Privacy Preserving Network Analysis of Distributed Social Networks

Varsha Bhat Kukkala, Jaspal Singh Saini, and S.R.S. Iyengar

Indian Institute of Technology Ropar, Punjab, India - 140001
varsha.bhat@iitrpr.ac.in, jaspal.singh@iitrpr.ac.in,
sudarshan@iitrpr.ac.in

Abstract. Social network analysis as a technique has been applied to a diverse set of fields, including, organizational behavior, sociology, economics and biology. However, for sensitive networks such as hate networks, trust networks and sexual networks, these techniques have been sparsely used. This is majorly attributed to the unavailability of network data. Anonymization is the most commonly used technique for performing privacy preserving network analysis. The process involves the presence of a trusted third party, who is aware of the complete network, and releases a sanitized version of it. In this paper, we propose an alternative, in which, the desired analysis can be performed by the parties who distributedly hold the network, such that : (a) no central third party is required; (b) the topology of the underlying network is kept hidden. We design multiparty protocols for securely performing few of the commonly studied social network analysis algorithms. The current paper addresses a secure implementation of the most commonly used network analysis measures, which include degree distribution, closeness centrality, PageRank algorithm and K-shell decomposition algorithm. The designed protocols are proven to be secure in the presence of an arithmetic black-box extended with operations like comparison and modulo.

Keywords: social network analysis, secure multiparty computation, centrality measures

1 Introduction

Understanding the structure that emerges due to the interaction between various social entities has intrigued many scientists. This emergent structure, known as a social network, is observed to exhibit certain common characteristics, such as scale free degree distribution [1], high clustering coefficient [2], community structure [3], core-periphery structure [4] and several others. The study investigating the topological characteristics of networks is termed as *social network analysis* (SNA). A study of these properties has helped infer the functional underpinnings of several complex networks [5–7].

The behavior of individuals observed locally, when clubbed with the global network structure, reveals the emergent collective behavior. Hence, it is the network topology that allows for inferences to be made on several functional aspects

of the system. As a result, a multitude of tools, techniques and algorithms are available, which take the network structure as their input and provide a range of inferences about the network. It is therefore important that we have the network structure prior to applying the techniques of SNA. However, there are several networks of interest which are not easily accessible. Firstly, the network may be distributedly held, such that each person has a partial or local view of the global network. This local view can hold sensitive information of the individual, preventing her from disclosing it. An example would be that of a *hate network* on a set of individuals. Nodes in such a network would comprise of individuals, while an edge between two nodes would express the feeling of hatred between the corresponding individuals. A hate network is the ignored counterpart of the well studied friendship network. The underlying network is considered to be directed, such that, a directed edge from node A to node B, denoted as (A, B), would express person A's hatred towards person B. In such a network, it is clear that no individual would have knowledge of the global network. Each individual is only aware of the directed edges that she has to other individuals in the network (local view), leading to the network structure being distributedly held by several individuals. Secondly, even if an individual/organization has the global network, legal policies might not allow her to reveal the network. For example, human contact network collected through a database of hospital records cannot be disclosed publicly [8]. The current technique employed in collecting distributedly held sensitive data include surveys and interviews [9] conducted by a trusted third party. Once the network data is obtained, it is sanitized to ensure re-identification of the individuals is not possible. It is then released for data analysis [10, 11]. This process, known as *anonymization*, is believed to preserve privacy and at the same time, allows for an inferential study to be made on the network.

Even though anonymization is a widely opted for technique to perform SNA on private networks, there are several studies that point to its shortcomings [12–15]. Firstly, there have been numerous instances where sanitized data released in public has been de-anonymized [16, 13]. The released network structure data is combined with some auxiliary information to re-identify nodes in the original network. Secondly, during anonymization, graph structure is slightly perturbed in order to resist re-identification attacks [17, 18]. This perturbation leads to restructuring of the original network, which might not be desirable in certain cases. What would be ideally desired is to be able to perform the required analysis while the network is kept hidden, since it is the results of the analysis that we are interested in rather than learning the network structure. For example, if it is just the average shortest path length of a hate network that we are interested in computing, then releasing the network structure would provide access to more than the required information. In this paper, we explore the possibility of performing SNA on a network without having to reveal its topology.

The problem of securely computing a network parameter while keeping the network structure hidden, can be seen as an instance of multiparty computation (MPC). It involves designing protocols that allow a bunch of individuals to compute a function, a network parameter in our case, while hiding each of their private input, which would be the distributedly held network structure. Hogg and Adamic [19] have discussed on how MPC can be used to address the privacy concerns of the individuals on whom the social network is knit. The idea of MPC was first introduced by Yao [20], where he proposed a method for two millionaires to determine who amongst them is richer, without revealing each other's wealth. The idea was generalized to a multiparty setting as follows: designing a protocol to securely determine a globally known function $y = F(x_1, x_2, \ldots, x_n)$, where x_i is the private input supplied by party P_i. Security here means that the individuals participating in the protocol, referred to as parties, do not learn any additional information, apart from what is computable using just their respective input and output. MPC has been extensively applied in a myriad of problems including computational geometry, voting, bench-marking, etc. In this paper, we provide secure implementation of protocols that compute a network parameter without releasing the network structure.

1.1 Our Contribution

Determining a network parameter p is modeled as securely computing a function $F(x_1, x_2, \ldots, x_n) = p$, where x_i is the private input provided by party P_i. The private input of a party P_i is a graph $G_i(V_i, E_i)$, such that $G(V, E) = \bigcup_{i=1}^{n} G_i = (\bigcup_{i=1}^{n} V_i \ , \ \bigcup_{i=1}^{n} E_i \ )$. The graph G captures the global network, whose parameter we are interested in computing.

The protocols in the paper are designed considering two different scenarios, based on the type of input provided by the parties. The parties could be individuals who are themselves a part of the network. When a node in the network corresponds to a party in the protocol, we term such parties as *internal agents*. For example, consider the trust network on all employees of an organization, where an edge (A,B) represents that individual A trusts individual B. Each employee participates as an internal agent in the protocol and reports the trust she has with other employees as her private input. Alternatively, we could have a set of parties who collectively hold the network on a globally known set of individuals. In this case, each party has a sub-graph on the set of individuals. For example, consider the financial transaction network on a set of individuals, where we assume that an individual is allowed to have accounts in multiple banks. An edge (A,B) represents that there has been a minimum m number of transactions between node A and node B, which could be within the same bank or across different banks. Here, the parties who collectively hold the network information are the banks that the individuals have accounts in. Such parties are termed as *external agents*. The protocols designed in the paper support modeling parties as either internal or external agents.

The paper provides implementation details for securely computing degree distribution, closeness centrality using Dijkstra's algorithm, PageRank or eigenvector

centrality using the random surfer model, and K-shell decomposition algorithm. We also provide details of several sub-protocols that have been commonly used while computing the above network parameter protocols. These have been described in Appendix A as discretionary sub-protocols .
The designed protocols are proven to be secure in the presence of an extended arithmetic black-box $\mathcal{F}_{ABB}$, which is defined in Section 4. All the proposed network parameter protocols except PageRank protocol are perfectly secure, while PageRank protocol is statistically secure, given a perfectly secure $\mathcal{F}_{ABB}$ functionality. We calculate the cost involved in each of the proposed protocols, in terms of the number of operations invoked of the $\mathcal{F}_{ABB}$ functionality, which include addition, multiplication, equality/comparison and private modulo reduction operations. The exact communication cost, computation complexity and the number of rounds involved would depend on the implementation of the assumed arithmetic functionality.

2 Related work

Many recent works have focused on securely implementing graph algorithms. Brickell and Shamatikov [21] looked at the problem of securely computing single source shortest path and all pair shortest path. The proposed protocol is restricted to a two party setting, where the solution is computed on the union of the graphs held by the two parties. Moreover, the adversarial model assumed is that of the semi-honest model. Aly et al. [22] propose a set of protocols for securely computing shortest path problems in the multiparty setting. The authors propose a secure implementation of Bellman-Ford and Dijkstra's algorithm for the shortest path problem. Additionally, they consider the maximum flow problem and propose secure implementations of the Edmonds-Karp and Push-Relabel algorithms. In a successive work, Aly and Vyve [23] consider the problem of securely finding the cycle with the least averaged cost, better known as the minimum mean cycle problem. They also address the minimum cost flow problem in the information theoretic model, which involves minimizing the cost involved in sending a flow from a source node to a destination node over a graph whose edges have a capacity metric. It also provides an efficient implementation of Dijkstra's algorithm, which we use as a building block in this paper. There are works that propose data oblivious algorithms, such as the one proposed by Blanton et al. [24], which hide the input dependent memory access pattern. Data oblivious algorithms for breadth-first search, single-source single-destination shortest path, minimum spanning tree and maximum flow problems are proposed by Blanton et al. [24].

Apart from the works that have focused on graph algorithms, the theme of secure SNA has also been briefly studied. Keith Frikken and Philippe Golle describe a cryptographically secure method to compute the underlying anonymized network [25]. Here, the entire network is distributedly held by a set of parties, such that each row in the adjacency matrix corresponds to the input of a party.

The work assumes existence of additional authorities, who take responsibility of collecting data securely, using which they reconstruct the graph. Another work that proposes the construction of the graph is given by Bhat et al. [26], which is limited to the semi-honest model. The latest work in this direction is done by Tassa and Cohen [11], where they provide a secure implementation of an algorithm for network anonymization through clustering. The focus in all of the above works is to determine the structure of the underlying graph while preserving privacy. The motive of the current work is to make the required inferences without revealing the network structure.

Kerschbaum and Schaad [27] propose a technique for computing the closeness and betweenness of nodes in a distributedly held network. Their protocol assumes a threshold homomorphic encryption scheme, like that of Paillier's cryptosystem. Betweenness computation, in the specific case of supply chain networks, has been looked at by Fridgen and Garizy [28]. This is modeled for the semi-honest parties and has less stringent a definition of privacy, as the leaked neighborhood information is allowed for in the definition. It also relies on encryption schemes to exchange data and is secure in the cryptographic model. Tassa and Bonchi [29] propose a secure technique for determining the influence of a person in a social network, by combining the network data with the activity logs of the individuals in the network. They provide a multiparty protocol for the above, which is limited to the semi-honest adversarial model. Even though the aim of the paper is the same as ours, of finding influential nodes, their's relies on the availability of activity logs of individuals. Our approach of finding important nodes in the network is based entirely on the network structure and relies on the standard techniques of measuring centrality ranking of nodes. The algorithm used to find central nodes are application specific and hence we discuss a few of the important ones.

3 Preliminaries

In this section, we present some basic terminologies used throughout the paper. Section 3.1 discusses on some basic graph theoretic notation used in the paper and Section 3.2 provides a brief introduction to the field of multiparty computation.

3.1 Graph theory

A graph/network $G(V, E)$ is an ordered pair, where V represents the set of nodes/vertices and E represents the set of edges, which is a relation on V. Attributing to the directional nature of the edges, a graph is sometimes also termed as a directed graph or a directed network. A graph is said to be undirected if $(u, v) \in E \iff (v, u) \in E$ for every $u, v \in V$. If a real value (called weight) is attached with each edge, then the graph is termed as a weighted graph.

A graph $G(V, E)$ can be represented using a square adjacency matrix $A = (a_{ij})_{n \times n}$, where a_{ij} is the i^{th} row j^{th} column entry of the matrix, representing

the edge between node $i, j \in V$. In case of an unweighted graph, the value of $a_{ij} = 1$ if $(i, j) \in E$, otherwise $a_{ij} = 0$. On similar lines, we can define a symmetric/undirected adjacency matrix and a weighted adjacency matrix. The i^{th} row of the adjacency matrix A is termed as the i^{th} adjacency vector, which is represented using the notation v_i, hence $A = [v_i]_{n \times 1}$.

3.2 Multiparty computation

The following section formalizes the basic definitions and notions that are imperative to designing any secure multiparty protocol and aid in providing the rigorous proof for the same. The notations used here are borrowed from the comprehensive work [30] written by Cramer et al. The section consists of only those definitions and terminologies that are essential and we suggest [30] for further reading.

An MPC protocol consists of parties $P_1, P_2, \ldots, P_n$, where n is the number of parties. Each party P_i possess a private input x_i, and the parties are interested in securely computing some globally known function $F(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_n)$, where y_i denotes the output of the protocol sent to party P_i.

Modeling behavior of parties and adversary
The behavior of parties during the execution of the protocol could be broadly classified as honest, semi-honest, or malicious. A party is termed *honest* if she sincerely follows the protocol and does not collude with any other party. A party that is not honest is said to be *corrupt*. Corruption is further modeled as semi-honest or malicious, as discussed below.

A corrupt party is said to be *semi-honest* if she does not deviate from the specified protocol. However, the party may collaborate with others and try accessing private information, which would violate privacy of the remaining honest parties. Such parties are also known as *honest-but-curious* because they are honest in not deviating from the protocol but are curious to learn information about the honest parties. A party is said to be *malicious* when no assumptions are made regarding her behavior. Such a party is given the liberty to behave arbitrarily during the protocol execution. These parties are in a sense stronger than their counterparts in the semi-honest model and hence it is a tougher challenge to prove security in the malicious adversarial model. In order to model the corruption of the parties, we assume the existence of a central attacker called the *adversary*, who has control over all the corrupt parties. The extent of control depends on the assumption of the adversarial model. The protocols designed in this paper derive its security properties from the security of the extended arithmetic functionality $\mathcal{F}_{ABB}$ defined in Section 4.

Security in MPC protocol
The aim of a multiparty protocol is to allow the distributed computation of a desired function while guaranteeing the security of the protocol. To deem a protocol *secure*, it is required that a few conditions be met, which include:

- Correctness: The output generated by the protocol must be the desired value of the function F on the inputs $x_1, x_2, \ldots, x_n$.
- Privacy: The protocol must be designed such that a party learns nothing but the intended result. It should be noted that, any information that the party can gather by just using its input and output does not amount to breach in privacy. In an ideal world, the parties have access to their private input and the designated output. Any information that the party can deduce from the input and output alone is termed as *allowed leakage*. The information that the party gathers during the run of the protocol i.e. her *view* is termed as the *actual leakage*. For privacy to hold, the actual leakage of the set of corrupt parties must be within their allowed leakage.
- Robustness: Just as privacy is concerned with knowledge gained, robustness focuses on the influence gained by a corrupt party. The influence that an adversary gains during the execution of the protocol is called *actual influence*, while that possible in an ideal world is called *allowed influence*. Allowed influence would include input substitution i.e. a party P_i can input x_i' to the protocol rather than her true input x_i, whereas actual influence could represent any possible deviation from the described protocol. A protocol is said to be *robust* if the effect of an actual influence is obtainable from allowed influence.

A protocol is said to be private if the information in the view of the party is no more than the input and output of the party. Therefore, a protocol is private if there exists an algorithm S (also known as a simulator) which inputs the allowed leakage of the corrupt parties ($\{x_i, y_i\}_{i \in C}$) and outputs the views of the set of corrupt parties denoted by C. The values generated by the simulator are called the simulated values. The proof of robustness is given on similar lines as well. A protocol π is said to be robust if there exists a simulator S which inputs the actual influence of corrupt parties, and it outputs an equivalent allowed influence of the corrupt parties. Generally we assume that input substitution is the only allowed influence unless otherwise stated.

Security models

The security of a protocol is categorized based on the behavior of the adversary and on the computational power of the adversary. If a protocol π is proven to be secure in the semi-honest adversarial model, then it is said to be *passively secure*. If π assumes the malicious adversarial model, it is said to be *actively secure*. Next, we differentiate the security of a protocol based on the computational capabilities of the adversary. If a protocol π is proven to be secure under the assumption that the adversary is computationally bounded with attacks that are feasible in polynomial time complexity only, we say that the protocol is *computationally secure*. If π makes no assumptions about the computational capabilities of the adversary, then the protocol is said to be *perfectly secure* if the simulated values are perfectly indistinguishable from the actual leakage, while it is said to be *statistically secure* if the simulated values are statistically indistinguishable

from the actual leakage.

Universal composability theorem
Before stating the Universal Composability (UC) theorem, we define an ideal functionality. The exact formulation of an ideal functionality is given in terms of interactive systems, which can be found in [30]. Intuitively, an *ideal functionality* $\mathcal{F}$ is a secure protocol with the required input/output behavior. The actual leakage of $\mathcal{F}$ is precisely the allowed leakage, and the actual influence of $\mathcal{F}$ is precisely the allowed influence. Hence, when designing protocols for a required functionality, our aim is always to design a protocol as secure as its corresponding ideal functionality. Generally, if X is the function that we desire to compute, then $\mathcal{F}_X$ would represent the ideal functionality for computing X and π_X would denote the protocol designed to be as secure as $\mathcal{F}_X$.

Let the protocol π_G represent the implementation of an ideal functionality $\mathcal{F}_G$, such that π_G is as secure as $\mathcal{F}_G$. Let $\mathcal{F}_H$ be another ideal functionality corresponding to the protocol $\pi_H \diamond \mathcal{F}_G$, such that $\pi_H \diamond \mathcal{F}_G$ is as secure as $\mathcal{F}_H$. Here, $\diamond$ represents the composition operation, which implies that the protocol π_H invokes the ideal functionality $\mathcal{F}_G$ as a sub-protocol. Then, UC theorem states that $\pi_H \diamond \pi_G$ is as secure as $\mathcal{F}_H$. UC theorem allows us to replace ideal functionalities in a complex protocol, involving the composition of several sub-protocols, by their respective secure implementations. This eases the process of proving the security of a protocol composed with various other secure sub-protocols.

4 Building Blocks

The proposed network parameter protocols are designed using the arithmetic black-box $\mathcal{F}_{ABB}$. The ideal functionality $\mathcal{F}_{ABB}$ allows the n parties to perform the following operations over a field $\mathbb{F}_p$:

1. Store: A party P can store an element $a \in \mathbb{F}_p$ in the arithmetic black-box. We depict this operation using the following notation:
   $[b] \leftarrow_P a$, the element a is now securely stored in the memory location named b in the arithmetic black box $\mathcal{F}_{ABB}$.
2. Addition: If $[a]$ and $[b]$ are stored in $\mathcal{F}_{ABB}$, then the parties can store $[a]+_p[b]$ in $\mathcal{F}_{ABB}$, where $+_p$ represents addition modulo p. We will denote an addition operation as follows:
   $[c] \leftarrow [a] + [b]$, where c contains the sum $([a] +_p [b])$
3. Multiplication: If $[a]$ and $[b]$ are stored in $\mathcal{F}_{ABB}$, then the parties can store $[a]*_p[b]$ in $\mathcal{F}_{ABB}$, where $*_p$ represents multiplication modulo p. The notation for the multiplication operation would be:
   $[c] \leftarrow [a] * [b]$, where c contains the product $([a] *_p [b])$
4. Release: All the parties must be able to release a secret $[a]$ stored in $\mathcal{F}_{ABB}$ to all or a set of parties. We use the following notation to depict a public release of a secret and a private release of a secret to a particular party P respectively:

$b \leftarrow [a]$
$b \leftarrow_P [a]$
where b represents the value stored in the location a of $\mathcal{F}_{ABB}$

There are many implementations of the arithmetic black-box, including Shamir secret sharing [31, 32] and Pallier cryptosystem [33]. One can use these implementations, depending on the security requirement and efficiency expectations. We further append the operations of the arithmetic black-box with the following set of operations:

1. Comparison: All the parties can securely compare $[a]$ and $[b]$. We will use the following notation to compare two secrets:
   $[c] \leftarrow [a] \stackrel{?}{<} [b]$, where c stores 1 if $([a] < [b])$, else c stores 0
2. Equality: Similar to the comparison sub-protocol defined above, the equality protocol allows parties to securely check whether two stored values are equal. We use the following notation to denote a secure equality operation:
   $[c] \leftarrow [a] \stackrel{?}{=} [b]$, where c is 1 if $[a]$ equals $[b]$, else c equals 0
3. Private Modulo Reduction: The n parties holding $[a]$ and $[b]$ can securely compute $[a]$ *mod* $[b]$. The notation we use to signify this operation is as follows:
   $[c] \leftarrow [a]$ *mod* $[b]$, where c contains $[a]$ *mod* $[b]$

An implementation of the above operations can be found in [32]. The arithmetic black-box appended with the above mentioned operations will be termed as the extended arithmetic black-box and will be represented by the ideal functionality $\mathcal{F}_{ABB}$ itself.

Let $A = (a_{ij})_{k \times k}$ represent a matrix, then $[A]$ signifies that all the entries of the matrix are stored individually in the $\mathcal{F}_{ABB}$ functionality.

Further, we use a secure implementation of Dijkstra's Algorithm for computing single source shortest paths, from an adjacency matrix stored in the $\mathcal{F}_{ABB}$ functionality [22, 23]. This protocol will be used for securely computing closeness centrality of a node in a network, given in Section 5.2. We will use the following notation to signify a call to the sub-protocol for computing single source shortest path:

$$[d_1, d_2, \ldots, d_{|V|}] \leftarrow Dijkstra([A], [u])$$

where A is the adjacency matrix corresponding to the graph $G(V, E)$ under consideration, u represents the source vertex, d_i represents the distance from node u to node i. $[d_1, d_2, \ldots, d_{|V|}]$ is the shorthand notation for $[d_1], [d_2], \ldots, [d_{|V|}]$.

5 Secure Network Parameter Computation

Social network analysis (SNA) focuses on computing several network parameters to probe into the structural aspects of the network. In this section, we look at how a few well studied network parameters can be securely computed,

thus facilitating the study of previously inaccessible sensitive networks. The network parameters explored in this section include degree distribution, closeness centrality, Google PageRank and K-shell decomposition algorithm.

5.1 Degree distribution

Studying the degree distribution of a social network is one of the primary and simple steps in SNA. Unlike the binomial degree distribution in random graphs, real world complex networks depict the peculiar scale-free degree distribution [34–36]. Networks with scale-free degree distribution have very few nodes with high degree and majority of the nodes with low degree. The scale-free degree distribution is defined as:

$$P(degree \ = \ k) \propto k^{-\gamma}$$

The above distribution is determined by the parameter γ. Most real world complex networks including WWW [37], Internet [38] and various online social networks [39] depict scale-free degree distribution with γ between 2 and 3. Hence, investigating the degree distribution of unexplored sensitive networks can be an interesting direction to pursue. Consider, as an example, the hate network on employees of an organization. As discussed previously, each employee is assumed to have the knowledge of only those whom she hates in the network. That is, she is aware of only her out-going links in the network. Hence, there is no individual who has the global picture of the network. Additionally, an employee is unaware of the in-links to her, that is identity of employees who hate her. It is the in-degree distribution that would reveal the overall hatred present in the network. One can study the correlation between the distribution of hatred amongst employees in an organization and the overall productivity. In general, it would make a good study to observe the most widely occurring hatred distributions across several organizations.

In this section, we propose a protocol π_{ID} for securely computing the in-degree distribution of a directed network, held distributedly by a set of parties. The proposed protocol can be easily modified for computing out-degree distribution in directed networks and degree distribution in undirected networks.

In-degree distribution

The protocol π_{ID} assumes $[A]$ as its input, where A stores an unweighted adjacency matrix stored in the $\mathcal{F}_{ABB}$ functionality. The construction of $[A]$ from a set of parties is discussed in Appendix A. In steps 1-2 of the protocol, the parties securely compute the in-degree of every node in the graph, and the in-degree of node $i \in |V|$ is stored in id_i. In steps 3-7, we compute the number of nodes having in-degree j, which is represented as d_j. The proposed protocol requires $\Theta(|V|^2)$ addition and $\Theta(|V|^2)$ comparison operations.

Theorem 1. *The protocol π_{ID} securely implements $\mathcal{F}_{ID}$ with perfect security in the presence of the $\mathcal{F}_{ABB}$ functionality.*

Proof sketch. The sequence of $\mathcal{F}_{ABB}$ operations invoked by the π_{ID} protocol is a function of only the public value n i.e. the number of vertices in the graph under consideration. The correctness of the protocol follows trivially from the structure of the π_{ID} protocol. The security of the protocol follows from the UC theorem and the $\mathcal{F}_{ABB}$ functionality. □

Protocol 1 *in_degree_distribution*() π_{ID}

Input: $[A]$, where A stores an unweighted adjacency matrix
Output: $[d_0], [d_1], [d_2], \ldots [d_{|V|-1}]$, where d_i stores the number of vertices in the graph, represented by the adjacency matrix A, having i as their in-degree

1: **for** $i = 1$ to $|V|$ **do**
2: $\quad [id_i] \leftarrow \sum_{j=1}^{|V|} [a_{ji}]$
3: **for** $i = 1$ to $|V|$ **do**
4: $\quad$ **for** $j = 0$ to $|V| - 1$ **do**
5: $\quad\quad [d_{ij}] \leftarrow ([id_i] \stackrel{?}{=} j)$
6: **for** $j = 0$ to $|V| - 1$ **do**
7: $\quad [d_j] \leftarrow \sum_{i=1}^{|V|} [d_{ij}]$

5.2 Closeness centrality

Since its introduction in 1950 by Bavelas [40], closeness centrality measure has been one of the most widely utilized centrality measures. Closeness centrality of a node u in a graph G is defined as:

$$C(u) = \frac{1}{\sum_{i=1}^{|V|} d(u,i)}$$

where $d(u,i)$ represents the distance of node i from node u in graph G, which may be directed or undirected.

High closeness centrality nodes correspond to highly influential individuals in a social network [41]. A recent study [42] claims that clients are as responsible as sex-workers for the spreads of AIDS, since clients too correspond to highly central individuals in the sexual network. This hypothesis on sensitive networks, can be tested using the secure closeness centrality protocol we propose in this section. Further we provide the implementation details for securely computing the closeness centrality of a single node in the network. This protocol can be easily extended for computing the closeness measure of a set of nodes in the network.

The closeness centrality protocol π_C inputs $[A]$ and $[i]$, where A stores an adjacency matrix and i stores the index of a vertex. In step 1, we compute the

distance of node i from all the nodes in the network. This can be computed using the *Dijkstra()* protocol given in Section 4. In step 2, we securely add these distances to obtain the reciprocal of the closeness centrality of node $[i]$. The number of operations used in the protocol π_C is asymptotically the same as the Dijkstra's implementation, which uses $\Theta(|V|^3)$ additions operations, $\Theta(|V|^3)$ multiplications operations and $\Theta(|V|^2)$ comparisons/equality checks.

Protocol 2 *closeness_centrality*() π_C

Input: $[A]$, where A stores an adjacency matrix
$[i]$, where i stores the index of a vertex, which is an element of $\{1, 2, \ldots |V|\}$
Output: $[c_i]$, reciprocal of the closeness centrality of node $[i]$
1: $[d_{i1}, d_{i2}, \ldots, d_{in}] \leftarrow Dijkstra([A], i)$
2: $[c_i] \leftarrow \sum_{j=1}^{|V|} [d_{ij}]$

Theorem 2. *The closeness centrality protocol π_C securely implements $\mathcal{F}_C$ with perfect security in the presence of the $\mathcal{F}_{ABB}$ functionality.*

The proof follows on similar lines to that of Theorem 1.

5.3 Google PageRank

Larry Page and Sergey Brin developed the PageRank algorithm to better order the results fetched for a query on a search engine [43, 44]. The idea was to rank web pages on the Internet, based on the number and ranks of the in-links of the page. According to Google PageRank centrality, a node in a graph is said to be important if the nodes pointing to it are important. In social networks, high page rank valued individuals are correlated to highly influential and popular individuals [45]. Consider a *supply chain network* of several organizations, where an edge (u, v) would denote that organization u is a supplier (of raw material or an end product) to organization v. This is yet another example of a private network where parties (or organizations) would be unwilling to disclose their business relations [28]. Yet every organization would be interested in learning how influential it is, based on the strategic position it occupies in the network. In this section, we provide a secure implementation of an algorithm to determine the PageRank value of nodes, in a network distributedly held by individuals/organizations. There are numerous algorithms in the literature for computing the PageRank value of nodes, but in this paper we employ the random surfer algorithm.

Firstly, we discuss three sub-protocols, which will aid in realizing the secure PageRank algorithm.

The protocol *no_zero_outdegree*() inputs an unweighted adjacency matrix $[A]$. The protocol increases the out-degree of a node with zero out-degree to $|V|$, by adding outgoing links to all the nodes from the zero out-degree node.

The protocol returns this modified adjacency matrix. It uses $\Theta(|V|^2)$ addition operations and $\Theta(|V|^2)$ equality checks.

Protocol 3 *no_zero_outdegree*() π_{NOD}

Input: $[A]$, where A stores an unweighted adjacency matrix
Output: $[A]$, where A stores a modified input matrix, such that a row consisting of $|V|$ zeroes is converted into a row of $|V|$ ones

1: **for** $i = 1$ to $|V|$ **do**
2: $\quad [d_i] = \sum_{j=1}^{|V|} [a_{ij}]$
3: $\quad$ **for** $j = 1$ to $|V|$ **do**
4: $\quad\quad [a_{ij}] \leftarrow [a_{ij}] + ([d_i] \stackrel{?}{=} 0)$

Lemma 1. *The protocol π_{NOD} securely implements $\mathcal{F}_{NOD}$ with perfect security in the presence of the $\mathcal{F}_{ABB}$ functionality.*

Proof sketch. The correctness and privacy of the protocol follows directly from the protocol structure and it is easy to observe that the sequence of instructions is dependent only on the number of vertices. □

The protocol *random_number*() inputs a number k, where $k \in \mathbb{F}_p$, and outputs $[r]$, where $r \in_R \{1, 2, \ldots, k\}$. It uses $\Theta(n)$ addition operations, $\Theta(n)$ multiplication operations, $\Theta(n)$ comparison operations and $\Theta(1)$ private modulo reduction operations.

Protocol 4 *random_number*() π_{RAND}

Input: $k \in \mathbb{F}_p$
Output: $[r]$, where $[r] \in_R \{1, 2, \ldots, k\}$

1: $[r] \leftarrow 0$
2: **for** $i = 1$ to n **do**
3: $\quad [r_i] \leftarrow_{P_i} r_i$, where $r_i \in_R \{0, 1, 2, \ldots, k-1\}$
4: $\quad [r] \leftarrow [r] + ([r_i] * ([r_i] \stackrel{?}{<} [k]))$
5: $[r] \leftarrow ([r] \; mod \; [k]) + 1$

Lemma 2. *Let $r_1, r_2, \ldots, r_l \in_R \mathbb{Z}_m$ for some $m, l \geq 1$, where the set $\mathbb{Z}_m$ equals $\{0, 1, \ldots, m-1\}$, then $\left(\left(\sum_{i=1}^{l} r_i + c \right) \; mod \; m \right) \in_R \mathbb{Z}_m$, for any $c \in \mathbb{Z}_m$.*

The proof of the above lemma can be found in [30]

Lemma 3. *The protocol π_{RAND} securely implements $\mathcal{F}_{RAND}$ with perfect security in the presence of the $\mathcal{F}_{ABB}$ functionality.*

Proof sketch. In the i^{th} iteration of the for loop in step 2, if $r_i < k$ then r_i is added to r, and otherwise the value of r remains unchanged. After n iterations of the for loop at step 2, at least one honest party P_j would have added a random number $r_j \in_R \mathbb{Z}_k$. Hence, from lemma 2 the output of the protocol r is a random element of $\mathbb{Z}_k$. This completes the proof of correctness. The security follows directly from the fact that addition, comparison and private modulo reduction operations are provided by the $\mathcal{F}_{ABB}$ functionality. □

The protocol *random_neighbour*() inputs an unweighted matrix $[A]$ and a vertex $[cur]$. The protocol outputs a random neighbor $[u]$ of vertex $[cur]$ i.e. $[u] \in_R \{v \mid ([cur], v) \in E\}$. This protocol uses $\Theta(n|V|^2)$ additions operations, $\Theta(|V|^2)$ multiplications operations and $\Theta(|V|^2)$ comparison/equality checks.

Protocol 5 *random_neighbour*() π_{RN}

Input: $[A]$, where A stores an unweighted adjacency matrix, such that each row has at least one non-zero entry
$[cur]$, where *cur* stores the index of a vertex
Output: $[u] \in_R \{v|([cur], v) \in E\}$ i.e. u stores a random neighbour of $[cur]$

1: $[u] \leftarrow [0]$
2: **for** $i = 1$ to $|V|$ **do**
3: $\quad [temp] \leftarrow (i \stackrel{?}{=} [cur])$
4: $\quad$ **for** $j = 1$ to $|V|$ **do**
5: $\quad\quad$ **for** $k = 1$ to n **do**
6: $\quad\quad\quad [r_{jk}] \leftarrow_{P_k} r_{jk}$, where $r_k \in_R \mathbb{Z}_p$
7: $\quad\quad [r_j] \leftarrow \sum_{k=1}^{n} [r_{jk}]$
8: $\quad [min] \leftarrow [p-1]$
9: $\quad$ **for** $j = 1$ to $|V|$ **do**
10: $\quad\quad [check] \leftarrow [temp] * ([r_j] \stackrel{?}{<} [min]) * ([a_{ij}] \stackrel{?}{=} 1)$
11: $\quad\quad [min] \leftarrow [min] + [check] * ([r_j] - [min])$
12: $\quad\quad [u] \leftarrow [u] + ([j] - [u]) * [check]$

Lemma 4. *The protocol π_{RN} securely implements $\mathcal{F}_{RN}$ with statistical security in the presence of the $\mathcal{F}_{ABB}$ functionality.*

Proof sketch. No change is made to the value of variable u in $(|V|-1)$ iterations of the for loop on step 2. The only iteration where u is updated is when the index variable i equals the input vertex *cur*. To pick a random neighbor of *cur*, we associate a random number r_j with $a_{cur,j}$ entry of the matrix, for $1 \leq j \leq |V|$. The vertex u is updated with the index of the neighbor of *cur* associated with the least random number. Since all the random numbers are independently generated and no two random numbers are the same (which occurs with only negligible probability), we can conclude that we store a neighbor of vertex *cur* uniformly at random in u. □

Next we present a secure implementation of the PageRank computation algorithm. The algorithm inputs l, α and $[A]$, where l represents the length of the random walk we take on the underlying network $[A]$ and $(1-\alpha/p)$ is the teleportation probability in the random surfer model, with p being the size of the field $\mathbb{F}_p$. In steps 1-2 we initialize a variable $[count_i]$ for every vertex $i \in V$. At the end of the protocol run, the variable $count_i$ will contain the number of times vertex i was visited during the random walk. In steps 4-5 we pick a random vertex r and assign it to be the starting location of the random walk. In each iteration of the while loop on step 6, we hop from the current vertex $[cur]$ to a vertex v after updating the value $count_{cur}$. The vertex v is selected to be a random neighbor of cur with probability (α/p) and it is selected to be a random vertex in the network with probability $(1-\alpha/p)$. The proposed protocol pi_{PR} uses $\Theta(nl|V|^2)$ additions, $\Theta(l|V|^2)$ multiplications, $\Theta(l|V|^2)$ comparison/equality checks and $\Theta(l)$ private modulo reduction operations.

Protocol 6 *page_rank*() π_{PR}

Input: l, the length of the random walk
α, such that $(1-\alpha/p)$ is the teleportation probability
$[A]$, which is an unweighted adjacency matrix

Output: $[count_1], [count_2], \ldots [count_{|V|}]$, where $count_i$ stores the number of times vertex i is visited during the random walk

1: **for** $i = 1$ to $|V|$ **do**
2: $\quad [count_i] \leftarrow [0]$
3: $[A] \leftarrow no\_zero\_outdegree([A])$
4: $[r] \leftarrow random\_number([|V|])$
5: $[cur] \leftarrow [r]$
6: **while** $l > 0$ **do**
7: $\quad$ **for** $i = 1$ to $|V|$ **do**
8: $\quad\quad [count_i] \leftarrow [count_i] + ([cur] \stackrel{?}{=} i)$
9: $\quad$ **for** $i = 1$ to n **do**
10: $\quad\quad [r_i] \leftarrow_{P_i} r_i$, where $r_i \in_R \mathbf{Z}_p$
11: $\quad [r'] \leftarrow \sum_{i=1}^{n} [r_i]$
12: $\quad [flag] \leftarrow ([r'] \stackrel{?}{<} \alpha)$
13: $\quad [u] \leftarrow random\_neighbor([A], [cur])$
14: $\quad [v] \leftarrow [u] * [flag]$
15: $\quad [u] \leftarrow random\_number(|V|)$
16: $\quad [v] \leftarrow [v] + [u] * ([1] - [flag])$
17: $\quad [cur] \leftarrow [v]$
18: $\quad l = l - 1$

Theorem 3. *The protocol π_{PR} securely implements $\mathcal{F}_{PR}$ with statistical security in the presence of the $\mathcal{F}_{ABB}$ functionality.*

The proof of the above theorem follows directly from lemma 3, lemma 4 and the correctness of the protocol π_{PR}.

5.4 K-shell decomposition

Core-periphery structure is one of the most prominent and well studied mesoscale structures found in real world complex networks, including social networks. It was first introduced in 2000 by Borgatti and Everett [4]. A network is said to possess core-periphery structure if: the nodes in the network can be partitioned into two disjoint sets, namely, *core* and *periphery*; the set of core nodes are densely connected; periphery nodes are sparsely connected; periphery nodes are easily reachable from the core nodes. The set of core nodes are observed to be influential spreaders, since they play a key role in information diffusion [46].

In the year 2003, Batagelj and Zaversnik [47] used the K-shell algorithm for identifying the core-periphery structure in an undirected unweighted network. The K-shell algorithm assigns a shell number to each node, such that, higher the shell number, higher is the coreness coefficient of the node. The algorithm begins by assigning shell number 0 to isolated nodes. Then we prune nodes of degree 1, until the degree of all the nodes in the network is greater than 1. The nodes which are pruned are assigned shell number 1. Further the algorithm prunes nodes of degree 2 or less and assign these nodes shell number 2, and so on. The exact formulation of the K-shell algorithm can be found in [47]. In this section, we provide a secure implementation of the K-shell algorithm, which can be used for finding the set of influential spreaders securely in a distributedly held social network.

The protocol begins by initializing a few variables. For every vertex i, the variable $mark_i$ is initialized to 0, and is further set to 1 when node i is assigned its shell number. The variable $shell_i$ is initialized to 0 and is later updated to the shell number of node i. The variable cur_shell stores the current shell number, which is assigned to the nodes pruned in the current step. In each iteration of the for loop on step 5, we securely assign the shell number for precisely one node with its shell number. In steps 8-11, we find the least degree node u in the graph, which is to be pruned next. The value of cur_shell is updated in step 12 to the maximum of cur_shell and deg_u. In steps 13-15, we update the value $shell_u$ and $mark_u$ to the correct values of shell number of vertex u and 1 respectively. Finally, in line 16-19, we update the adjacency matrix under consideration by removing the node u and all its adjacent edges from the network. The proposed secure K-shell algorithm uses $\Theta(|V|^3)$ addition operations, $\Theta(|V|^3)$ multiplication operations and $\Theta(|V|^3)$ comparison/equality operations.

Next we present a proof of correctness of the k-shell decomposition protocol. A vertex u is said to be *marked* if $mark_u = 1$ and it is said to be *unmarked* otherwise. At the start of the protocol all the vertices in the graph are unmarked.

Lemma 5. *In the protocol π_{KD}, after $|V|$ iterations of the for loop on step 8, the variable u contains the index of the least degree unmarked vertex in the graph represented by the adjacency matrix A.*

Proof sketch. In steps 8-11, we traverse trough the list of all the vertices. We store the "current" minimum degree unmarked vertex stored in u. In case we

find an unmarked vertex v with degree lower than that of u, then we update u as v and we further update deg_u, which stores the degree of the "current" least degree unmarked vertex. □

Let $u^{(k)}$ represent the least degree unmarked vertex selected after the for loop on step 8 in the k^{th} iteration of the for loop on step 5.

Lemma 6. *In the k^{th} iteration of the for loop in step 5 of the protocol π_{KD}, the variable $shell_{u^{(k)}}$ is updated with correct shell number of vertex $u^{(k)}$.*

Proof sketch. This can be proved by using induction over the number of marked vertices. Let us assume that $(k-1)$ vertices have been marked and updated with their correct shell number. Then the vertex $u^{(k)}$ is marked in the k^{th} iteration of the for loop on step 11. The variable $shell_{u^{(k)}}$ is updated with the maximum of $shell_{u^{(k-1)}}$ and $|\{v \text{ is unmarked} | \{u^{(k)}, v\} \in E\}|$ i.e. the number of unmarked neighbors of $u^{(k)}$, which is the correct shell number for $u^{(k)}$ in accordance with the k-shell decomposition algorithm. □

Protocol 7 *kshell_decomposition()* π_{KD}

Input: $[A]$, where A is an unweighted undirected adjacency matrix with no self loops
Output: $[shell_1], [shell_2], [shell_3], \ldots [shell_{|V|}]$, where $shell_i$ stores the shell number of vertex i

1: **for** $i = 1$ to $|V|$ **do**
2: $\quad [mark_i] \leftarrow 0$
3: $\quad [shell_i] \leftarrow 0$
4: $[cur\_shell] \leftarrow 0$
5: **for** $iter = 1$ to $|V|$ **do**
6: $\quad [deg_u] \leftarrow |V|$
7: $\quad [u] \leftarrow 0$
8: $\quad$ **for** $i = 1$ to $|V|$ **do**
9: $\quad\quad [deg_i] \leftarrow \sum_{j=1}^{|V|} [a_{ij}]$
10: $\quad\quad [u] \leftarrow [u] + (([i] - [u]) * ([mark_i] \stackrel{?}{=} 0) * ([deg_i] \stackrel{?}{<} [deg_u]))$
11: $\quad\quad [deg_u] \leftarrow [deg_u] + (([deg_i] - [deg_u]) * (i \stackrel{?}{=} [u]))$
12: $\quad [cur\_shell] \leftarrow [deg_u] + (([cur\_shell] - [deg_u]) * ([cur\_shell] \stackrel{?}{>} [deg_u]))$
13: $\quad$ **for** $i = 1$ to $|V|$ **do**
14: $\quad\quad [shell_i] \leftarrow ([cur\_shell] * (i \stackrel{?}{=} [u])) + [shell_i]$
15: $\quad\quad [mark_i] \leftarrow (i \stackrel{?}{=} [u]) + [mark_i]$
16: $\quad$ **for** $i = 1$ to $|V|$ **do**
17: $\quad\quad$ **for** $j = 1$ to $|V|$ **do**
18: $\quad\quad\quad [a_{ij}] \leftarrow [a_{ij}] + ((0 - [a_{ij}]) * (i \stackrel{?}{=} [u]))$
19: $\quad\quad\quad [a_{ji}] \leftarrow [a_{ji}] + ((0 - [a_{ji}]) * (i \stackrel{?}{=} [u]))$

Theorem 4. *The protocol π_{KD} securely implements $\mathcal{F}_{KD}$ with perfect security in the presence of the $\mathcal{F}_{ABB}$ functionality.*

Proof sketch. The correctness of the algorithm follows directly from Lemma 6. The protocol π_{KD} has a well defined sequence of addition, multiplication and equality/comparison operations, which can be securely performed using the $\mathcal{F}_{ABB}$ functionality. □

6 Conclusions

Multiparty computation has been extensively applied in the domains of computational geometry, voting, bench-marking, etc. In this paper, we discuss on how MPC tools and techniques can be of interest to performing social network analysis. Study of sensitive networks, including financial transaction networks, sexual networks, trust networks and enmity networks, has largely been hampered by the unavailability of data due to privacy issues. It is mostly the case that these sensitive networks have the data distributedly held. In this paper, we present a set of MPC protocols which can be used to securely compute some network parameters on a distributedly held network. Network measures securely implemented include degree distribution, closeness centrality, PageRank and K-shell decomposition algorithm. To further build on this idea, one can securely implement other network parameters like reciprocity, homophily, betweenness, etc. Another important dimension to this work can be to improve on the efficiency of various network parameter protocols, using available MPC efficiency improvement techniques. One can further study the practical feasibility of the proposed MPC protocols for performing secure SNA on large sensitive networks. The broad aim of the paper is to highlight the possibility of exploring problems lying in the intersection of the two domains, namely, multiparty computation and private social networks.

References

1. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. science **286**(5439) (1999) 509–512
2. Barrat, A., Weigt, M.: On the properties of small-world network models. The European Physical Journal B-Condensed Matter and Complex Systems **13**(3) (2000) 547–560
3. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proceedings of the national academy of sciences **99**(12) (2002) 7821–7826
4. Borgatti, S.P., Everett, M.G.: Models of core/periphery structures. Social networks **21**(4) (2000) 375–395
5. Eguiluz, V.M., Chialvo, D.R., Cecchi, G.A., Baliki, M., Apkarian, A.V.: Scale-free brain functional networks. Physical review letters **94**(1) (2005) 018102
6. Kim, Y., Choi, T.Y., Yan, T., Dooley, K.: Structural investigation of supply networks: A social network analysis approach. Journal of Operations Management **29**(3) (2011) 194–211
7. Easley, D., Kleinberg, J., et al.: Networks, crowds, and markets: Reasoning about a highly connected world. Significance **9** (2012) 43–44

8. Liljeros, F., Giesecke, J., Holme, P.: The contact network of inpatients in a regional healthcare system. a longitudinal case study. Mathematical Population Studies **14**(4) (2007) 269–284
9. Rocha, L.E., Liljeros, F., Holme, P.: Simulated epidemics in an empirical spatiotemporal network of 50,185 sexual contacts. PLoS Comput Biol **7**(3) (2011) e1001109
10. Heatherly, R., Kantarcioglu, M., Thuraisingham, B.: Preventing private information inference attacks on social networks. Knowledge and Data Engineering, IEEE Transactions on **25**(8) (2013) 1849–1862
11. Tassa, T., Cohen, D.J.: Anonymization of centralized and distributed social networks by sequential clustering. Knowledge and Data Engineering, IEEE Transactions on **25**(2) (2013) 311–324
12. Narayanan, A., Shmatikov, V.: De-anonymizing social networks. In: Security and Privacy, 2009 30th IEEE Symposium on, IEEE (2009) 173–187
13. Narayanan, A., Shi, E., Rubinstein, B.I.: Link prediction by de-anonymization: How we won the kaggle social network challenge. In: Neural Networks (IJCNN), The 2011 International Joint Conference on, IEEE (2011) 1825–1834
14. Bhaskar, R., Laxman, S., Smith, A., Thakurta, A.: Discovering frequent patterns in sensitive data. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM (2010) 503–512
15. Kleinberg, J.M.: Challenges in mining social network data: processes, privacy, and paradoxes. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM (2007) 4–5
16. Narayanan, A., Shmatikov, V.: Robust de-anonymization of large sparse datasets. In: Security and Privacy, 2008. SP 2008. IEEE Symposium on, IEEE (2008) 111–125
17. Xue, M., Karras, P., Chedy, R., Kalnis, P., Pung, H.K.: Delineating social network data anonymization via random edge perturbation. In: Proceedings of the 21st ACM international conference on Information and knowledge management, ACM (2012) 475–484
18. Fard, A.M., Wang, K.: Neighborhood randomization for link privacy in social network analysis. World Wide Web **18**(1) (2015) 9–32
19. Hogg, T., Adamic, L.: Enhancing reputation mechanisms via online social networks. In: Proceedings of the 5th ACM conference on Electronic commerce, ACM (2004) 236–237
20. Yao, A.C.C.: Protocols for secure computations. FOCS **82** (1982) 160–164
21. Brickell, J., Shmatikov, V.: Privacy-preserving graph algorithms in the semi-honest model. In: Advances in Cryptology-ASIACRYPT 2005. Springer (2005) 236–252
22. Aly, A., Cuvelier, E., Mawet, S., Pereira, O., Van Vyve, M.: Securely solving simple combinatorial graph problems. In: Financial Cryptography and Data Security. Springer (2013) 239–257
23. Aly, A., Van Vyve, M.: Securely solving classical network flow problems. In: Information Security and Cryptology-ICISC 2014. Springer (2014) 205–221
24. Blanton, M., Steele, A., Alisagari, M.: Data-oblivious graph algorithms for secure computation and outsourcing. In: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, ACM (2013) 207–218
25. 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, ACM (2006) 89–98

26. Kukkala, V.B., Iyengar, S., Saini, J.S.: Secure multiparty graph computation. In: 2016 8th International Conference on Communication Systems and Networks (COMSNETS), IEEE (2016) 1–2
27. 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, ACM (2008) 9–14
28. Fridgen, G., Garizy, T.Z.: Supply chain network risk analysis - A privacy preserving approach. In: 23rd European Conference on Information Systems, ECIS 2015, Münster, Germany, May 26-29, 2015. (2015)
29. Tassa, T., Bonchi, F.: Privacy preserving estimation of social influence. In: EDBT. (2014) 559–570
30. Cramer, R., Damgard, I., Nielsen, J.B.: Secure multiparty computation and secret sharing-an information theoretic appoach. Book Draft (2012)
31. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the twentieth annual ACM symposium on Theory of computing, ACM (1988) 1–10
32. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Theory of Cryptography. Springer (2006) 285–304
33. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques, Springer (1999) 223–238
34. Strogatz, S.H.: Exploring complex networks. Nature **410**(6825) (2001) 268–276
35. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of modern physics **74**(1) (2002) 47
36. Dorogovtsev, S.N., Mendes, J.F.: Evolution of networks. Advances in physics **51**(4) (2002) 1079–1187
37. Adamic, L.A., Huberman, B.A.: Power-law distribution of the world wide web. Science **287**(5461) (2000) 2115–2115
38. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: ACM SIGCOMM computer communication review. Volume 29., ACM (1999) 251–262
39. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, ACM (2007) 29–42
40. Bavelas, A.: Communication patterns in task-oriented groups. Journal of the acoustical society of America (1950)
41. Wasserman, S., Faust, K.: Social network analysis: Methods and applications. Volume 8. Cambridge university press (1994)
42. Hsieh, C.S., Kovářík, J., Logan, T.: How central are clients in sexual networks created by commercial sex? Scientific reports **4** (2014)
43. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: bringing order to the web. (1999)
44. Brin, S., Page, L.: Reprint of: The anatomy of a large-scale hypertextual web search engine. Computer networks **56**(18) (2012) 3825–3833
45. Franceschet, M.: Pagerank: Standing on the shoulders of giants. Communications of the ACM **54**(6) (2011) 92–101
46. Kitsak, M., Gallos, L.K., Havlin, S., Liljeros, F., Muchnik, L., Stanley, H.E., Makse, H.A.: Identification of influential spreaders in complex networks. Nature physics **6**(11) (2010) 888–893

47. Batagelj, V., Zaversnik, M.: An o (m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003)
48. Kissner, L., Song, D.: Privacy-preserving set operations. In: Advances in Cryptology–CRYPTO 2005, Springer (2005) 241–257

Appendix A Discretionary Sub-Protocols

This section presents various functionalities which will be used to store adjacency matrices of various forms in the $\mathcal{F}_{ABB}$ functionality. These set of implementations are termed as *discretionary sub-protocols*. These include a set of four protocols for constructing the adjacency matrix, $force\_boolean()$ protocol for converting the input adjacency matrix into a symmetric adjacency matrix, $remove\_self\_loops()$ protocol for removing self-loops of the input adjacency matrix and $check\_symmetricity()$ protocol to check if the input adjacency matrix represents a directed or an undirected network.

A.1 Adjacency Matrix Construction

A network can be represented using adjacency lists, an adjacency matrix or an incidence matrix. For security concerns, we deal with only the adjacency matrix representation of a network. As depicted in the work presented by Kerschbaum and Schaad [27], sometimes the network is held by external agents (individuals not a part of the network), who collaborate to perform SNA securely. Whereas, sometimes the individuals who are a part of the network themselves (internal agents) may be the ones collaborating, as shown in the work by Keith Frikken and Philippe Golle [25]. In this section we propose four variations of the adjacency matrix construction protocol, which can be employed depending on whether the network to be constructed is weighted/unweighted and whether the parties are internal or external agents. The aim of these protocols is to construct the adjacency matrix A of the underlying graph G. It is assumed that the vertex set V is public, which if not the case, can be securely computed using secure set intersection protocols [48]. The scenario of internal and external agents results in the input of all the n parties to be given in one of the following two forms:

1. External agents: Each party P_i has a graph G_i as her private input, such that $G = \bigcup_{i=1}^{n} G_i$
2. Internal agents: Each party P_i represents a vertex in G and has her adjacency vector v_i as the private input

Protocol 8 *weighted_external_adjacency_construction*() $\pi_{adj}^{(1)}$

Output: $[A]$, where A stores a weighted adjacency matrix

1: **for** j = 1 to $|V|$ **do**
2: **for** k= 1 to $|V|$ **do**
3: $[min] \leftarrow [p-1]$
4: **for** i = 1 to n **do**
5: $[a_{jk}^{(i)}] \leftarrow_{P_i} a_{jk}^{(i)}$
6: $[min] \leftarrow [min] * ([min] \overset{?}{<} [a_{jk}^{(i)}]) + [a_{jk}^{(i)}] * ([1] - ([min] \overset{?}{<} [a_{jk}^{(i)}]))$
7: $[a_{jk}] \leftarrow [min]$

Protocol 9 *weighted_internal_adjacency_construction*() $\pi_{adj}^{(2)}$

Output: $[A]$, where A stores a weighted adjacency matrix

1: **for** i = 1 to $|V|$ **do**
2: **for** j = 1 to $|V|$ **do**
3: $[a_{ij}] \leftarrow_{P_i} a_{ij}$

Protocol 10 *unweighted_external_adjacency_construction*() $\pi_{adj}^{(3)}$

Output: $[A]$, where A stores an unweighted adjacency matrix

1: **for** i = 1 to n **do**
2: **for** j = 1 to $|V|$ **do**
3: **for** k= 1 to $|V|$ **do**
4: $[a_{jk}^{(i)}] \leftarrow_{P_i} a_{jk}^{(i)}$
5: $[a_{jk}] \leftarrow [1] - \prod_{i=1}^{|V|}([1] - [a_{jk}^{(i)}])$
6: $[A] \leftarrow force\_boolean([A])$

Protocol 11 unweighted_internal_adjacency_construction $\pi_{adj}^{(4)}$

Output: $[A]$, where A stores an unweighted adjacency matrix

1: $[A] \leftarrow weighted\_internal\_adjacency\_construction()$
2: $[A] \leftarrow force\_boolean([A])$

Further the graph G in each case mentioned above could be weighted or unweighted. Thus, the arising four variations can be handled as mentioned below:

1. Weighted adjacency matrix construction by external parties: In the weighted adjacency matrix case, the weight of an edge e is defined as the minimum of the weights of e in the graphs G_i for $1 \leq i \leq n$. The proposed protocol uses $\Theta(n|V|^2)$ addition operations, $\Theta(n|V|^2)$ multiplication operations and $\Theta(n|V|^2)$ comparison operations.
2. Weighted adjacency matrix construction by internal parties: In this protocol each party stores her adjacency vector entries in the $\mathcal{F}_{ABB}$ functionality.
3. Unweighted adjacency matrix construction by external parties: This protocol uses $\Theta(|V|^3)$ addition operations and $\Theta(|V|^3)$ multiplication operations.
4. Unweighted adjacency matrix construction by internal parties: This protocol can be easily constructed using the $weighted\_internal\_adjacency\_construction()$ and $force\_boolean()$ protocols. The cost involved in protocol $\pi_{adj}^{(4)}$ is the same as the π_{BOOL} or $force\_boolean()$ protocol i.e. $\Theta(|V|^2)$ addition operations and $\Theta(|V|^2)$ equality checks.

A.2 Additional Security Sub-protocols

In this section, we present a set of sub-protocols that can be used to enforce a few additional constraints on the input network.

The protocol $check\_symmetricity()$ inputs an adjacency matrix $[A]$. It checks if a_{ij} equals a_{ji} for $1 \leq i, j \leq n$. If even one check fails, the protocol returns 0, else it returns 1. This protocol uses $\Theta(|V|^2)$ additions and $\Theta(|V|^2)$ equality checks.

The $force\_boolean()$ protocol takes as input an adjacency matrix $[A]$. The protocol checks if each entry of the adjacency matrix is 0/1, if not the case, then the corresponding entries are rounded to 1. Hence, the output of the algorithm is an adjacency matrix corresponding to an unweighted graph. This protocol uses $\Theta(|V|^2)$ addition operations and $\Theta(|V|^2)$ equality checks.

To ensure that a network does not contain any self-loops, we need to force all the diagonal entries of the corresponding adjacency matrix to 0. Protocol $remove\_self\_loops()$ inputs an adjacency matrix $[A]$ and outputs the modified adjacency matrix with no self-loops.

Protocol 12 $check\_symmetricity()$ π_{SYM}

Input: $[A]$, where A stores an adjacency matrix
Output: $[r]$, where r stores 1 if A is symmetric and 0 otherwise

1: $[temp] \leftarrow [0]$
2: **for** i = 1 to $|V| - 1$ **do**
3: **for** j = (i+1) to $|V|$ **do**
4: $[temp] \leftarrow [temp] + [1] - ([a_{ij}] \stackrel{?}{=} [a_{ji}])$
5: $[r] \leftarrow ([temp] \stackrel{?}{=} [0])$

Protocol 13 $force\_boolean()$ π_{BOOL}

Input: $[A]$, where A stores an adjacency matrix
Output: $[A]$, where A stores an unweighted adjacency matrix

```
1: for i = 1 to |V| do
2:     for j = 1 to |V| do
3:         [a_ij] ← [1] − ([a_ij] ?= 0)
```

Protocol 14 $remove\_self\_loops()$ π_{SL}

Input: $[A]$, where A stores an adjacency matrix
Output: $[A]$, where A is the same as the input but with self-loops removed

```
1: for i = 1 to |V| do
2:     [a_ii] ← 0
```