts M_2 from M_1 and therefore obtains the unlabelled mapped adjacency matrix of the underlying network MA_D.

$$
\begin{aligned}
M_1 - M_2 &= (R + CRM' + MA_D) - (CRM' + R) \\
&= MA_D
\end{aligned}
\tag{14}
$$

Step 4: Reverse Mapping

At this step of the protocol, each party has the unlabelled adjacency matrix and also has knowledge of the splitter **k** value. With **k** known, each of them easily reverse map the values in the matrix. That is, replace all the values in the matrix that are greater than **k** by 0's and the rest by 1's. Thus, you obtain the required adjacency matrix, ensuring no labels are revealed.

3.4 Proof of Correctness

We have seen that all the matrices and vectors that are dealt with belong to Z_q, set of integers modulo q. Now, it is important to understand that the value q helps in making every mapped adjacency vector equiprobable from the sample space. This is so because, even if the occurrence of 0's and 1's in the adjacency vector are skewed, it is lost when they are mapped to values in the set $\{0, 1, 2, \ldots, q\}$ such that every value is equally likely. Thus, q value is dependent on the density of 1's in the adjacency matrix of the network, D. Based on the value of D, the range $[0, q]$ is split into two parts ranging from $[0, Dq]$ and $[(Dq + 1), q]$. All the 1's in the adjacency vector map to range $[0, Dq]$ in the mapped adjacency vector and similarly 0's to $[(Dq + 1), q]$.

Now we need to show that following the steps of the protocol actually leads to the correct output, i.e. it generates the desired matrix at the end. As mentioned earlier, this Section will deal only with the correctness of the third step of the protocol assuming the previous as a black box.

Firstly each party P_i computes his modified CRM which is used as the secret s_{i1} in secure addition. On the basis of the correctness of secure addition, we can be guaranteed that the sum matrix M computed is the sum of all the individual CRM_i's and the mapped adjacency matrix MA. Thus, the CRM' masks the matrix MA in the matrix M. Now to obtain the desired MA_D matrix, a reshuffling of the columns of matrix MA is required such that the column ordering matches the row ordering. It can be seen that this reshuffling has been distributed among all parties where each party brings his column to the right place. It is to be noted that while shuffling columns of matrix M, the underlying CRM' is undisturbed as all its columns are identical. But, this is done in conjunction with a random matrix to hide the data change that is happening. Once the MA is changed to MA_D, the underlying CRM' and all the random matrices are removed.

3.5 Proof of Security

From the SMPGC-Protocol, we can clearly see that, it is enough to show the security of the step of computing the unlabelled mapped adjacency matrix. Let this be implemented by an interactive system denoted as π_{UG}. The security of the *hidden permutation* is assumed true and it will be discussed in full length in the Section 4.4. Now, to establish security through behavioral equivalence, we model an ideal functionality F_{ug} whose internal structure is ignored while the functionality achieved is the same as that of π_{ug}.

Agent F_{UG}

- **initialize:** The ideal functionality keeps track of three sets, A (Actively Corrupted Parties), P (Passively Corrupted parties), C (Corrupted Parties). It also keeps bits $delivery-round$, $evaluated$, $inputs-ready$, $input-ready_1$, $input-ready_2, \dots, input-ready_n \in \{0,1\}$ initially set to 0.
- **Honest inputs:** On input ($clockin$, i) on $UG.infl$ for $i \notin A$, read a message from $UG.in_i$. If there was a message x_i on $UG.in_i$ and $input-ready_i = 0$, then set $input{-}ready_i \leftarrow 1$, store (i, x_i) and output $(input, i)$ on $UG.leak$. x_i is an adjacency vector showing the relationships of i^{th} party.
- **Corrupted inputs:** On input $(change, i, x_i)$ on $UG.infl$, where $i \in A$ and $evaluated$=0, set $input-ready_i \leftarrow 1$ and store (i, x_i), overriding any such previous value stored for party i (i.e. as long as the function has not been evaluated on the inputs, the corrupted parties are allowed to change their inputs).
- **Simultaneous inputs:** If it holds in some round that after the clock-in phase ends there exist $i, j \notin A$ such that $input-ready_i = 0$ and $input-ready_j = 1$, then do a complete break down. If it happens in some round that after the clock-in phase ends that $input-ready_i = 1$ for all $i \notin A$ and $inputs-ready = 0$, then set $inputs-ready \leftarrow 1$ and for each $i \in A$ where $input-ready_i = 0$, store $(i, x_i) = (i, 0)$.
- **Evaluate function:** On input ($evaluate$) on $UG.infl$ where $input-ready = 1$ and $evaluated = 0$, set $evaluated \leftarrow 1$, generate an adjacency matrix using the inputs $x_1, x_2, \dots x_n$ and then permute this

adjacency matrix. Call this permuted matrix y. Then output $\{(i, y)\}_{i \in C}$ on $UG.leak$, and if C later grows, then output (j, y) on $ug.leak$ for the new $j \in C$.

- **Simultaneous output:** On input ($delivery{-}round$) on $UG.infl$, where $evaluated$= 1 and $delivery - round$= 0, proceed as follows: if we are at a point where no party $i \in A$ was clocked out yet, then set $delivery - round \leftarrow 1$.
- **Delivery:** On input $(clockout, i)$ on $UG.infl$, where $delivery - round$ = 1, output y on $UG.out_i$.

Now we propose a simulator S_{ug} to prove that $\pi_{ug} \diamond \pi_{AT}$ is indistinguishable from $F_{ug} \diamond S_{ug}$. S_{ug} knows the input of each person. Using this data, calculation of splitter is easy. The values a_i for each party P_i is obtained using the protocol π_{prm} whose security will be proven. It chooses CRM randomly for all the parties as in the protocol and follows rest of the instructions for the protocol as it is. Now, the security of the protocol for secure addition is already proven. In Step 3, S chooses a random matrix R_i for each party P_i and follows rest of the instructions mentioned in the step as it is. Rest of the steps are followed by the simulator as mentioned by the protocol with no new random choices being made. The adversary against which we are going to show the security of this protocol is again Env^t where $t < n-1$ because the protocol for secure addition is being used as a sub-protocol. Even if the view of t $(< n-1)$ corrupted parties is fixed, still S_{ug} is able to make random choices for honest parties as they are independent of the choices made by corrupted parties. That is, in no way do the choices of corrupt parties limit the choices of the honest parties.

4 The SMPPC Problem

4.1 Problem Definition

There are n parties. They want to assign each one of themselves a number from 1 to n. It can be viewed as permuting the parties such that each party knows only its position in the permutation. A party should not know anything about the position of other parties in the permutation. The permutation generated by the protocol should be random. The protocol should be such that it does not favour any party. This problem is termed as *secure multiparty permutation computation* problem.

4.2 SMPPC Protocol

Step 1: All the parties (active participants) choose a random number between $1 - n$

Step 2: Now begins the rounds of questioning. In each round of questioning, the following question is asked- "Has anyone chosen the number i? ", where i runs from 1 *to* n. The answer to the question is obtained through voting implemented using the protocol for secure addition where each party shares a secret 1 to denote a positive vote (Yes) and 0 otherwise (No).

Step 3: What we obtain as answer in the questioning round i, is the number of people who happened to choose their random number as i. All the parties also maintain an ordered list of current free slots. Free slots list has the numbers that are yet to be assigned. Initially the list is empty.
- If the voting results in the answer zero, it indicates none have chosen the number i. The particular i is then queued in the list of free slots.
- If the voting results in the answer one, then there are no duplicates. Hence, it is assumed that the party is allotted the first number in the free slot list.Thus, the particular i is dequeued from the list. If the free slot list is currently empty, then the party is assumed to to be allotted the number i itself.
- If the voting results in an answer greater than one, then it indicates that there are duplicates as more than one party has chosen the same number. In this case, none of the parties are assigned any number in this round. The particular i is added to the free slot instead.

Step 4: Let the number of parties that chose unique integers in Step 1 be x. Then, after the last round of questioning - Has anyone chosen the number 'n'? - all the x parties would have been allotted a unique number in the range $1 - x$.

Step 5: The Steps 1 through 4 is repeated until all the parties are assigned a random number. Each cycle of Steps 1-4 is called a pass. In each pass, the following changes are made:
- Only those parties that had selected duplicate numbers in the previous pass and hence not allotted a number, actively participate. The parties who are assigned a number continue participating passively.
- The x parties whose number was fixed in any of the previous passes continue to participate passively where they only take part in voting. Their vote is 0 in each round of the passes.
- When repeating the Step 1, the active participants now choose a random integer from the range $[(x+1), n]$ instead of $[1, n]$
- In each pass, for questioning rounds only the integers from $[(x+1), n]$ will be considered.

4.3 Proof of Correctness

To prove the correctness of the protocol, it is enough to show that:

1. Each party is assigned a unique number once the execution of the protocol is finished.
2. The protocol does not favour any party, i.e. the probability of some party being assigned a number i is the same as the probability that any other party is assigned i.

Proof: In the protocol, a party is assigned some number $i \in \{1, 2, \ldots, n\}$ only when it picked a number $j \in \{1, 2, \ldots, n\}$ and no other party picked the number j. Once a party is assigned a number from $\{1, 2, \ldots, n\}$, its contribution towards the protocol is passive. It will not be assigned any number again. No other party will chose the number assigned to this party in the remaining rounds of the protocol, as the random selection of the numbers is allowed from the non-assigned numbers only. There are only n numbers, so each of the n parties will be assigned a unique number. However, a question may arise - what if the parties does not follow the protocol? This situation will be tackled in the following Section.

The final choice of the permutation by the protocol is random as it is based on the random choices of the parties. Each possible permutation should be equally likely to be assigned to the parties. To prove this, it is enough to show that a party P_i is equally likely to be assigned any number from $\{1, 2, \ldots, n\}$. In each pass of the protocol, P_i selects a number from $i \in \{1, 2, \ldots, n\}$, uniformly at random. Whether i will be assigned to this party depends upon the random selections made by other parties in the protocol. Assuming all parties of equal intelligence, we can see that there can exist no strategy for a party to ensure he is assigned a desired number i in the permutation. If there were to be such a strategy, then all the parties would employ the same, resulting in a collision of choices which in turn causes the failure of assignment. Another observation to be made is that the parties have no incentive to obtain a particular number assigned in the permutation. Everyone playing the protocol is guaranteed of an assignment and no assignment is more favourable than the other. This being the case, it is easy to see that the protocol does not favour any party during the assignment of the numbers.

4.4 Proof of Security

As we discussed in the preliminaries, we need to define a simulator S_{prm} for the permutation protocol π_{prm} to prove that π_{prm} is as secure as the ideal functionality F_{prm}. Following is the description of F_{prm}:

Agent F_{prm}

- **initialize:** The ideal functionality keeps track of three sets, A (Actively Corrupted Parties), P (Passively Corrupted parties), C (Corrupted Parties). It also keeps bits $delivery - round$, $permuted$, $ready$, $ready_1$, $ready_2$,..., $ready_n \in \{0, 1\}$,
  initially set to 0.
- **Honest inputs:** On input ($clockin$, i) on $prm.infl$ for $i \notin A$, read a message from $prm.in_i$. If there was a message $ready$ on $prm.in_i$ and $ready_i = 0$, then set $ready_i \leftarrow 1$, store $(i, ready)$ and output $(input, i)$ on $prm.leak$.

- **Corrupted inputs:** On input $(change, i, ready)$ on $prm.infl$, where $i \in A$ and $permuted$=0, set $ready_i \leftarrow 1$ and store $(i, ready)$, overriding any such previous value stored for party i.
- **Simultaneous inputs:** If it holds, in some round, that after the clock-in phase ends there exists $i, j \notin A$ such that $ready_i = 0$ and $ready_j = 1$, then do a complete break down. If it happens in some round that after the clock-in phase ends, $ready_i = 1$ for all $i \notin A$ and $ready = 0$, then set $ready \leftarrow 1$ and for each $i \in A$ where $ready_i = 0$, store $(i, ready)$.
- **Permutation:** On input $(permute)$ on $prm.infl$ where $ready = 1$ and $permuted = 0$, set $permuted \leftarrow 1$, select a permutation of integers in [1,n] uniformly at random. Let this permutation be $\{a_1, a_2, a_3, \ldots, a_n\}$ where each $a_i \in \{1, 2, \ldots, n\}$. Then output $\{(i, a_i)\}_{i \in C}$ on $prm.leak$, and if C later grows, then output (j, a_i) on $prm.leak$ for the new $j \in C$.
- **Simultaneous output:** On input $(delivery - round)$ on $prm.infl$, where $permuted$= 1 and $delivery - round$= 0, proceed as follows: if we are at a point where no party $i \in A$ was clocked out yet, then set $delivery - round \leftarrow 1$.
- **Delivery:** On input $(clockout, i)$ on $prm.infl$, where $delivery - round$ = 1, output a_i on $prm.out_i$.

Let str be the string that is obtained by concatenating views of all the parties from P_1 to P_n. If we prove that str for π_{prm} and $F_{prm} \circ S_{prm}$ are statistically indistinguishable from each other then Env^t can't distinguish one system from another. Following is the description of the Simulator.

We know that the protocol π_{add} is secure under dynamic and active corruption with threshold $t = n - 1$. Using the Universally Composable (UC) theorem mentioned in [6], we don't need to separately define the simulator's actions for π_{add} used in π_{prm} as a sub protocol. We can use the simulator S_{add}'s actions when π_{add} is being executed in the π_{prm}. We'll define rest of the S_{prm}'s actions. There are no inputs that parties give to the protocol, but random choices. At the start of the protocol Env^t sends the set of parties that it is going to corrupt to the port $AT.infl$.

We will first consider passive and static security. Let C be the set of passively corrupted parties. Note that, C can't be of size more than $n - 2$ otherwise the security of π_{add} will break. The simulator S_{prm} should be such that π_{prm} and $F_{prm} \circ S_{prm}$ behave similarly. We will run a copy of π_{prm} inside the S_{prm} under same adversary (or environment) Env^t, while describing the actions of simulator in order to make the decisions taken by simulator clear. In the first step of the protocol, each party picks a random number from the set $\{1, 2, \ldots, n\}$. The number generated by party P_i will be leaked to simulator through $prm.leak_i$

where $P_i \in C$. For the set of honest parties it randomly generates a number from $\{1, 2, \ldots, n\}$. The honest parties were doing exactly the same in π_{prm}. Views of corrupted parties are available to S_{prm}, the string *str* consisting of views of all parties until now, are indistinguishable for both the systems. In the second step of protocol, all the parties are performing addition using π_{add}, S_{prm} will mimic the behavior of S_{add} in this step. The contents of *free slot list* is known to all the parties, therefore it is known to the simulator. In the Step 4, Step 1 is repeated just with a different range to pick the numbers from. Therefore, the actions of simulator in Step 1 can be extended to actions required in Step 3. Simulator will always add 0 to the view of an party which has already been assigned a number. For the rest of the parties, simulator behaves in same way it did in Step 1. The Step 5 is just a repetition of Step 1 to Step 4 and we have already dealt with those cases. **Therefore, the protocol π_{prm} is secure under passive and static corruption under the threshold $t = n - 2$.**

Take the case for active and static security. As mentioned earlier, input substitution is not considered as deviation or active corruption of a party. In the protocol π_{prm}, we shall see what constitute active corruption of a party. Let C be the set of actively corrupted parties. In the Step 1 of the protocol, an adversary can only change the number that the corrupted party was going to select. This deviation is same as the input substitution. We assume the party to be honest whose input is the number selected by the party. In Step 2 and 3, the deviation from protocol can be of the following types:

1. A party has chosen number i in Step 1, but he votes 0 for the question: *Has anyone chosen the number i?* and votes 1 for the question: *Has anyone chosen the number j?* where $j \neq i$.
2. A party votes $x \geq 2$ for any question or it votes 1 for multiple questions (multiple votes).
3. A party votes a negative number to counter the effect of its vote $x \geq 2$ in a different question.

The deviation of type 1 is input substitution only i.e. if a party is voting 1 for the question: *Has anyone chosen the number j?* but it had chosen i in the previous step then we assume that it is an honest party which would have chosen j in Step 1.

The deviation of type 2 can be easily detected by summing up the answer to each question at the end of each pass. If the sum is more than n then there are actively corrupted parties present in the protocol and protocol will halt.

The deviation of type 3 cannot always be detected. There might be the cases when the sum of result of all the questions is equal to n, but there are actively corrupt parties present in the protocol whose influence is more than the allowed influence. For example, assume only one party, say P_i, is actively corrupted. If P_i does not deviate from the protocol, assume the result of voting for the question:

"*Has anyone chosen the number j?*" is 1 because only P_i has chosen j. Also assume the result of the voting for the question: "*Has anyone chosen the number k*" is 2, because some two parties have chosen the number k. The party P_i can vote 3 (instead of 1) for the question: "*Has anyone chosen the number j?*" and it votes -2 (i.e. $m-2$, if we are doing the addition under modulo m, $m > n$) for the question: "*Has anyone chosen the number k*". In the case mentioned above the active deviation of P_i can be detected by anyone one of the parties which voted 1 for the number k, because they know that result should be at least 1 but it is 0 in the case above. But if P_i votes -1 (m-1) for the number k and -1 for some other number k', assuming that at least 2 parties voted 1 for the number k' and k both. In this case, the party P_i's deviation will not get caught and can result in disrupting the output of the protocol. Assume exactly 2 parties voted 1 for the number k, the result of the question: "*Has anyone chosen the number k*" is going to be 1 because P_i voted -1 for k. Both the parties will now assume that there is no collision (when in fact there is a collision) and they will take the first number from free slot list or they both will take number k if free slot list is empty. The result is, two parties are assigned the same number, which is undesirable. Also, for the number j there was no collision, but the deviation resulted in the collision at number j. But, this does not mean that the adversary can make an actively corrupted party deviate from protocol in this way without worrying about getting caught. When an adversary is actively corrupting a party with deviation type 3, it runs the risk of getting caught.

The set of actively corrupted parties can cause another type of problem to the protocol called starvation. The adversary can deliberately choose the same number for more than one actively corrupted parties in each pass. This will prevent the protocol from reaching its completion. Thus the adversary is starving the protocol. In the Section 4.5, we show that the expected number of passes required for the completion of protocol is $O(log(n))$. If the number of passes completed has become $O(n^2)$ then the parties can suspect that there is an active adversary starving the protocol, because otherwise the probability of protocol reaching $O(n^2)$ passes is very less.

The protocol π_{prm} is not completely secure under the active corruption.

Take the case for passive and dynamic security. There can be different set of passively corrupted parties C for each step of the π_{prm}. The set of corrupted parties cannot be changed during a round. If in the same round, the adversary changes the set of corrupted parties from C_1 to C_2 then it is equivalent to the adversary corrupting the $C_1 \cup C_2$ in that round. The argument can be extended to any number of times the adversary changed the set of corrupted parties during a round. Without losing the generality, we can assume that the adversary does not change the set of corrupted parties in a round. In the Step 1, the adversary can corrupt a set C, of size $n-2$, parties and get to know their views, i.e. which number did they choose. In the next step, i.e. the rounds of questioning, the adversary corrupts only one party P_i which was not corrupted in Step 1. After these rounds are completed, the adversary will get to know the number that the party P_i chose in the Step 1, based on the information: to which question did P_i vote 1. The adversary knows what all the parties in the set C are going to vote for each question and it also knows what P_i is going to vote in each questioning round. Let the result for the question: "*Has anyone chosen the number j?*", be x according to the information known to the adversary, but the actual result is $x+1$. The extra vote is coming from the party which is honest in all the rounds. Therefore, adversary is able to determine the input of the honest party without crossing the security threshold $t = n - 2$ in any round (or step).

Therefore, the protocol π_{prm} is not secure under dynamic corruption.

4.5 Expected number of passes required by SMPPC

In the protocol SMPPC, the parties whose choices collided in a pass were not assigned any number during that pass. They are required to choose from unassigned numbers in the following pass. The protocol goes on until after some pass each party has been assigned a number. The question arises that: *How many passes are required so that each party is assigned a number?*
Let X_i be the indicator random variable,

$$X_i = \begin{cases} 1 & exactly\ one\ person\ chose\ number\ i \\ 0 & otherwise \end{cases} \tag{15}$$

Let X be the random variable:

$$X = \sum_{i=1}^{n} X_i \tag{16}$$

Given n parties at the start of the round which are not assigned any number, $E(X)$ denote the expected number of parties which will be assigned a number in this pass.

$$E(X) = E(\sum_{i=1}^{n} X_i) = n \sum E(X_i) \tag{17}$$

We can take the sum out of the expectation because each party's choices are independent of other parties. The expectation of X_i is:

$$E(X_i) = \binom{n}{1} * \frac{1}{n} * (1 - \frac{1}{n})^{n-1} \tag{18}$$

$$E(X_i) \to \frac{1}{e} \qquad\qquad \ldots as\ n \to \infty \tag{19}$$

$$E(X) = n * E(X_i) = \frac{n}{e} \tag{20}$$

In each pass, $(\frac{n}{e})$ is the expected number of parties that are assigned a number. In the next pass, $(n - \frac{n}{e})$ parties will actively participate.

$$\begin{cases} n * (1 - \frac{1}{e}) : & The\ number\ of\ parties\ not\ assigned\ a\ number\ after\ pass\ 1 \\ n * (1 - \frac{1}{e})^2 : & The\ number\ of\ parties\ not\ assigned\ a\ number\ after\ pass\ 2 \\ n * (1 - \frac{1}{e})^3 : & The\ number\ of\ parties\ not\ assigned\ a\ number\ after\ pass\ 3 \\ \dots \\ \dots \\ \dots \end{cases}$$

After each pass, a constant fraction of parties are being assigned a number. Therefore, after $O(log(n))$ number of passes, all the parties will be assigned a number. **The expected number of passes required by our protocol is** $O(log(n))$.

References

1. Backstrom, L., Dwork, C., Kleinberg, J.: Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In: Proceedings of the 16th international conference on World Wide Web. pp. 181–190. ACM (2007)
2. Backstrom, L., Huttenlocher, D., Kleinberg, J., Lan, X.: Group formation in large social networks: membership, growth, and evolution. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 44–54. ACM (2006)
3. Cha, M., Haddadi, H., Benevenuto, F., Gummadi, P.K.: Measuring user influence in twitter: The million follower fallacy. ICWSM 10(10-17), 30 (2010)
4. Cha, M., Mislove, A., Gummadi, K.P.: A measurement-driven analysis of information propagation in the flickr social network. In: Proceedings of the 18th International Conference on World Wide Web. pp. 721–730. WWW '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1526709.1526806
5. Chester, S., Kapron, B.M., Srivastava, G., Venkatesh, S.: Complexity of social network anonymization. Social Network Analysis and Mining 3(2), 151–166 (2013)
6. Cramer, R., Damgard, I., Nielsen, J.B.: Secure multiparty computation and secret sharing-an information theoretic approach. Book Draft (2012)
7. Gnanasekar, V., Jayanthi, S.: Privacy preservation of social network data against structural attack using k-auto restructure. International Journal of Computer Science & Information Technologies 5(2) (2014)
8. Hay, M., Miklau, G., Jensen, D., Towsley, D., Weis, P.: Resisting structural reidentification in anonymized social networks. Proceedings of the VLDB Endowment 1(1), 102–114 (2008)
9. Hay, M., Miklau, G., Jensen, D., Weis, P., Srivastava, S.: Anonymizing social networks. Computer Science Department Faculty Publication Series p. 180 (2007)
10. Kapucu, N., Yuldashev, F., Demiroz, F., Arslan, T.: Social network analysis (sna) applications in evaluating mpa classes. Journal of Public Affairs Education pp. 541–563 (2010)
11. Li, N., Gillet, D.: Identifying influential scholars in academic social media platforms. In: Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. pp. 608–614. ACM (2013)

12. Liu, K., Terzi, E.: Towards identity anonymization on graphs. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. pp. 93–106. ACM (2008)
13. Potterat, J., Phillips-Plummer, L., Muth, S., Rothenberg, R., Woodhouse, D., Maldonado-Long, T., Zimmerman, H., Muth, J.: Risk network structure in the early epidemic phase of hiv transmission in colorado springs. Sexually transmitted infections 78(suppl 1), i159–i163 (2002)
14. Salathe, M., Kazandjieva, M., Lee, J.W., Levis, P., Feldman, M.W., Jones, J.H.: A high-resolution human contact network for infectious disease transmission. Proceedings of the National Academy of Sciences 107(51), 22020–22025 (2010)
15. Singh, L., Zhan, J.: Measuring topological anonymity in social networks. In: Granular Computing, 2007. GRC 2007. IEEE International Conference on. pp. 770–770. IEEE (2007)
16. Zhou, B., Pei, J.: Preserving privacy in social networks against neighborhood attacks. In: Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. pp. 506–515. IEEE (2008)