Network Deprived SNA : An Alternative To Anonymization

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 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 various social network analysis methods. The network parameters addressed in this paper include degree distribution, clustering coefficient, closeness centrality, pagerank algorithm and K-shell decomposition algorithm. The designed protocols are secure in the malicious adversarial model with less than one third corrupt parties.

Keywords: Privacy preserving 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 these structural properties 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 network structure, reveals the emergent collective behavior observed globally. Hence, it is the structure 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 conclusions 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 therefore the negative 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 entire global network. However, each individual would be aware of the directed edges that she has to other individuals in the network (local view). This shows the possibility where the knowledge of the network structure is distributely held by several individuals. Secondly, even if an individual 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 in performing 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 perform the required analysis while the network is kept hidden, since it is results of the analysis that we are interested in rather than learning the network structure. For example, if it is just the average clustering coefficient 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 structure.

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 is abstracted to a multiparty setting as follows: designing a protocol to securely determine a globally known function $y = F(x_1, x_2, \dots, x_n)$, 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 $F(x_1, x_2, \dots, x_n) = p$, 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, 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 provide implementation details for securely computing degree distribution, clustering coefficient, 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 section 6 as discretionary sub-protocols .

The designed protocols work under the malicious adversarial model. The assumption of the adversarial model is based on the implementation of the tools and techniques, described as building blocks, in section 4. All the proposed network parameter protocols except PageRank protocol are perfectly secure, while PageRank protocol is statistically secure. We calculate the cost involved in each of the proposed protocol, in terms of the number of 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 above mentioned operations. We discuss these operations in Section 4.

2 Related work

Seminal works in MPC prove that any function can be securely computed in the standard cryptographic model [21–23]. Similar results are proved in the information theoretic model as well [24, 25]. However, the current literature aims at designing problem specific efficient protocols.

Many recent works have focused on securely implementing graph algorithms. Brickell and Shamatikov [26] 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 parties. Moreover, the adversarial model assumed is that of the semi-honest model. Aly et al. [27] 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 [28] 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. [29], 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. [29].

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 [30]. 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. [31], 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 necessary inferences without revealing the network structure.

Kerschbaum and Schaad [32] 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 [33]. 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 [34] 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 to find important nodes in the network is based entirely on the network structure and relies on the standard techniques of measuring centrality ranking of the 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 Network Science

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 $u, v \in V$.

In the definition of a graph above, $(u, v) \in E$ or $(u, v) \notin E$ i.e. each ordered pair is either present in the graph or absent. We term these networks as unweighted graphs as well. A generalization of the above would be to associate a real value $w(u, v)$ with each edge $(u, v) \in E$, which are termed as weighted networks. For example, hate network can be a weighted directed network, with the weights representing the extent of hatred. On the other hand, Facebook friendship network is an unweighted undirected network.

A graph $G'(V', E')$ is said to be a subgraph of $G(V, E)$ if $V' \subseteq V$, $E' \subseteq E$ and E' is a relation on V'. The union of the undirected graphs $G_1(V_1, E_1)$, $G_2(V_2, E_2)$, $\ldots, G_k(V_k, E_k)$ represented as $G(V, E) = \bigcup_{i=1}^{k} G_i(V_i, E_i)$ is defined as $V = \bigcup_{i=1}^{k} V_i$ and $E = \bigcup_{i=1}^{k} E_i$. The union of weighted graphs $G_1(V_1, E_1)$, $G_2(V_2, E_2)$, $\ldots, G_k(V_k, E_k)$ represented as $G(V, E) = \bigcup_{i=1}^{k} G_i(V_i, E_i)$ is defined as $V = \bigcup_{i=1}^{k} V_i$ and $w(u, v) = min_{1 \leq i \leq k} w_i(u, v)$, where w is the edge weight function for G and w_i is the edge weight function for G_i.

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 as well, 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 cryptographic protocol and aid in providing the rigorous proof for the same. The notations used here are borrowed from the comprehensive work [35] written by Cramer et al. The section consists of only those definitions and terminologies that are essential and suggest [35] for further reading.

Definition 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 denote 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 additional information by combining their views. A party is said to be *malicious* when no assumptions are made regarding her behavior. Such a party is given the full 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. The protocols designed in this paper assume the presence of malicious parties to model corruption, known as the malicious adversarial model. The corruption model in which all the corrupt parties are semi-honest is said to be the semi-honest adversarial model.

In order to keep the corruption model simple, 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. In a semi-honest adversarial model, the central adversary is capable of examining anything that the corrupt parties see during the run of the protocol. That is, the adversary is given access to the transcripts of all the corrupt parties. Transcript of a party P_i is the sequence of data that is visible to P_i, during the run of the protocol. This would include its input, a random vector that is used to model the randomness in its decisions during the protocol run, the messages it receives from other parties and finally the output it receives. The transcript is also sometimes referred to as the party's view. In order for the protocol to be secure under this model, we must ensure that the adversary does not learn any additional information by combining the transcripts of all the corrupt parties. Under the malicious adversarial model, not only does the central adversary have access to the transcripts of the corrupt parties, but it also decides on how the corrupt parties behave during the execution of the protocol.

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 must be the same as the 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 by the party. A protocol is said to be robust if the effect of an actual influence is obtainable from allowed influence.

One can observe that for proving security under the semi-honest adversarial model, it is sufficient to prove that the protocol preserves privacy, when the proof of correctness is implicit in the structure of the designed protocol. However, proof of correctness must be explicitly stated in case of the malicious adversarial model. Similarly, the proof of robustness is applicable only when we consider the malicious adversarial model.

A protocol is said to be private if the information in the transcript of the party is no more than the input and output of the party. Therefore, a protocol is private if there exist 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 corrupt parties. Here, C denotes the set of all corrupt parties. 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 exist 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 as well as the assumptions about 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 and hence can launch attacks that are feasible in polynomial time complexity only, we say that the protocol is computationally secure. Let us say π makes no assumptions about the computational capabilities of the adversary, and the adversary is assumed to have unbounded resources. In such a case, 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.

Universally Compositibility Theorem

Before stating the Universally 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 [35]. Intuitively, an ideal functionality $\mathcal{F}$ is a private and robust 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 $\mathcal{F}_H$. Therefore, 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 for proving the security of a complex protocol.

Secret Sharing

Many MPC protocols are based on a secret sharing scheme. A secret sharing scheme allows a party to distribute shares of her secret s with all the n parties, such that any threshold t or more number of parties can pool in their shares together to reconstruct the secret s. This is also known as an (n, t) threshold secret sharing scheme. As an example, let us consider Shamir secret sharing scheme [36]. Here, a party P constructs a polynomial $f(x) = s + a_1x + a_2x^2 + \cdots + a_{(t-1)}x^{(t-1)}$, where s is the secret to be shared, $a_i \in_R \mathbb{Z}_p$ for all i, and p is a prime number. Then party P shares the value $f(i)$ with the party P_i. It follows from Lagrange's Interpolation theorem that any collaboration of less than t parties cannot reveal any information about the secret s, whereas the collaboration of any t or more parties can determine the value of the constant term of the function. In the case above, we assumed that the party P shared consistent shares of its secret s with all the parties. However, when proving security in the malicious adversarial model, we need to ensure that the shares of the secret are consistent. A secret sharing scheme which ensures that all the distributed shares of a secret are consistent, is known as a *verifiable secret sharing scheme*. We point the curious reader to work of Chor et. al [37] for a more detailed illustration on verifiable secret sharing schemes.

4 Building Blocks

In this section, we describe the basic notations used throughout the paper along with some frequently used preliminary techniques. Each MPC protocol consists of n parties denoted as $P_1, P_2, \ldots, P_n$. We assume that there exists a secure channel of communication between any two parties. Further, we suppose that each party shares her secret using a verifiable secret sharing scheme (VSS) over a finite field $\mathbb{F}_p$, where p is a prime number. We use the notation $[a]$ to depict that $a \in \mathbb{F}_p$ is distributedly held by the n parties using a VSS scheme. We will assume that the VSS scheme provides the following operations as sub-protocols:

1. Addition: If n parties hold $[a]$ and $[b]$, then they can compute $[a +_p b]$, where $+_p$ represents addition modulo p. We will denote an addition operation as follows:
   $[c] \leftarrow [a] + [b]$, where c contains the sum $(a +_p b)$
2. Multiplication: The n parties sharing $[a]$ and $[b]$ can securely compute $[a *_p b]$, where $*_p$ represents multiplication modulo p. The notation for the multiplication operation would be:
   $[c] \leftarrow [a] * [b]$, where c contains the product $(a *_p b)$
3. Comparison: All the parties can securely compare $[a]$ and $[b]$. We will use the following notation to compare two secrets:
   $[c] \leftarrow [a < b]$, where c is 1 if $(a < b)$, else c equals 0
4. Equality: Similar to the comparison sub-protocol defined above, the equality protocol allows parties to securely check whether two shared secrets are equal or not. We use the following notation to denote a secure equality operation:
   $[c] \leftarrow [a = b]$, where c is 1 if a equals b, else c equals 0
5. 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 $a \; mod \; b$
6. Release: All the parties must be able to release a secret $[a]$ in public or to a specific party. We use the following notation to depict a public release and a private release to party P respectively:
   $a \leftarrow [a] \qquad a \leftarrow_P [a]$

Many VSS schemes like Shamir Secret Sharing scheme allow a secure implementation of the above mentioned operations [25, 38]. As shown in [25], any function comprising of the above mentioned operations can be securely computed using a VSS scheme with perfect security in the malicious adversarial model with less than n/3 corrupt parties.

Let $A = (a_{ij})_{k \times k}$ represent a matrix, then $[A]$ represents that all the entries of the matrix are distributedly held by the n parties using the above mentioned VSS scheme.

Further, we suppose a secure implementation of Dijkstra's Algorithm is available, for computing single source shortest paths, using a distributedly held adjacency matrix of the graph [27, 28]. This protocol will be used in securely computing closeness centrality of a node in a network, given in section 5.3. 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 and $[d_1, d_2, \ldots, d_{|V|}]$ 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 network parameters can be securely computed, thus facilitating the study of previously inaccessible networks. The network parameters explored in this section include degree distribution, clustering coefficient, 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 [39–41]. 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:

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

In the above distribution is determined by the parameter γ. Most real world complex networks including WWW [42], internet [43] and various online social networks [44] 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 one 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} commences by invoking the $adjacency\_matrix\_construction()$

sub-protocol denoted by π_{ADJ}, which allows all parties to securely compute $[A]$. Here A is the adjacency matrix representing the network under consideration. The implementation details of π_{ADJ} is given in section 6.1. In steps 2-3, the parties securely compute the in-degree of every node in the graph, and the in-degree of node $i \in |V|$ is stored as id_i. In steps 4-8, we compute the number of nodes having in-degree j, which is represented as d_j. In steps 9-10, we release the in-degree distribution of the directed network in public. The proposed protocol requires $\Theta(|V|^2)$ additions and $\Theta(|V|^2)$ comparisons, excluding the number operations involved in the *adjacency_matrix_construction*() protocol. The cost involved in the *adjacency_matrix_construction*() protocol is dependent on the type of network to be constructed, and hence we provide a detailed discussion on it in section 6.1. Henceforth, while computing the number of operations involved in network parameter protocols, we will ignore the operations involved in the *adjacency_matrix_construction*() sub-protocol.

Protocol 1 *in_degree_distribution*() π_{ID}

```
Output: $(d_0, d_1, d_2, \dots d_{|V|-1})$
 1: $[A] \leftarrow adjacency\_matrix\_construction()$
 2: for i = 1 to $|V|$ do
 3:     $[id_i] \leftarrow \sum_{j=1}^{|V|} [a_{ji}]$
 4: for i = 1 to $|V|$ do
 5:     for j = 0 to $|V| - 1$ do
 6:         $[d_{ij}] \leftarrow [id_i = j]$
 7: for j = 0 to $|V| - 1$ do
 8:     $[d_j] \leftarrow \sum_{i=1}^{|V|} [d_{ij}]$
 9: for j = 0 to $|V| - 1$ do
10:     $d_j \leftarrow [d_j]$
```

Theorem 1. *The protocol π_{ID} securely implements $\mathcal{F}_{ID}$ with perfect security in the malicious adversarial model with less than $n/3$ corrupt parties.*

Proof. Let F_{adj} be the ideal functionality for constructing the adjacency matrix. The protocol π_{ID} begins with the construction of the adjacency matrix by invoking F_{adj} as a sub-protocol. In steps 2 to 8, we perform a predetermined sequence of addition and comparison operations. Secure implementation of these operations are implicitly provided by the assumed secret sharing scheme, described in section 4. As seen previously, the operations can be performed securely in the malicious adversarial model, with less than $n/3$ corrupt parties (details in section 4). In steps 9 to 10, we globally release the required degree distribution of the underlying network, where the revealed value d_j represents the number of nodes with degree j. The correctness of the protocol is visible by the structure of the designed protocol. To complete the proof of security, we employ the UC theorem to replace the ideal functionality F_{adj} with π_{adj} that securely implements it, as given in section 6.1.

$\square$

5.2 Clustering Coefficient

The German sociologist, Georg Simmel put forth the hypothesis of *triadic closure*, which states that two individuals are more likely to be friends in the future if they have common friends [45]. With the advent of computing technology and the Internet, this hypothesis was later confirmed to be true in many large-scale online social networks [44]. This concept was later reformed as *clustering coefficient* by Watts and Strogatz [46]. Intuitively, a node in a friendship network is said to have a large clustering coefficient if many of the node's friends are friends within themselves and vice-versa. It would be interesting to check whether many nodes in a trust network have large clustering coefficients, as is the case with friendship networks. This motivates us to propose a secure multiparty computation protocol for securely computing the clustering coefficient of nodes in a distributedly held social network. In the following sub-section we provide an MPC protocol for securely computing the clustering coefficient of a node in an undirected network. The proposed protocol can easily be modified for computing the clustering coefficient of nodes in a directed social network and for computing global clustering coefficient of the network as well.

Undirected Clustering Coefficient

In an undirected graph $G(V, E)$, represented by a symmetric adjacency matrix A, the *local clustering coefficient* of node i with degree k is defined as:

$$CC(i) = \sum_{j=1}^{|V|} \left(a_{ij} * \left(\sum_{k=1}^{|V|} (a_{jk} * a_{ki}) \right) \right) / \binom{k}{2} \tag{1}$$

The numerator of $CC(i)$ represents the number of edges in G whose end points are adjacent to node i, while the denominator represents maximum number of such edges. The clustering coefficient of a node always lies between 0 and 1.

Protocol 2 *clustering_coefficient*() π_{CC}

```
Input: $i \in \{1, 2, \ldots n\}$
Output: $CC(i)$ to party $P_i$
 1: $[A] \leftarrow$ unweighted_internal_adjacency_matrix_construction()
 2: $x \leftarrow$ check_symmetricity($[A]$)
 3: $[TC_i] \leftarrow 0$
 4: if $x == 0$ then
 5:     Abort the protocol
 6: for j=1 to $|V|$ do
 7:     if j $\neq$ i then
 8:         for k = 1 to $|V|$ do
 9:             if k $\neq$ j and k $\neq$ i then
10:                 $[TC_i] \leftarrow [TC_i] + ( [a_{ij} = 1] * [a_{jk} = 1] * [a_{ki} = 1] )$
11: $[TC_i]$ is revealed to party $P_i$
12: $P_i$ computes $CC(i) = TC_i / \binom{k}{2}$, where $k$ is her degree
```

The proposed protocol assumes that each party represents a vertex in the graph under consideration. Therefore, each party is expected to input her adjacency vector (i.e. her outgoing edges) to the adjacency construction sub-protocol. To construct the unweighted social network we use a variation of the adjacency matrix construction protocol, namely, *unweighted_internal_adjacency_matrix _construction*(). The implementation details of this protocol can be found in Section 6.1. In the clustering coefficient protocol, we call the sub-protocol *check _symmetricity*() to check whether the network under consideration is undirected or not. If the check fails, then we abort the protocol, else we continue. The implementation details of *check_symmetricity*() protocol can be found in Section 6.1. Computing the clustering coefficient of node i in the network can be reduced to counting the total number of triangles present on node i. The protocol determines the required triangle count in steps 6 to 10. The triangle count of vertex i is released to party P_i, who further computes her clustering coefficient using the triangle count TC_i and degree k. The protocol uses $\Theta(|V|^2)$ comparison operations and $\Theta(|V|^2)$ addition operations.

Theorem 2. *The protocol π_{CC} securely implements $\mathcal{F}_{CC}$ with perfect security in the malicious adversarial model with less than $n/3$ corrupt parties.*

The proof of the above theorem follows on similar lines as that provided in Theorem 1.

5.3 Closeness Centrality

Since its introduction in 1950 by Bavelas [47], 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:

$$C(u) = \frac{1}{\sum_{i=1}^{|V|} d(u,i)} \tag{2}$$

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[48]. A recent study [49] 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 inputs the index i of the node whose closeness centrality is to be computed. The protocol begins by securely constructing the underlying network. Next, in step 2, 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 3, we securely add these distances to obtain the denominator of the closeness centrality of node i. Further we reveal the computed sum of distances to required set of parties. 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 3 $closeness\_centrality()$ π_C

```
Input: $i \in \{1, 2, \ldots |V|\}$
 1: $[A] \leftarrow Adjacency\_Matrix\_Construction()$
 2: $[d_{i1}, d_{i2}, \ldots, d_{in}] \leftarrow Dijkstra(\ [A],\ i)$
 3: $[c_i] \leftarrow \sum_{j=1}^{|V|} [d_{ij}]$
 4: Output construction
```

Theorem 3. *The closeness centrality protocol π_C securely implements $\mathcal{F}_C$ with perfect security in the malicious adversarial model with less than $n/3$ corrupt parties.*

5.4 Google PageRank Protocol

Larry Page and Sergey Brin developed the PageRank algorithm to better order the results fetched for a query on a search engine [50, 51]. The idea was to rank web pages on the internet, based on the number and ranks of 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 [52]. 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 [53]. 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 be useful for abstracting some functionalities from the secure PageRank algorithm.

The protocol *no_zero_outdegree*() inputs a distributedly held 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 $\Theta(|V|^2)$ additions operations, $\Theta(|V|^2)$ multiplications operations and $\Theta(|V|^2)$ equality checks.

Protocol 4 *no_zero_outdegree*() π_{NOD}

```
Input: $[A]$
Output: $[A]$
1: for i=1 to $|V|$ do
2:     $[d_i] = \sum_{j=1}^{|V|} [a_{ij}]$
3:     for j=1 to $|V|$ do
4:         $[a_{ij}] \leftarrow [a_{ij}] + [d_i = 0] * [1]$
```

Lemma 1. *The protocol π_{NOD} securely implements $\mathcal{F}_{NOD}$ with perfect security in the malicious adversarial model with less than $n/3$ corrupt parties.*

The protocol *random_number*() inputs a number k, where $k \in \mathbb{Z}_p$, 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 5 *random_number*() π_{RAND}

```
Input: $k$
Output: $[r]$
1: $[r] \leftarrow 0$
2: for i=1 to n do
3:     $P_i$ distributes $[r_i]$, where $r_i \in_R \mathbb{Z}_p$
4:     $[r] \leftarrow [r] + [r_i] * [r_i < k]$
5: $[r] \leftarrow [r \ mod \ k] + 1$
```

Lemma 2. *Let $r_1, r_2, \ldots, r_l \in_R \mathbb{Z}_m$ for $m \geq 1$, then* $\left(\sum_{i=1}^{l} r_i \ mod \ m\right) \in_R \mathbb{Z}_m$.

Lemma 3. *The protocol π_{RAND} securely implements $\mathcal{F}_{RAND}$ with perfect security in the malicious adversarial model with less than $n/3$ corrupt parties.*

Proof. 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, atleast 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 $r \in_R \mathbb{Z}_k$. This completes the proof of correctness. The privacy and robustness of the protocol π_{RAND} follows directly from the fact that addition, comparison and private modulo reduction operations are provided by the underlying linear secret sharing scheme. □

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

Protocol 6 *random_neighbour*() π_{RN}

```
Input: $[A], [cur]$
Output: $[u]$
1: $[u] \leftarrow [0]$
2: for i=1 to $|V|$ do
3:     $[temp] \leftarrow [i = cur]$
4:     for j=1 to $|V|$ do
5:         Party $P_k$ distributes $r_k \in_R \mathbb{Z}_p$, for all $1 \leq k \leq n$
6:         $[r_j] \leftarrow \sum_{k=1}^{n} [r_k]$
7:     $[min] \leftarrow [p-1]$
8:     for j=1 to $|V|$ do
9:         $[check] \leftarrow [temp] * [r_j < min] * [a_{ij} = 1]$
10:        $[min] \leftarrow [check] * [r_j] + ([1] - [check]) * [min]$
11:        $[u] \leftarrow [u] * ([1] - [check]) + [j] * [check]$
```

Lemma 4. *The protocol π_{RN} securely implements $\mathcal{F}_{RN}$ with statistical security in the malicious adversarial model with less than $n/3$ corrupt parties.*

Proof. 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. □

Protocol 7 *page_rank()* π_{PR}

```
Input: $ITER$, $\alpha$
 1: for i = 1 to $|V|$ do
 2:     $[count_i] \leftarrow [0]$
 3: $[A] \leftarrow construct\_adjacency\_matrix()$
 4: $[A] \leftarrow force\_boolean([A])$
 5: $[A] \leftarrow no\_zero\_outdegree([A])$
 6: $[r] \leftarrow random\_number([|V|])$
 7: $[cur] \leftarrow [r]$
 8: while $ITER > 0$ do
 9:     for i = 1 to $|V|$ do
10:         $[count_i] \leftarrow [count_i] + [cur = i]$
11:     for i = 1 to $n$ do
12:         $P_i$ distributes $[r_i]$, where $r_i \in_R \mathbf{Z}_p$
13:     $[r'] \leftarrow \sum_{i=1}^{n} [r_i]$
14:     $[flag] \leftarrow [r' < \alpha]$
15:     $[u] \leftarrow random\_neighbor([A], [cur])$
16:     $[v] \leftarrow [u] * [flag]$
17:     $[u] \leftarrow random\_number(|V|)$
18:     $[v] \leftarrow [v] + [u] * ([1] - [flag])$
19:     $[cur] \leftarrow [v]$
20:     $ITER = ITER - 1$
21: Output construction
```

Next we present a secure implementation of the PageRank computation algorithm. The algorithm inputs $ITER$ and α, where $ITER$ represents the length of the random walk we take on the underlying network and α is a parameter for deciding the teleportation probability in the random surfer model. 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 3-5, we construct an unweighted directed adjacency matrix such that no node has out-degree zero. In steps 6-7 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 in step 8, 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 π_{PR} uses $\Theta(ITER * n * |V|^2)$ additions, $\Theta(ITER * |V|^2)$ multiplications, $\Theta(ITER * |V|^2)$ comparison/equality checks and $\Theta(ITER)$ private modulo reduction operations.

Theorem 4. *The protocol π_{PR} securely implements $\mathcal{F}_{PR}$ with statistical security in the malicious adversarial model with less than $n/3$ corrupt parties.*

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

5.5 K-shell Decomposition

Core-periphery structure is one of the most prominent and well studied meso-scale 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 [54].

In the year 2003, Batagelj and Zaversnik [55] proposed 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 [55]. 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 step 5, we construct the adjacency matrix A. In steps 6-7, we ensure that the constructed adjacency matrix is unweighted and contains no self-loops. Further, in steps 8-10 we check if the underlying network of A is symmetric, if not the case, we abort the protocol. In each iteration of the for loop given in step 11, we securely assigns a node with its shell number. In steps 14-17, we find the least degree node u in the graph, which is to be pruned next. The value of *cur_shell* is updated in steps 18 to the maximum of *cur_shell* and deg_u. In steps 19-21, 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 22-25, we update the adjacency matrix under consideration by removing the node u and all its adjacent edges from the network. We do not precisely define the output construction step since the required output and the receivers of the output will be application specific. The proposed secure K-shell algorithm uses $\Theta(|V|^3)$ addition operations, $\Theta(|V|^3)$ multiplication operations and $\Theta(|V|^3)$ comparison/equality operations.

Protocol 8 *kshell_decomposition*() π_{KD}

```
1: for i = 1 to $|V|$ do
2:     $[mark_i] \leftarrow 0$
3:     $[shell_i] \leftarrow 0$
4: $[cur\_shell] \leftarrow 0$
5: $[A] \leftarrow construct\_adjacency\_matrix()$
6: $[A] \leftarrow force\_boolean([A])$
7: $[A] \leftarrow no\_sel\_loops([A])$
8: $check \leftarrow check\_symmetricity([A])$
9: if $check == 0$ then
10:     abort the protocol
11: for iter = 1 to $|V|$ do
12:     $[deg_u] \leftarrow |V|$
13:     $[u] \leftarrow 0$
14:     for i = 1 to $|V|$ do
15:         $[deg_i] \leftarrow \sum_{j=1}^{|V|} [a_{ij}]$
16:         $[u] \leftarrow [i][mark_i = 0][deg_i < deg_u] + [u]([1] - [mark_i = 0][deg_i < deg_u])$
17:         $[deg_u] \leftarrow [deg_i][i = u] + [deg_u]([1] - [i = u])$
18:     $[cur\_shell] \leftarrow [cur\_shell][cur\_shell > deg_u] + [deg_u]([1] - [cur\_shell > deg_u])$
19:     for i= 1 to $|V|$ do
20:         $[shell_i] \leftarrow [cur\_shell][i = u] + [shell_i]$
21:         $[mark_i] \leftarrow [i = u] + [mark_i]$
22:     for  i = 1 to $|V|$ do
23:         for j = 1 to $|V|$ do
24:             $[a_{ij}] \leftarrow [0][i = u] + [a_{ij}]([1] - [i = u])$
25:             $[a_{ji}] \leftarrow [0][i = u] + [a_{ji}]([1] - [i = u])$
26: Output construction
```

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 14, the variable u contains the index of the least degree unmarked vertex in the graph represented by the adjacency matrix A.*

Let $u^{(k)}$ represent the least degree unmarked vertex selected after the for loop on step 14 in the k^{th} iteration of the for loop on step 11.

Lemma 6. *In the k^{th} iteration of the for loop in step 11 of the protocol π_{KD}, the variable $shell_{u^{(k)}}$ is updated with correct shell number of vertex $u^{(k)}$.*

Proof. 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.

Theorem 5. *The protocol π_{KD} securely implements $\mathcal{F}_{KD}$ with statistical security in the malicious adversarial model with less than $n/3$ corrupt parties.*

Proof. 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. Using a VSS scheme supporting the above mentioned operations ensure the privacy and robustness of the protocol. □

6 Discretionary Sub-Protocols

This section contains implementation details of the sub-protocols, which were assumed in most of the network parameter protocols (section 5). These set of common functionalities are termed as *discretionary sub-protocols.* These include the protocols $adjacency\_matrix\_construction()$ for constructing the adjacency matrix, $force\_boolean()$ for converting the input adjacency matrix into a symmetric adjacency matrix, $remove\_self\_loops()$ for removing self-loops of the input adjacency matrix and $ranking()$ for allocating ranks to all the vertices of the graph on the basis of a network parameter.

6.1 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 [32], 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 [30]. 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. However, for convenience of notation, we use the *adjacency_matrix_construction*() sub-protocol to denote an invocation to any of the four variations. 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 [56]. 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

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.

Protocol 9 *weighted_external_adjacency_construction*() $\pi_{adj}^{(1)}$

```
Output: [A]
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:             $P_i$ distributes $a_{jk}^{(i)}$
6:             $[min] \leftarrow [min] * [min < a_{jk}^{(i)}] + [a_{jk}^{(i)}] * ([1] - [min < a_{jk}^{(i)}])$
7:         $[a_{jk}] \leftarrow [min]$
```

2. Weighted adjacency matrix construction by internal parties: In this protocol each party shares her adjacency vector entries using a verifiable secret sharing scheme.

Protocol 10 *weighted_internal_adjacency_construction*() $\pi_{adj}^{(2)}$

```
Output: [A]
1: for i = 1 to $|V|$ do
2:     for j = 1 to $|V|$ do
3:         Party $P_i$ distributes $a_{ij}$
```

3. Unweighted adjacency matrix construction by external parties: This protocol uses $\Theta(|V|^3)$ addition operations and $\Theta(|V|^3)$ multiplication operations.

Protocol 11 *unweighted_external_adjacency_construction*() $\pi^{(3)}_{adj}$

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

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 $\pi^{(4)}_{adj}$ is the same as the π_{BOOL} or *force_boolean*() protocol i.e. $\Theta(|V|^2)$ addition operations and $\Theta(|V|^2)$ equality checks.

Protocol 12 unweighted_internal_adjacency_construction $\pi^{(4)}_{adj}$

```
Output: [A]
1: $[A] \leftarrow$ weighted_internal_adjacency_construction()
2: $[A] \leftarrow$ force_boolean([A])
```

6.2 Additional Security Sub-protocols

While constructing the unweighted adjacency matrix, an additional sub-protocol to force the entries to be Boolean valued was used. There are several other applications which demand that the underlying adjacency matrix satisfies similar constraints. These restrictions on the entries of the adjacency matrix are imposed in order to curb malicious parties from learning additional information and to ensure correctness of the protocol. For example, consider a network parameter computation protocol π, that expects as inputs a shared symmetric adjacency matrix. Hence, there is a need for a sub-protocol which performs a symmetricity check on the constructed adjacency matrix. In this section, we present a set of sub-protocols that can be used to enforce additional constraints on the network.

Check for adjacency matrix to be Symmetric
Many of the proposed network parameter protocols, such as Clustering coefficient and Kshell decomposition algorithm, are designed to handle only symmetric

graphs. Hence, there is a need to design a sub-protocol to test the symmetricity of the shared adjacency matrix.

The proposed protocol inputs a distributedly held graph $[A]$. 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.

Protocol 13 *check_symmetricity*() π_{SYM}

```
Input: $[A]$
Output: $res$
1: $[temp] \leftarrow [0]$
2: for i = 1 to $|V| - 1$ do
3:     for j = (i+1) to $|V|$ do
4:         $[temp] \leftarrow [temp] + [1] - [a_{ij} = a_{ji}]$
5: $[res] \leftarrow [temp = 0]$
6: $res \leftarrow [res]$
```

Enforcing the adjacency matrix to represent an unweighted graph

Clustering coefficient computation protocol and Google PageRank algorithm use the functionality $force\_boolean()$ to ensure that the graph under consideration is an unweighted graph. In this section, we provide an implementation of this protocol.

The $force\_boolean()$ protocol takes as input $[A]$ i.e. a distributedly held adjacency matrix. The protocol checks if the values of the adjacency matrix are 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.

Protocol 14 $force\_boolean()$ π_{BOOL}

```
Input: $[A]$
Output: $[A]$
1: for i = 1 to $|V|$ do
2:     for j = 1 to $|V|$ do
3:         $[a_{ij}] \leftarrow [1] - [a_{ij} = 0]$
```

Remove Self Loops

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 a distributedly held adjacency matrix $[A]$ and outputs the modified adjacency matrix.

Protocol 15 $remove\_self\_loops()$ π_{SL}

```
Input: $[A]$
Output: $[A]$
1: for i = 1 to $|V|$ do
2:     $[a_{ii}] \leftarrow 0$
3: return $[A]$
```

6.3 Ranking

The protocols proposed in section 5 focus on revealing the centrality measure values of nodes. But many a times, network analysts desire the ranking of the nodes with respect to some centrality measure, and not the exact centrality values. In this section we propose *ranking*() protocol, which inputs the network parameter values of all the nodes in the network, and it outputs the rank of all the nodes. Nodes with the same centrality value receive the same rank. This protocol uses $\Theta(|V|^2)$ addition operations and $\Theta(|V|^2)$ comparison operations.

Protocol 16 $ranking()$ π_R

```
Input: $[value_i]$, $\forall i = 1, 2, \dots |V|$
Output: $[rank_i]$, $\forall i = 1, 2, \dots |V|$
1: for i = 1 to $|V|$ do
2:     $[rank_i] \leftarrow [0]$
3: for i = 1 to $|V|$ do
4:     for j = 1 to $|V|$ do
5:         $[rank_i] \leftarrow [rank_i] + [value_i < value_j]$
```

7 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. Degree distribution and clustering coefficient are parameters that are characteristic of all real world networks. The closeness centrality, PageRank and K-shell algorithm capture the influential or central nodes in the network. To further build on this idea, one can securely implement other network parameters like reciprocity, homophiliy, 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 bring to light the vast unexplored mine 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. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. Journal of the ACM (JACM) **38**(3) (1991) 690–728
22. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the nineteenth annual ACM symposium on Theory of computing, ACM (1987) 218–229
23. Yao, A.: How to generate and exchange secrets. In: Foundations of Computer Science, 1986., 27th Annual Symposium on, IEEE (1986) 162–167
24. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: Proceedings of the twentieth annual ACM symposium on Theory of computing, ACM (1988) 11–19
25. 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
26. Brickell, J., Shmatikov, V.: Privacy-preserving graph algorithms in the semi-honest model. In: Advances in Cryptology-ASIACRYPT 2005. Springer (2005) 236–252
27. 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
28. Aly, A., Van Vyve, M.: Securely solving classical network flow problems. In: Information Security and Cryptology-ICISC 2014. Springer (2014) 205–221
29. 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
30. 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
31. 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
32. 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
33. 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)
34. Tassa, T., Bonchi, F.: Privacy preserving estimation of social influence. In: EDBT. (2014) 559–570
35. Cramer, R., Damgard, I., Nielsen, J.B.: Secure multiparty computation and secret sharing-an information theoretic appoach. Book Draft (2012)

36. Shamir, A.: How to share a secret. Communications of the ACM **22**(11) (1979) 612–613
37. Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.: Verifiable secret sharing and achieving simultaneity in the presence of faults. In: Foundations of Computer Science, 1985., 26th Annual Symposium on, IEEE (1985) 383–395
38. 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
39. Strogatz, S.H.: Exploring complex networks. Nature **410**(6825) (2001) 268–276
40. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of modern physics **74**(1) (2002) 47
41. Dorogovtsev, S.N., Mendes, J.F.: Evolution of networks. Advances in physics **51**(4) (2002) 1079–1187
42. Adamic, L.A., Huberman, B.A.: Power-law distribution of the world wide web. Science **287**(5461) (2000) 2115–2115
43. 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
44. 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
45. Georg, S.: Soziologie. Leipzig: Drunker und Humbolt (1908)
46. Watts, D.J., Strogatz, S.H.: Collective dynamics of small-worldnetworks. nature **393**(6684) (1998) 440–442
47. Bavelas, A.: Communication patterns in task-oriented groups. Journal of the acoustical society of America (1950)
48. Wasserman, S., Faust, K.: Social network analysis: Methods and applications. Volume 8. Cambridge university press (1994)
49. Hsieh, C.S., Kovářík, J., Logan, T.: How central are clients in sexual networks created by commercial sex? Scientific reports **4** (2014)
50. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: bringing order to the web. (1999)
51. Brin, S., Page, L.: Reprint of: The anatomy of a large-scale hypertextual web search engine. Computer networks **56**(18) (2012) 3825–3833
52. Franceschet, M.: Pagerank: Standing on the shoulders of giants. Communications of the ACM **54**(6) (2011) 92–101
53. Fridgen, G., Zare Garizy, T.: Supply chain network risk analysis-a privacy preserving approach. In: Proceedings of the International Conference on Information Systems, Münster, Germany, Mai 2015. (2015)
54. 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
55. Batagelj, V., Zaversnik, M.: An o (m) algorithm for cores decomposition of networks. arXiv preprint cs/0310049 (2003)
56. Kissner, L., Song, D.: Privacy-preserving set operations. In: Advances in Cryptology–CRYPTO 2005, Springer (2005) 241–257