Secure Centrality Computation Over Multiple Networks

Gilad Asharov
Cornell-Tech
New York
asharov@cornell.edu

Francesco Bonchi
ISI Foundation
Turin, Italy
francesco.bonchi@isi.it

David García Soriano
Eurecat & Pompeu Fabra University
Barcelona, Spain
david.garcia@eurecat.org

Tamir Tassa
The Open University
Ra'anana, Israel
tamirta@openu.ac.il

ABSTRACT

Consider a multilayer graph, where the different layers correspond to different proprietary social networks on the same ground set of users. Suppose that the owners of the different networks (called hosts) are mutually non-trusting parties: how can they compute a centrality score for each of the users using all the layers, but without disclosing information about their private graphs?

Under this setting we study a suite of three centrality measures whose algebraic structure allows performing that computation with provable security and efficiency. The first measure counts the nodes reachable from a node within a given radius. The second measure extends the first one by counting the number of paths between any two nodes. The final one is a generalization to the multilayer graph case: not only the number of paths is counted, but also the multiplicity of these paths in the different layers is considered.

We devise a suite of multiparty protocols to compute those centrality measures, which are all provably secure in the information-theoretic sense. One typical challenge and limitation of secure multiparty computation protocols is their scalability. We tackle this problem and devise a protocol which is highly scalable and still provably secure. We test our protocols on several real-world multilayer graphs: interestingly, the protocol to compute the most sensitive measure (i.e., the multilayer centrality) is also the most scalable one and can be efficiently run on very large networks.

Keywords

Social Networks; Centrality Measures; Multilayer Graphs; Secure Multiparty Protocols

1. INTRODUCTION

Nowadays social networking platforms, such as Facebook or Twitter, have realized that their proprietary social graph is an important asset with inestimable value, thus they keep it secret for obvious reasons of commercial benefits, as well as due to privacy legislation.[1,2] At the same time, users are more and more allowed to log in one platform using another platform's account (e.g., log in Tripadvisor using a Facebook account) or by explicitly connecting their accounts in different networks[3]. This flexibility gives rise to a situation in which multiple social networks are defined over the same set of users (nodes), but the connections among these users (links) are secret information of the different social networking platforms' owners (which we call in the following *hosts*).

Multilayer graphs (sometimes called *multi-dimensional networks* or *multiplex networks*) are graphs that are composed of several layers, where each layer is a graph on the same set of nodes. Recently, the problem of computing node centrality measures in multilayer graphs has received a lot of attention [17, 9, 20, 31]. All of these studies assume a centralized setting, where all the layers are accessible at once. Contrarily to the literature, in this work we consider the problem of computing centrality measures in *distributed* multilayer graphs, i.e., where each layer in the graph is held by a different host. We assume that the hosts are mutually distrustful and do not wish to reveal any sensitive information about their perspective layer. In particular, we study three simple – yet meaningful – measures of centrality, one progressively generalizing the previous, which enable efficient and provably secure computation.

The first measure counts the number of nodes reachable from a node within a given radius D: this is a straightforward generalization of the most trivial definition of centrality, which is the *degree* (corresponding to the case $D = 1$).

The second measure extends the first one by counting the number of paths between any two nodes: this is the truncated version of the classic index introduced by the sociologist Leo Katz in 1953 [19], which, according to a recent axiomatization by Boldi and Vigna [4], is one of the centrality definitions exhibiting most of the desirable properties of centrality measures. The Katz index has been widely applied in link prediction (where it is known to be one of the best performing predictors [22]), anomalous link detection [28], and recommendation [29]. It is also worth mentioning that the Katz index is a close progenitor of PageRank [5]: the two measures differ only by a constant factor and by the ℓ_1 normalization of the adjacency matrix in PageRank [4].

While the first two measures are relevant for both a normal (i.e. single-layered) graph and a multilayer graph, the

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052602

[1] http://techcrunch.com/2013/01/24/my-precious-social-graph/
[2] http://mashable.com/2012/07/27/twitter-instagram-find-friends/
[3] http://domain.me/connect-facebook-and-twitter/

third measure extends the Katz index to the multilayer setting, by keeping in consideration the fact that one node might be reachable from another node by different paths and in different layers. For example, if Bob is a friend of Alice in Facebook, Twitter and Linkedin, the chances of Bob being influenced by Alice in doing some action are, arguably, greater than Carol's, if Carol is a friend of Alice only in Facebook. Based on this intuition we introduce our third centrality measure, called *Multilayer Truncated Katz*.

The main challenges. The problem we study is significantly more challenging than the centralized case, as no algorithm can freely access all the layers at once. *Secure multiparty computation*, introduced by Yao in 1982 [36], provides general solutions for problems that are similar in nature to ours: mutually distrustful parties wish to compute some joint function on their inputs without revealing them to one another. There are three main different approaches for this problem: garbled circuits [37, 25], boolean sharing [14, 13], and arithmetic sharing [3, 8, 1]. Unfortunately, all of these general solutions do not cope easily with the volume of graphs we are interested in, as these protocols require the transmission of a large amount of data and heavy processing for the participating parties. Despite the latest important efficiency improvements in the area (see [27, 2, 10], to state a few), designing the right protocol for a given task is highly non-trivial, as it requires some subtle algorithmic design, and as it depends on many parameters that should be considered and on the exact application scenario (such as the properties of the underlying network, the computational resources of the parties, etc.).

The protocols that we introduce in this paper are based on the protocol of Ben-Or, Goldwasser and Wigderson [3] (BGW). The BGW protocol allows n parties, P_i, $1 \leq i \leq n$, each holding a private integer x_i in some finite field $\mathbb{F}$, to jointly compute a multi-polynomial $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ over those private inputs, without disclosing to each other their private inputs. The main challenge in this context is to cast the needed centrality scores as polynomials, over the private inputs of the different hosts, with as minimal degree as possible, as the degree is tightly connected to the running time of the protocol. This is non-trivial, since even a simple operation such as an IF-statement is translated to a polynomial with relatively high degree (specifically, the size of the underlying finite field, which in our setting is larger than the number of nodes). devise two protocols, the second of which is highly scalable and can be efficiently run on very large networks. We obtain this significant scalability boost by modifying the output of the protocol. Specifically, all of our centrality scores depend on a radius parameter D; while our basic protocol for the multilayer Katz score outputs only the D-centrality score, our scalable protocol outputs the d-centrality score for all $1 \leq d \leq D$. This slight modification of the problem definition enables not only the scalability boost (which makes the difference between a theoretical protocol and a practical one) but also increased flexibility and independence for the collaborating hosts, as explained in Section 5.

Paper contributions and roadmap. The contributions of this paper are summarized as follows:

- We initiate investigation in the area of secure distributed computation of centrality measures over multiple, mutually non-trusting, social networks.
- Our main contribution is a suite of multiparty protocols to compute three different measures of centrality of increasing complexity. Our protocols are provably secure in the information-theoretic sense.
- The third of the measures of centrality that we consider is the most sensitive one, as it takes into consideration the multiplicity of paths in the multilayer graph. For that measure we devise a scalable protocol and empirically show that it can be efficiently run on very large real-world multilayer graphs.

To the best of our knowledge *ours is the first study that deals with problems of computing centrality scores over multiple, mutually non-trusting, social networks.*

The paper is organized as follows. The next section discusses related work. Section 3 introduces the three measures of centrality and provides the formal problem statement and some background in Secure Multiparty Computation needed to understand the protocols, which are given in Section 4. Scalability is discussed in Section 5, while Section 6 reports our empirical assessment, and Section 7 concludes the paper.

2. RELATED WORK

Several authors have proposed definitions of centrality measures in multigraphs [17, 9, 20, 31]. However, none of these previous articles deal with the distributed setting in which each layer in the graph is owned by a different party.

Secure protocols have been shown to be useful in a wide-variety of applications. Protocols for secure computation were studied in the area of *privacy-preserving data mining* where the data is distributed among several parties who aim to jointly perform data mining on the unified corpus of data, while protecting the data records of each of the data owners from the other parties. For example, Lindell and Pinkas [24] showed how to securely build an ID3 decision tree when the training set is distributed horizontally; Lin *et al.* [23] discussed secure clustering using the EM algorithm over horizontally distributed data; and the problem of distributed association rule mining was studied in [34, 38] in the vertical setting, and in [18, 33] in the horizontal setting.

3. PRELIMINARIES

We consider a setting with ℓ players (named *hosts*), $H_1, \ldots, H_\ell$, each one holding a private directed graph on the same set of nodes V. We do not tackle here the technical problem of *network reconciliation* [21], i.e., identifying which node corresponds to which node in different networks, as it is beyond the scope of this paper. We assume that the problem is solved and we know the identity of the same users across different networks: this is more and more the case as nowadays many users explicitly connect their accounts in different networks. A directed edge (or *an arc* for short) (u, v) indicates the fact that v is a follower of u, i.e., v is notified about u's activities and thus u can influence v, or in other terms, information can propagate from u to v.[4]

The private graph of H_t, $1 \leq t \leq \ell$, is $G_t = (V, E_t)$ where E_t can be described by an adjacency matrix $A_t : V^2 \rightarrow \{0, 1\}$. The entry in A_t that corresponds to the ordered

[4]We adopt here the direction of links used in the information propagation and social influence literature. Sometimes the opposite direction is adopted: a directed edge (u, v) indicates the fact that u endorses v, as it is the case in, e.g., the WWW graph where web pages refer to other pages.

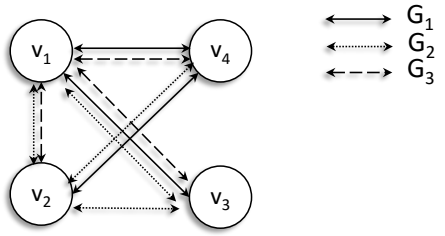

Figure 1: An example of a multilayer graph.

pair of nodes (u, v) is 1 if and only if E_t has an arc from u to v. Those private graphs induce the *unified graph* $G = (V, E)$ where $E = \bigcup_{t=1}^{\ell} E_t$. Its adjacency matrix is $A = \bigvee_{t=1}^{\ell} A_t$; namely, for every $(u, v) \in V^2$, $A(u, v)$ (the entry of A corresponding to the pair (u, v)) equals 1 if and only if there exists $1 \leq t \leq \ell$ such that $A_t(u, v) = 1$.

The private graphs G_t, $1 \leq t \leq \ell$, induce also a *multilayer graph* $\mathcal{G} = (V, \{E_1, \ldots, E_\ell\})$. While in G each ordered pair of nodes $(u, v) \in V^2$ is either connected by an arc or not, as indicated by $A(u, v)$, in $\mathcal{G}$ such a pair is connected by any number of arcs between 0 and ℓ. The number of connecting arcs for any ordered pair of nodes is given by the entries of the matrix $B = \sum_{t=1}^{\ell} A_t$.

Definition 1 (Path and path scenario) *Consider the collection of private graphs $G_1, \ldots, G_\ell$ and the corresponding unified graph G and multigraph $\mathcal{G}$. If $(u, v) \in V^2$ then a (directed) path from u to v is a sequence of nodes $\langle w_0 = u, w_1, \ldots, w_{s-1}, w_s = v\rangle$ such that $(w_{i-1}, w_i) \in E$ for all $1 \leq i \leq s$; in that case s is the length of the path. A path scenario is a path as defined above together with a sequence of indices $1 \leq t_1, \ldots, t_s \leq \ell$ such that $(w_{i-1}, w_i) \in E_{t_i}$ for all $1 \leq i \leq s$.*

Example 1 *Let $\mathcal{G} = (V, \{E_1, E_2, E_3\})$ be the multigraph induced by the three graphs G_1, G_2, G_3 depicted in Figure 1. An example of path scenario in $\mathcal{G}$ is $(\langle v_1, v_2, v_4\rangle, \langle 3, 2\rangle)$, indicating that one can go from v_1 to v_4 trough v_2 with an arc in E_3 and then with an arc in E_2.*

The following is a simple observation (we omit the proof for space economy).

Theorem 1 *For any integer $k \geq 1$, $A^k(u, v)$ equals the number of paths of length k from u to v in the unified graph G, while $B^k(u, v)$ equals the number of path scenarios of length k from u to v.*

Example 2 *Consider again the three graphs G_1, G_2, G_3 of Figure 1 and their adjacency matrices A_1, A_2, A_3:*

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Then the matrices A and B are:

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 2 & 3 & 2 \\ 2 & 0 & 1 & 2 \\ 3 & 1 & 0 & 0 \\ 2 & 2 & 0 & 0 \end{pmatrix}.$$

For example $A(1, 2) = 1$ indicates that there is an arc in G from v_1 to v_2, while $B(1, 2) = 2$, indicates that such arc exists in two of the private graphs. Next, we have

$$A^2 = \begin{pmatrix} 3 & 2 & 1 & 1 \\ 2 & 3 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \end{pmatrix}, \quad B^2 = \begin{pmatrix} 17 & 7 & 2 & 4 \\ 7 & 9 & 6 & 4 \\ 2 & 6 & 10 & 8 \\ 4 & 4 & 8 & 8 \end{pmatrix}.$$

Here, $A^2(1, 2) = 2$ indicates that G has two paths from v_1 to v_2 of length 2 (one through v_3 and one through v_4). However, $B^2(1, 2) = 7$ as there are 7 ways to realize those paths in $\mathcal{G}$.

Definition 2 *Given an integer D and k scalar weights $\beta_1 \geq \beta_2 \geq \cdots \geq \beta_k$, we define $A^{[D]} := \sum_{k=1}^{D} \beta_k A^k$ and $B^{[D]} := \sum_{k=1}^{D} \beta_k B^k$.*

When all weights $\beta_k = 1$, it is easy to see that $A^{[D]}(u, v)$ equals the number of paths of length at most D from u to v in G, while $B^{[D]}(u, v)$ equals the number of path scenarios of length at most D from u to v. The weights β_k are used to give more importance to shorter paths over longer ones.

We are now ready to define the three measures of centrality which are the main focus of our work. In the context of social networks they may be viewed as measures of influence or prestige. The measures are denoted for simplicity π_1, π_2, and π_3 in increasing order of complexity. The first measure counts the number of nodes that are reachable from u by a path of length at most D in the union graph G.

Definition 3 (Reach) *For a nonnegative integer x, let $\flat(x) = 0$ if $x = 0$ and $\flat(x) = 1$ if $x > 0$. Then the reach of a node $u \in V$ is*

$$\pi_1^D(u) = \sum_{v \in V} \flat(B^{[D]}(u, v)).$$

The reach of a node is a straightforward generalization of the most trivial definition of centrality, which is the *degree* of a node (corresponding to the case $D = 1$).

The next measure we consider is the classic index introduced by the sociologist Leo Katz in 1953 [19].

Definition 4 (Truncated Katz) *The Katz score of a node u truncated at D is*

$$\pi_2^D(u) = \sum_{v \in V} A^{[D]}(u, v).$$

In the classic Katz index $\beta_k = a^k$ for a constant a, and $D = \infty$: the infinite sum converges if $a < 1/\rho(A)$, where $\rho(A)$ is the spectral radius of A. When we consider a given $D < \infty$, the index is called *Truncated Katz* [11, 35, 26]. Truncated Katz can be seen as an extension of the *Reach* score: instead of just counting the reachable nodes, it gives each such node v a multiplicity weight that equals the number of different paths from u to v of length at most D in G, and it then applies the distance weights β_k.

The measures π_1 and π_2 are relevant for both a normal graph and a multigraph. The third measure, π_3, that we introduce next, is relevant only in a multigraph setting. It gives higher weights to nodes v that can be reached by u in more ways: e.g., if Bob is a friend of Alice in Facebook, Twitter and Linkedin, the chances of Bob being influenced by Alice in doing some action are, arguably, greater than Carol's, if Carol is a friend of Alice only in Facebook. Based on this intuition we introduce *Multi-layer Truncated Katz*.

Definition 5 (Multi-layer Truncated Katz) *The Katz score of a node u truncated at D over a multigraph $\mathcal{G}$ is*

$$\pi_3^D(u) = \sum_{v \in V} B^{[D]}(u, v).$$

Surprisingly, Multi-layer Truncated Katz is not only the most sensitive of the three measures, it is also the most efficient and scalable to compute. Before getting into the details, some background in secure multiparty computation is needed.

3.1 Secure multiparty computation

In the setting of secure multiparty computation (MPC) [36], n mutually distrustful parties, $P_1, \ldots, P_n$, that hold private inputs $x_1, \ldots, x_n$ wish to compute some joint function on their inputs, i.e., $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$, without revealing any unnecessary information about these inputs (except for what is logically learned from the output).

MPC enables the parties to compute the function using an interactive protocol, where each party P_i learns exactly y_i and nothing else. The security of the protocol is preserved even in the presence of an adversarial entity that corrupts some of the parties, combines their transcripts and tries to learn information about the honest parties' inputs. In this work, we consider a semi-honest adversary (who follows the protocol's specification but tries to learn more than it should by inspecting the protocol's transcript) that may have an unbounded computational power.

MPC provides a formal and rigorous definition of security, where the standard definition follows the "simulation paradigm", and it is similar in its nature to the definition of semantic security of encryptions [15] and zero-knowledge proofs [16]. In order to prove that a protocol is secure in the presence of a semi-honest adversary, one needs to show the existence of a simulator that receives as input the inputs and outputs of the corrupted parties, and simulates the messages that the corrupted parties see during the execution of the protocol. The existence of such a simulator implies that all the messages that the corrupted parties see during the interaction can be computed solely from their own input and output. Hence, the messages that the honest parties send during the protocol do not provide any other meaningful information. The formal definitions are more subtle, and we refer to [7, 13, 1] for formal definitions. We emphasize that even though secure protocols are very powerful, they cannot hide information that is logically leaked from the function that the parties compute. (For example, in case where three parties compute, say, the average of their salaries, the result reveals to each party the average of the other two.)

3.2 The BGW protocol

Our protocols, described in Section 4, are inspired by the BGW protocol [3] (the security of which was proven in [1]), with the efficiency improvement of [12]. Consider n parties, P_i, $1 \leq i \leq n$, each holding a private element x_i in some finite field $\mathbb{F}$. They wish to jointly compute a function over those inputs, $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, where y_i is the output to P_i, without disclosing to each other their private input. To that end, the parties $P_1, \ldots, P_n$ agree on some arithmetic circuit C that computes f over the finite field $\mathbb{F}$; the field's size must be greater than the number of participating parties as well as greater than the apriori bound on the input and output values. The circuit consists of two different types of gates: addition gates, and multiplication gates. Let $\alpha_1, \ldots, \alpha_n$ be distinct non-zero elements in $\mathbb{F}$; then α_i will be used as a public identifier of P_i. The parties preserve the following invariant during the computation: the value of each wire of the circuit is secret-shared using a Shamir's $(t+1)$-out-of-n secret sharing scheme (see [30]), with $t < n/2$. The protocol consists of the following three stages: input sharing phase, circuit emulation phase, and output reconstruction phase.

The input sharing phase. In this phase, each party P_i shares its input x_i with all parties; i.e., it chooses a random polynomial g_i of degree t such that $g_i(0) = x_i$, and it then sends to each party P_j the value $g_i(\alpha_j)$, $1 \leq j \leq n$.

The circuit emulation phase. In this phase, the parties emulate the computation of $C(x_1, \ldots, x_n)$, where in each gate, the parties compute shares of the value of the output wire using their shares of the input wire by invoking a secure protocol. There are two types of gates to consider: addition gates and multiplication gates.

Addition gates. The computation of the output shares can be performed locally and without any interaction, since if $f_1(\alpha_i)$ and $f_2(\alpha_i)$ are the shares that P_i holds for the two input wires to an addition gate, then $f(\alpha_i) = f_1(\alpha_i) + f_2(\alpha_i)$ is a valid sharing of the output wire. Indeed, if the polynomials $f_1(x)$ and $f_2(x)$ are of degree at most t, then so is their sum $f(x) := f_1(x) + f_2(x)$; and, in addition, $f(0) = f_1(0) + f_2(0)$.

Multiplication gates. The case of multiplication gates is more involved as it requires interaction among the parties. In particular, given shares $f_1(\alpha_i)$ and $f_2(\alpha_i)$ for the two input wires of a multiplication gate, then $f(\alpha_i) := f_1(\alpha) \cdot f_2(\alpha_i)$ are shares of a polynomial $f(x)$ with the correct constant term $f(0) = f_1(0) \cdot f_2(0)$, as required, but its degree could be as high as $2t$, while we need the sharing polynomial to be of degree at most t. Hence, the players must interact in order to reduce the degree of that polynomial. The degree reduction procedure can be done using the method of [12], which is based on the fact that if f is a polynomial of degree at most $n-1$ and $\alpha_1, \ldots, \alpha_n$ are n distinct non-zero points in the field, then the constant term $f(0)$ is a linear combination of the other points on that polynomial. That is, $f_1(0) \cdot f_2(0) = f(0) = \sum_{i=1}^{n} \lambda_i \cdot f(\alpha_i)$, where $\lambda_i := \prod_{j \neq i} \alpha_j / (\alpha_i - \alpha_j)$ are the Lagrange coefficients.

The multiplication sub-protocol proceeds as follows. Given the shares $f_1(\alpha_i), f_2(\alpha_i)$ of the party P_i, the party P_i locally multiplies these two shares and gets the value $f(\alpha_i)$. Then, it chooses a polynomial $g_i(x)$ of degree at most t such that $g_i(0) = f(\alpha_i) = f_1(\alpha_i) \cdot f_2(\alpha_i)$. It then shares the polynomial g_i with all parties, so that each party P_j receives the share $g_i(\alpha_j)$. At the end of this stage, each party P_j holds the shares $g_1(\alpha_j), \ldots, g_n(\alpha_j)$. Next, let us define the polynomial $h(x) := \sum_{i=1}^{n} \lambda_i \cdot g_i(x)$, which has a degree at most t. Each party P_j locally computes the linear combination $\sum_{i=1}^{n} \lambda_i \cdot g_i(\alpha_j) = h(\alpha_j)$, which is its share in the implicitly defined polynomial $h(x)$. Note that $h(x)$ is a polynomial of degree at most t, and $h(0) = \sum_{i=1}^{n} \lambda_i \cdot g_i(0) = \sum_{i=1}^{n} \lambda_i \cdot f(\alpha_i) = f_1(0) \cdot f_2(0)$. We later relate to this multiplication procedure as "the multiplication protocol of BGW".

Output reconstruction phase. In this phase, each party P_i receives all the shares of the output wire that corresponds to its respective output y_i, and it then reconstructs y_i.

4. THE PROTOCOLS

All of our protocols are based on sharing values among the ℓ hosts using Shamir's ℓ'-out-of-ℓ secret sharing scheme. If x is any integer that the ℓ hosts share among themselves,

then $[x]$ denotes the set of all ℓ shares in x and $[x]_t$ denotes H_t's share, $1 \le t \le \ell$. The parameters of the scheme must satisfy $\ell' < \ell/2$, where ℓ' is the number of corruptions that our protocols can deal with. We note that this bound is tight for protocols that are perfectly secure [3].

Our protocols use the following building blocks as sub-protocols. The F_{Bool} sub-protocol, defined below, is used in implementing the operator $\flat$ (see Definition 3) in a secure manner. The F_{OR} sub-protocol is used when computing shares of the adjacency matrix A of the unified graph.

The F_{Bool} sub-protocol. Assume that the hosts hold shares $[x]$ in some nonnegative integer x using an ℓ'-out-of-ℓ secret sharing scheme. The function F_{Bool} enables them to translate those shares into shares in the corresponding boolean value $\flat(x)$. Formally, $F_{Bool}(x_1, \ldots, x_\ell) \to (y_1, \ldots, y_\ell)$, where $(x_1, \ldots, x_\ell)$ are ℓ'-out-of-ℓ shares of a secret x, and $(y_1, \ldots, y_\ell)$ are fresh ℓ'-out-of-ℓ shares of $y = \flat(x)$ (i.e., $y = 1$ if $x > 0$ and $y = 0$ if $x = 0$). Note that the definition of this function does not leak to a set of less than ℓ' parties any information about the value of x nor that of y.

The classical way to realize this function (e.g. [6]) is based on Fermat's little theorem, by which $x^{p-1} = \flat(x)$ in the prime-order field $\mathbb{Z}_p$. Therefore, in order to compute this function, the hosts can just raise the underlying secret of their input shares to the power $p-1$. By using the square-and-multiply method (see e.g. [32]), this results in a circuit of depth $O(\log p)$, which can be evaluated using the BGW protocol.

The F_{OR} sub-protocol. The OR function is defined by $F_{OR}(b_1, \ldots, b_\ell) \to (y_1, \ldots, y_\ell)$, where $b_1, \ldots, b_\ell$ are private bits (b_t is known only to H_t) and $(y_1, \ldots, y_\ell)$ are fresh ℓ'-out-of-ℓ shares of the bit $y := \vee_{i=1}^{\ell} b_i$, which is also kept secret. Since $y = 1 - \prod_{i=1}^{\ell}(1 - b_i)$, such a function can be implemented by the BGW protocol using only $\ell - 1$ multiplication gates, which can be parallelized and implemented in $O(\log \ell)$ layers.

We proceed to describe three protocols, Π_j, for computing π_j^D, $j = 1, 2, 3$. We begin with Π_3 which is the simplest one, as well as the most efficient one.

4.1 Protocol for Multigraph Truncated Katz

We present here Protocol Π_3 that enables the hosts to compute together $\pi_3^D(u)$ for all $u \in V$. By Definition 5, $\pi_3^D(u) = \sum_{v \in V} B^{[D]}(u, v)$, where $B^{[D]} := \sum_{k=1}^{D} \beta_k B^k$ and $B = \sum_{t=1}^{\ell} A_t$. We begin by making the following observation, which enables us to refrain from computing explicit shares in all $|V|^2$ entries of the matrix powers B^k, $2 \le k \le D$, nor those of $B^{[D]}$. Define

$$s_k(u) = \sum_{v \in V} B^k(u, v)\,. \tag{1}$$

Then the required centrality score $\pi_3^D(u)$ of a specific user $u \in V$ is given by

$$\pi_3^D(u) = \sum_{k=1}^{D} \beta_k s_k(u)\,. \tag{2}$$

We show below how the hosts can compute shares in the values $s_k(u)$ for all $k = 1, \ldots, D$ and $u \in V$. After computing such shares, each host can compute shares in $\pi_3^D(u)$ for all $u \in V$ using Eq. (2). Finally, by implementing the secret sharing reconstruction procedure, any subset of ℓ' hosts can combine their shares to recover $\pi_3^D(u)$ for any $u \in V$.

The protocol begins with every host sharing its adjacency matrix with all hosts (Step 1). Specifically, each host H_t, $1 \le t \le \ell$, performs $|V|^2$ independent ℓ'-out-of-ℓ secret sharings for each entry in its adjacency matrix $A_t(u, v)$. At the end of this step, each host H_t holds a share $[A_j(u, v)]_t$ for each $(u, v) \in V^2$ and $1 \le j \le \ell$. Then, each host H_t computes $[B(u, v)]_t = \sum_{j=1}^{\ell} [A_j(u, v)]_t$, which is its share in an ℓ'-out-of-ℓ secret sharing of $B(u, v)$, for all $(u, v) \in V^2$. To conclude the initialization phase, H_t computes, for every $u \in V$, the sum $[s_1(u)]_t = \sum_{v \in V} [B(u, v)]_t$, which is its share in $s_1(u)$ (Step 2).

Next, the hosts can proceed by induction to compute shares in $s_k(u)$, for all $u \in V$ and $k = 2, \ldots, D$ (Steps 3-6). It is easy to see that Eq. (1) implies that

$$s_k(u) = \sum_{v \in V} B(u, v) s_{k-1}(v)\,. \tag{3}$$

Hence, as each host holds shares in $B(u, v)$ (computed in Step 1) as well as in $s_{k-1}(v)$ (induction), they can compute shares in $B(u, v)s_{k-1}(v)$ by implementing multiplication gates in the BGW protocol (Section 3.2) for all $(u, v) \in V^2$. (Note that all $|V|^2$ multiplication gates can be implemented in parallel.) After doing that (Step 4), they may add the received shares for $B(u, v)s_{k-1}(v)$ over all $v \in V$ in order to get a corresponding share in $s_k(u)$, for all $u \in V$ (Step 5). After completing the loop over k, each host H_t holds a share $[s_k(u)]_t$ in $s_k(u)$ for all $1 \le k \le D$ and $u \in V$. Hence, each host may proceed to compute on its own a share in $\pi_3^D(u)$ for each $u \in V$ using Eq. (2), as a weighted sum of its shares in $s_k(u)$, for $1 \le k \le D$ (Step 7). Finally, the hosts broadcast their shares in $\pi_3^D(u)$ for all $u \in V$ and then each host can recover $\pi_3^D(u)$ for any $u \in V$ from any subset of ℓ' shares (Step 8).

Protocol Π_3: Computing $\pi_3^D(u)$ for all $u \in V$.

Sub-Protocols: The multiplication protocol of BGW.
Input: Each host H_t holds its association matrix A_t.
1: **Sharing of the matrix B.** Each host H_t shares $A_t(u, v)$, $\forall (u, v) \in V^2$, with $H_1, \ldots, H_\ell$. Then, H_t computes its share in $B(u, v)$, $[B(u, v)]_t = \sum_{j=1}^{\ell} [A_j(u, v)]_t$, $\forall (u, v) \in V^2$.
2: H_t computes $[s_1(u)]_t = \sum_{v \in V} [B(u, v)]_t$, $\forall u \in V$.
3: **for** $k = 2, \ldots, D$ **do**
4: Given the shares $[B(u, v)]_t$ and $[s_{k-1}(v)]_t$, the hosts invoke BGW multiplication protocol to compute shares of $[B(u, v) \cdot s_{k-1}(v)]_t$, $\forall (u, v) \in V^2$
5: H_t computes $[s_k(u)]_t = \sum_{v \in V} [B(u, v)s_{k-1}(v)]_t$, $\forall u \in V$.
6: **end for**
7: H_t computes $[\pi_3^D(u)]_t = \sum_{k=1}^{D} \beta_k [s_k(u)]_t$, $\forall u \in V$.
8: Each host H_t broadcasts $[\pi_3^D(u)]_t$, and all hosts reconstruct $\pi_3^D(u)$ for every $u \in V$.
Output: Each host outputs $\pi_3^D(u)$ for all $u \in V$.

4.2 Protocol for Truncated Katz

Here we present Protocol Π_2 for computing $\pi_2^D(u)$ for all $u \in V$. The protocol is similar to Protocol Π_3, except for the initialization phase, where here the parties share the matrix $A = \bigvee_{t=1}^{\ell} A_t$ (rather than sharing the matrix $B = \sum_{t=1}^{\ell} A_t$). This is done by invoking the F_{OR} sub-protocol independently and concurrently for all $|V|^2$ entries of the matrix. Protocol Π_2 now proceeds (Steps 2-8) just like the corresponding steps in Protocol Π_3, where the hosts use their computed shares in the matrix A (rather than B as in Π_3).

Protocol Π_2: Computing $\pi_2^D(u)$ for all $u \in V$.

Sub-Protocols: The multiplication protocol of BGW and the F_{OR} sub-protocol.

Input: Each host H_t holds its association matrix A_t.

1: **Sharing of the matrix** A**.** The parties invoke the F_{OR} sub-protocol for every $(u,v) \in V^2$, where in each invocation the input of the host H_t is $A_t(u,v)$ and its output is its share $[A(u,v)]_t$, i.e., its share in $A(u,v) = \bigvee_{t=1}^{\ell} A_t$.
2: H_t computes $[s_1(u)]_t = \sum_{v \in V}[A(u,v)]_t$, $\forall u \in V$.
3: **for** $k = 2, \ldots, D$ **do**
4: Given the shares $[A(u,v)]_t$ and $[s_{k-1}(v)]_t$, the hosts invoke BGW multiplication protocol to compute shares of $[A(u,v) \cdot s_{k-1}(v)]_t$, $\forall (u,v) \in V^2$.
5: H_t computes $[s_k(u)]_t = \sum_{v \in V}[A(u,v)s_{k-1}(v)]_t$, $\forall u \in V$.
6: **end for**
7: H_t computes $[\pi_2^D(u)]_t = \sum_{k=1}^{D} \beta_k [s_k(u)]_t$, $\forall u \in V$.
8: Each host H_t broadcasts $[\pi_2^D(u)]_t$, and all hosts reconstruct $\pi_2^D(u)$ for every $u \in V$.

Output: Each host outputs $\pi_2^D(u)$ for all $u \in V$.

4.3 Protocol for the Reach score

Here we present Protocol Π_1 for computing $\pi_1^D(u)$ for all $u \in V$. That protocol, as opposed to the previous two, requires the hosts to compute shares in all powers of the relevant matrix, B; that renders this protocol more costly in terms of both computation and communication.

The protocol begins by sharing the matrix B among all hosts (Step 1). Then, the hosts proceed to compute shares in the matrix powers $B^2, \ldots, B^D$ (Steps 2-5). Since $B^k(u,v) = \sum_{w \in V} B^{k-1}(u,w)B(w,v)$, the hosts invoke the multiplication protocol of BGW to compute shares in the product $B^{k-1}(u,w)B(w,v)$ for all $(u,w,v) \in V^3$. After concluding the execution of those $|V|^3$ multiplications (which can be executed in parallel), each host H_t has a share $[B^{k-1}(u,w)B(w,v)]_t$ for all $(u,w,v) \in V^3$. It then proceeds to compute shares in $B^k(u,v)$ as follows: $[B^k(u,v)]_t = \sum_{w \in V}[B^{k-1}(u,w)B(w,v)]_t$.

Then, each host computes a share in each entry of the matrix $B^{[D]}$ (Step 6), and then the shares of $\flat(B^{[D]}(u,v))$ by invoking the F_{Bool} sub-protocol for all $(u,v) \in V^2$ in parallel (Step 7). Finally, each host computes a share in $\pi_1^D(u)$ for every $u \in V$ (Step 8), and then the hosts reconstruct $\pi_1^D(u)$ (Step 9).

Table 1: The communication complexity of our three protocols. Inp, Emu, Rec relate to the input sharing phase, circuit emulation phase and output reconstruction phase, respectively. $||\mathbb{F}||$ denotes the number of bits required to represent an element in the underlying field (which is $O(\log \ell)$ for Π_1, and $O(D \log(|V|\ell))$ for Π_2 and Π_3). The communication column shows the total number of field elements that are transmitted by all hosts.

Protocol	Stage	Rounds	Communication
Π_3	Inp	1	$\ell(\ell-1)\|V\|^2$
	Emu	$D-1$	$(D-1)\ell(\ell-1)\|V\|^2$
	Rec	1	$\ell(\ell-1)\|V\|$
Π_2	Inp	$1+\lceil \log \ell \rceil$	$\ell^2(\ell-1)\|V\|^2$
	Emu	D	$(D-1)\ell(\ell-1)\|V\|^2$
	Rec	1	$\ell(\ell-1)\|V\|$
Π_1	Inp	1	$\ell(\ell-1)\|V\|^2$
	Emu	$D-1$	$(D-1)\ell(\ell-1)\|V\|^3$
	Rec	$\|\|\mathbb{F}\|\|+1$	$\ell(\ell-1)(\|\|\mathbb{F}\|\|+\|V\|)$

Communication and round complexity. Table 1 summarizes the communication and round complexity of our protocols. Communication complexity measures the total number of field elements that are transmitted in the protocol by all hosts. Since our protocols are symmetric (in the sense that each host performs the exact same amount of work), the communication complexity of each host is the value in the table divided by ℓ. We also recall that in our scenario, both ℓ and D are relatively small (typically no more than 10) while $|V|$ can be several millions.

Protocol Π_1: Computing $\pi_1^D(u)$ for all $u \in V$.

Sub-Protocols: The multiplication protocol of BGW and the F_{Bool} sub-protocol.

Input: Each host H_t holds its association matrix A_t.

1: **Sharing of the matrix** B**.** Each host H_t shares $A_t(u,v)$, $\forall (u,v) \in V^2$, with $H_1, \ldots, H_\ell$. Then, H_t computes its share in $B(u,v)$, $[B(u,v)]_t = \sum_{j=1}^{\ell}[A_j(u,v)]_t$, $\forall (u,v) \in V^2$.
2: **for** $k = 2, \ldots, D$ **do**
3: Given the shares $[B^{k-1}(u,v)]_t$ and the shares $[B(u,v)]$, the hosts invoke the BGW multiplication protocol to receive shares of $[B^{k-1}(u,w) \cdot B(w,v)]_t$ for every $(u,v,w) \in V^3$.
4: For every $(u,v) \in V^2$, each host locally computes $[B^k(u,v)]_t = \sum_{w \in V}[B^{k-1}(u,w) \cdot B(w,v)]$,
5: **end for**
6: H_t computes $[B^{[D]}(u,v)]_t = \sum_{k=1}^{D} \beta_k [B^k(u,v)]_t$, $\forall (u,v) \in V^2$.
7: For every $(u,v) \in V^2$, the parties invoke the F_{Bool} sub-protocol where each host inputs its share $[B^{[D]}(u,v)]_t$ and receives as output a share $[\flat\left(B^{[D]}(u,v)\right)]_t$.
8: H_t computes $[\pi_1^D(u)]_t = \sum_{v \in V}[\flat(B^{[D]}(u,v))]_t$, $\forall u \in V$.
9: Each host H_t broadcasts $[\pi_1^D(u)]_t$, and all hosts reconstruct $\pi_1^D(u)$ for every $u \in V$.

Output: Each host outputs $\pi_1^D(u)$ for all $u \in V$.

Privacy. Our protocols are optimizations of circuits that implement the scores π_1^D, π_2^D and π_3^D, including placement of the gates and design of the layers. Correctness of the circuits is straightforward, hence their security follows directly from the security of the BGW protocol [3, 1].

Theorem 2 *Protocol* Π_j *for* $j \in \{1,2,3\}$*, privately computes the function* $F_j(A_1, \ldots, A_t) = (\{\pi_j^D(u) : u \in V\})$*, respectively, in the presence of an unbounded semi-honest adversary, with* $\ell' = \lceil \ell/2 \rceil - 1$.

5. SCALING TO BIG GRAPHS

All protocols in the previous section begin by computing shares in all entries of the matrix B (Π_3 and Π_1) or A (Π_2), which entails $\Omega(|V|^2)$ complexity. In case where $|V|$ is in the millions, that communication overhead is clearly impractical. In this section we show that it is possible to avoid sharing the matrix of the multigraph, but still compute *properties* of that multigraph, in particular the centrality score $\pi_3^D(u)$ for all $u \in V$. Specifically, we show how the hosts can compute the set of vectors $\mathbf{s}_1, \ldots, \mathbf{s}_D$, where $\mathbf{s}_k = (s_k(u) : u \in V)$ and $s_k(u)$ is as defined in Eq. (1), without sharing the matrices A or B, but, instead, sharing and transmitting $O(|V|)$ values only. This improvement dramatically reduces the runtime of the protocol and enables scaling the computation to graphs of millions of nodes.

Such a significant acceleration of runtime is possible due to the following two reasons. First, as we will see, our protocol heavily relies on the the special algebraic structure of the score π_3^D; in contrast, the algebraic structure of the other two measures, π_2^D and π_1^D, does not seem to allow a similar improvement in their respective protocols. Second, the Protocol Π_4 that we propose here reveals some additional values

to the hosts, in comparison to Protocol Π_3. Intuitively, by revealing more information there is less to hide, and therefore the computation can become more efficient. As a result, we benefit twice: the hosts gain more from the interaction, and enjoy this significant acceleration. To elaborate further, while protocol Π_3 reveals only the final vector of π_3^D scores, the efficient Protocol Π_4 that we present here reveals the set of vectors $\{\pi_3^d\}$ for every $1 \leq d \leq D$, and nothing more. We stress that these intermediate vectors reveal only general, yet meaningful, information about each of the nodes. Importantly, these vectors do not reveal any information about specific relationship between nodes nor specific information on any of the private graphs.

Our protocol proceeds in iterations, where in each iteration the hosts compute the vector $\mathbf{s}_k$ from the vector $\mathbf{s}_{k-1}$. Recall that by Eq. (3), $s_k(u) = \sum_{v \in V} B(u,v) s_{k-1}(v)$. If M is a matrix over V^2, we let $M[u]$ denote its row corresponding to u. Therefore, we can write $s_k(u) = \langle B[u], \mathbf{s}_{k-1} \rangle$, and, consequently,

$$s_k(u) = \left\langle \sum_{t=1}^{\ell} A_t[u], \mathbf{s}_{k-1} \right\rangle = \sum_{t=1}^{\ell} \langle A_t[u], \mathbf{s}_{k-1} \rangle \,. \qquad (4)$$

Moreover, if we let $\mathbf{1}$ denote the all-one vector of size $|V|$ then, by Eq. (1), $s_1(u) = \sum_{v \in V} B(u,v) = \langle B[u], \mathbf{1} \rangle = \sum_{t=1}^{\ell} \langle A_t[u], \mathbf{1} \rangle$. Hence, if we set $\mathbf{s}_0 := \mathbf{1}$, Eq. (4) holds for all $k \geq 1$.

In view of the above, Protocol Π_4 consists of a main loop over $k = 1, \ldots, D$ (Steps 2-9). In each iteration in that loop, each host H_t, for $1 \leq t \leq \ell$, individually computes the value $y_t(u) := \langle A_t[u], \mathbf{s}_{k-1} \rangle$, for all $u \in V$ (Step 3), where $\mathbf{s}_0$ was initialized to $\mathbf{1}$ in Step 1. Then, H_t secret shares $y_t(u)$, for all $u \in V$, with all hosts (Step 4). Each host then proceeds to locally sum up all the shares that it received from all ℓ hosts (Step 5). The resulting value for each $u \in V$ is that host's share, $[s_k(u)]_t$, for $s_k(u)$ (as implied by Eq. (4)). The hosts proceed to broadcast these shares (Step 6) and then each host reconstructs the value $s_k(u)$, for each $u \in V$, by applying the secret sharing reconstruction procedure, and construct the vector $\mathbf{s}_k$ (Steps 7-8). After completing the loop, each host has recovered all vectors $\mathbf{s}_k$, $1 \leq k \leq D$. Finally, each host computes $\pi_3^d(u)$ for all $u \in V$ and all $1 \leq d \leq D$ (Step 10).

Protocol Π_4: Computing $\pi_3^d(u)$, $\forall d \in \{1, \ldots, D\}, u \in V$.

Input: Each host H_t holds its association matrix A_t.

1: All hosts set $\mathbf{s}_0 := \mathbf{1}$.
2: **for** $k = 1, \ldots, D$ **do**
3: H_t computes $y_t(u) := \langle A_t[u], \mathbf{s}_{k-1} \rangle$, $\forall u \in V$.
4: H_t shares $y_t(u)$ with $H_1, \ldots, H_\ell$ for every $u \in V$. At the end of this step, H_t holds $[y_j(u)]_t$ for every $u \in V$ and $1 \leq j \leq \ell$.
5: H_t computes $[s_k(u)]_t = \sum_{j=1}^{\ell} [y_j(u)]_t$, $\forall u \in V$.
6: Each H_t broadcasts $\{[s_k(u)]_t : u \in V\}$.
7: H_t has now the shares $\{[s_k(u)]_j : 1 \leq j \leq t\}$. It proceeds to reconstruct the secrets $\{s_k(u) : 1 \leq j \leq t\}$.
8: H_t sets $\mathbf{s}_k = (s_k(u) : u \in V)$.
9: **end for**
10: H_t computes the value $\pi_3^d(u) = \sum_{k=1}^{d} \beta_k s_k(u)$ for every $u \in V$ and $1 \leq d \leq D$.

Output: H_t outputs $\pi_3^d(u)$ for every $u \in V$ and $1 \leq d \leq D$.

In addition to its scalability, Protocol Π_4 enjoys the following advantages:

(1) As opposed to the protocols described earlier, in Π_4 each host can internally represent its respective graph using *adjacency lists* or a *sparse* adjacency matrix, rather than a full *adjacency matrix*. Then, the individual computation of $\langle A_t[u], \mathbf{s}_{k-1} \rangle$ (Step 3) is proportional to the actual size of the row $A_t[u]$; in particular, it is independent of the size of the multigraph.

(2) The functionality that is computed in Π_4 is linear and does not contain any multiplication gates. Therefore, in this protocol, instead of using Shamir's ℓ'-out-of-ℓ secret sharing scheme with $\ell' < \ell/2$, we may use a secret sharing scheme with $\ell' = \ell$. The advantage is twofold: such a protocol is immune against coalitions of any size (while the previous protocols were immune only against coalitions smaller than ℓ'); and instead of a polynomial-based secret sharing scheme (which requires costly polynomial evaluations and interpolations), the ℓ-out-of-ℓ secret sharing scheme is a simple scheme in which any given secret is split into ℓ uniformly random shares whose sum equals the secret. In such a scheme, the generation of shares is very efficient and so is the reconstruction of the secret from all shares.

(3) Having the vectors $\mathbf{s}_1, \ldots, \mathbf{s}_D$ grants each host the freedom to apply an individual score function (say, by selecting its own weights $\beta_1, \ldots, \beta_D$).

Communication costs. As opposed to protocols Π_1, Π_2 and Π_3, the hosts here do not share the matrices A or B, and the communication complexity is proportional to $|V|$ only (and not $|V|^2$, or even $|E|$, where E is the set of edges in the unified graph). The number of rounds in each iteration is 2 (and therefore there are $2D$ rounds in total), and the total number of field elements that are transmitted in each iteration is $2\ell(\ell-1)|V|$ (or $2\ell(\ell-1)D|V|$ overall).

Computational costs. We summarize briefly the computational costs of our four protocols, in terms of running time per host. Protocol Π_1 (the Reach score, Section 4.3) runs in time $\widetilde{O}(|V|^3 D\ell^3)$, because it performs $O(|V|^3 D)$ secret multiplications, each of which takes $\ell^2 \ell' \widetilde{O}(||\mathbb{F}||)$ time. Protocols Π_2 (Truncated Katz, Section 4.2) and Π_3 (Multigraph Truncated Katz, Section 4.1) are more efficient and have roughly the same complexity, $\widetilde{O}(|V|^2 D^2 \ell^2)$. Finally, the running time of Π_4 can be shown to be $\widetilde{O}(|V| D^2 \ell^2 + D^2 T)$, where T is the sum of the number of edges of all graphs.

Privacy. We prove the following Theorem:

Theorem 3 *Protocol Π_4 privately computes the function $F(A_1, \ldots, A_t) = (\{\pi_3^d(u) : 1 \leq d \leq D, u \in V\})$ in the presence of an unbounded semi-honest adversary corrupting at most $\ell - 1$ parties.*

Proof. The protocol can be represented as D sequential invocations of a "secure addition" protocol. That is, let F_{add} be the following addition functionality,

$$F_{add}(\mathbf{y}_1, \ldots, \mathbf{y}_\ell) = (\mathbf{s}, \ldots, \mathbf{s}) \ ,$$

where each $\mathbf{y}_j = (y_j(u) : u \in V)$ is a vector of size $|V|$, and each component in the output is the vector $\mathbf{s} = \sum_{j=1}^{\ell} \mathbf{y}_j$. Thus, given the output of the previous stage $\mathbf{s}_{k-1}$ (where initially $\mathbf{s}_0$ is the all-one vector $\mathbf{1}$), each host H_t, $1 \leq t \leq \ell$, computes its input vector $\mathbf{y}_t^{(k)} = A_t \cdot \mathbf{s}_{k-1}$, and the parties then invoke the functionality F_{add}. The output of F_{add}, denoted $\mathbf{s}_k$, will be served as the input for the next stage. Note that Protocol Π_4 implements this F_{add}-functionality

exactly as the BGW protocol implements it; hence, the security of this sub-protocol relies on the security of the BGW protocol [3, 1]. Interpreting the protocol as D sequential invocations of the F_{add}-functionality, we have in fact a protocol in the F_{add}-hybrid model. It is easy to see that in this protocol, the corrupted parties do not receive any messages from the honest parties and their view consists of the outputs of the F_{add} invocations only. However, these values are exactly the vectors $\mathbf{s}_1, \ldots, \mathbf{s}_D$ which the simulator receives as input (as the output of the F_{add}-functionality for the corrupted parties). Therefore, simulation is straightforward, where the simulator just outputs the values $\mathbf{s}_1, \ldots, \mathbf{s}_D$ (that it received as input) as the outputs of F_{add}; clearly, those values are exactly the view of the corrupted parties in the real execution of the protocol Π_4. □

6. EXPERIMENTS

Since our protocols are provably secure (Theorems 2 and 3), and correct (i.e., return the same answer as their non-secure counterparts), the main goal of our experimental assessment is to study their efficiency. In this regard we will report and analyze their total work defined as the sum of running times over all hosts. Observe that, while the number and total size of communication messages may also be of interest, no experiments are required here as these numbers are completely determined by our protocols (see Table 1).

Experiments settings. All our experiments are run on a dual-core Intel i7-5600U CPU (2.60 GHz) with 16Gb RAM under Linux. We implemented our protocols in `C++` and compiled the code using `g++` with speed optimizations.
We used the NTL library[5] to perform computations with arbitrary length integers, as well as polynomial interpolation over large finite fields (as required by our protocols). The finite fields we use are integers modulo a prime large enough to guarantee exact computation of the results (e.g., of length $O(\log \ell)$ for π_1, and $O(D \log(|V|\ell))$ for π_2 and π_3). The threshold parameter ℓ' is set to $\lfloor(\ell-1)/2\rfloor$ for Π_1, Π_2, and Π_3 as in Theorem 2 (recall that $\ell \geq 3$ denotes the number of hosts). For Π_4 we set $\ell' = \ell$ as this protocol allows the use of the ℓ-out-of-ℓ secret sharing scheme (see Section 5). The weights β_k for matrix powers are set to $1/2^k$, $1 \leq k \leq D$.

Datasets. For our experiments we used publicly-available real-world multigraphs of various types and sizes, spanning different domains: offline relationships (aarhus), road networks (london), genetic interactions (hiv, arabi), online social networks (higgs, obama), interactions in sharing sites (youtube), human genealogies (wikitree) and co-authorship networks (dblp). All except dblp are separated by layers; for dblp we use instead the timestamp information to partition the edges into 3 groups. Some of these graphs are undirected; in this case we duplicate each edge. Table 2 reports their main characteristics and origin.[6,7,8,9]

6.1 π_3 as a median ranking

In this section we show that, if the hosts are interested in a ranking of "influential" users, the ranking obtained by joining forces through secure multiparty computation is intuitively better than the individual rankings that each host should be able to obtain on its own. Specifically, the ranking obtained by π_3 on the multilayer graph lies "in the middle" of all the individual rankings obtained on each layer (π_2 and π_3 coincide when using a single layer).

We proceed as follows for an ℓ-layer dataset:

(1) Compute all π_3 (or equivalently π_2) scores based on each single layer, obtaining ℓ rankings $R_1, \ldots, R_\ell$. Then we obtain the additional ranking $R_{\ell+1}$ induced by the π_3 scores on the multilayer network.

(2) Compute a symmetric table with the Kendall-Tau[10] correlation between all the rankings from the previous step.

(3) For each $i \in [\ell+1]$, define the global score of i as the sum of all Kendall-Tau correlations between R_i and R_j for $j \neq i$. The larger this value is, the better the ranking i correlates with all other rankings on average.

Tables 3 and 4 show the correlations thus obtained for arabi and youtube, respectively. We verify that the global score of the ranking induced by π_3 is larger than any of the individual scores. That is, by applying a secure protocol to compute π_3, all parties can obtain a ranking that is closer on average to all individual rankings.

6.2 Efficiency of Π_1, Π_2, and Π_3

We next report runtime comparison of the three basic protocols Π_1, Π_2, and Π_3. We already know, by the computational complexity analysis reported in Section 5, about the inefficiency of Π_1, which constraints us to use small graphs for this comparison.

Table 2: Dataset characteristics: name, directed (D) or undirected (U), source (please refer to the footnotes below), number of hosts, number of nodes, number of edges in the union graph $G = (V, E)$ (corresponding to the sum of entries of the matrix A), and number of edges considering their multiplicity (corresponding to the sum of entries of the matrix B).

dataset	U/D	url	ℓ	$\lvert V\rvert$	$\lvert E\rvert$	$\sum_{t=1}^{\ell} \lvert E_t\rvert$
aarhus	U	6	5	61	353	620
london	U	6	3	369	430	503
hiv	D	6	3	1,005	2,310	2688
arabi	U	6	7	6,980	17,497	18,117
youtube	U	7	5	14,992	10,726,107	32,980,158
higgs	D	6	3	304,691	904,404	1,110,962
wikitree	D	8	4	1,382,750	9,620,090	18,381,878
dblp	U	9	3	1,314,050	10,724,828	18,986,618
obama	D	6	3	2,281,259	6,283,002	9,182,052

Table 3: Kendall-Tau correlations between rankings induced by the 7 individual layers of arabi, and π_3-based ranking using all layers. The last column is the sum of the row values minus 1.

layer	1	2	3	4	5	6	7	π_3	$\sum$ - 1
1	1.0	-0.307	0.042	-0.001	0.033	0.052	0.0	0.314	0.13
2	-0.307	1.0	0.013	0.01	0.018	0.041	0.055	0.452	0.28
3	0.042	0.013	1.0	0.392	0.272	0.028	-0.002	0.055	0.8
4	-0.001	0.01	0.392	1.0	0.237	0.046	0.017	0.023	0.72
5	0.033	0.018	0.272	0.237	1.0	0.053	0.026	0.036	0.68
6	0.052	0.041	0.028	0.046	0.053	1.0	0.14	0.063	0.42
7	0.0	0.055	-0.002	0.017	0.026	0.14	1.0	0.031	0.27
π_3	0.314	0.452	0.055	0.023	0.036	0.063	0.031	1.0	**0.97**

Table 4: Kendall-Tau correlations between rankings induced by the 5 individual layers of youtube, and π_3-based ranking using all layers. The last column is the sum of the row values minus 1.

layer	1	2	3	4	5	π_3	$\sum$ - 1
1	1.0	0.421	0.379	0.386	0.219	0.401	1.8
2	0.421	1.0	0.37	0.239	0.317	0.483	1.83
3	0.379	0.37	1.0	0.35	0.353	0.615	2.07
4	0.386	0.239	0.35	1.0	0.137	0.374	1.49
5	0.219	0.317	0.353	0.137	1.0	0.617	1.64
π_3	0.401	0.483	0.615	0.374	0.617	1.0	**2.49**

[5] http://www.shoup.net/ntl/

[6] http://deim.urv.cat/~manlio.dedomenico/data.php

[7] http://socialcomputing.asu.edu/datasets/YouTube

[8] http://proj.ise.bgu.ac.il/sns/wikitree.html

[9] http://konect.uni-koblenz.de/networks/dblp_coauthor

[10] https://en.wikipedia.org/wiki/Kendall_tau_distance

Table 5: Running times (s) for computing π_3 in a centralized setting (left column) versus computation by Π_4 (right column). These running times represent the total work (summed over all hosts) to compute centralities for every node in the network.

dataset	$D=3$		$D=5$		$D=10$	
arabi	0.01	0.08	0.01	0.28	0.03	1.11
youtube	3.23	1.42	4.89	4.04	10.28	15.56
higs	0.88	1.28	1.20	3.27	2.19	14.56
wikitree	4.77	7.02	7.44	20.55	15.18	90.53
dblp	7.22	5.51	11.53	16.67	21.83	72.72
obama	5.98	7.99	9.45	24.11	17.94	109.26

Figure 2 reports the running time of the three protocols on the datasets aarhus, london, and hiv. It is important to note that in the first two, due to the different order of magnitude of the runtime of Π_1, the Y-axis has a logarithmic scale. On the third dataset instead, we do not report Π_1 at all. The figure confirms that, as expected, Π_3 and Π_2 are close in runtime, even if Π_3 is always faster. Π_1 instead is, as already mentioned several times, totally impractical.

6.3 Efficiency and scalability of Π_4

We now focus on our most efficient protocol, Π_4, and study how its runtime varies with the size of the graph, the number of hosts, and the parameter D, when everything else is fixed. Table 5 reports run-times of Π_4 versus a centralized computation, as a demonstration of the price of privacy.

We observe that in youtube, for $D=3$, the secure protocol runs faster than the centralized one. Moreover, while the run-times are comparable, they increase quadratically with D in the secure protocol, but roughly linearly in the centralized one. While these two facts may seem counterintuitive, the reason behind them is our choice of different implementations, tailored to the needs of each task so as to speed them up. The centralized computation of π_3 only needs to use arbitrary-size integer arithmetic, but there is no need to perform arithmetic modulo a large prime (which would slow things down). In contrast, for the secure computation, this is not possible as we need to work over a large finite field, whose size needs to be known right from the start. But instead of performing all computations modulo a large prime (of the order of $(\ell|V|)^D$), we choose to use D different primes of the order of $\ell|V|$, and then reconstruct the final result using the Chinese Remainder Theorem. This speeds up some computations as opposed to working directly on a very large field, as only word-size integers are used. However, as D increases, we pay a factor of D because we perform similar work D times. While this factor of D is also implicit in the centralized computation (because intermediate results may grow to be $O(D)$-bits long), in practice the real-world inputs are not worst-case and most intermediate results are less than $O(D)$-bits long, so the run-time increases by a sub-linear factor of D instead. Coupled with the fact that the number of rounds for both methods is D, this explains the slowdowns by roughly D and D^2 in both methods.

Figure 2 (right column) shows the run-time of Π_4. The first two plots show that run-time increases roughly quadratically as a function of D and of ℓ. The third plot examines the dependence of the run-time on the number of nodes: sub-networks of a certain size are created by picking a random node and performing a BFS until the required number of nodes is reached. Here we observe that run-time increases roughly linearly.

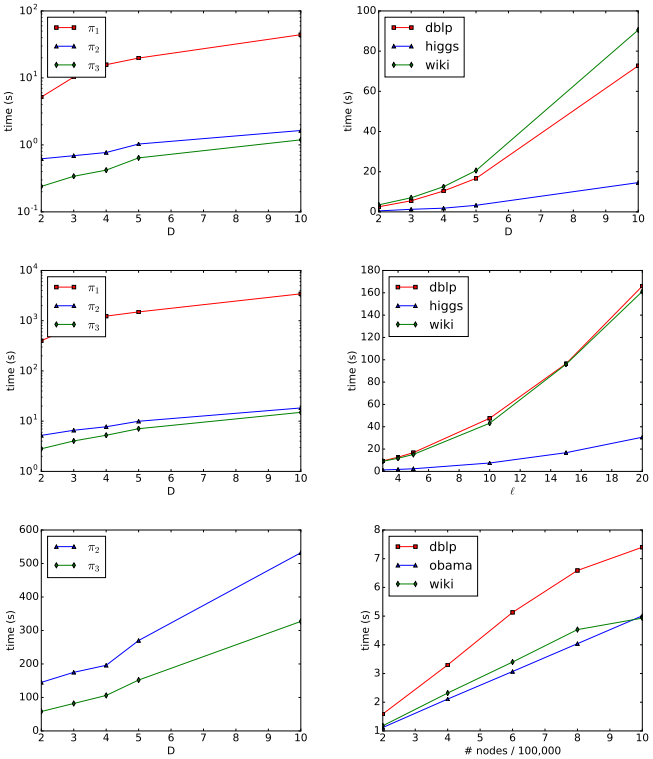

Figure 2: Left column: running time of protocols Π_1, Π_2, and Π_3 on three datasets (aarhus, london, and hiv top to bottom) for varying D. Right column reports scalability of Π_4. Top to bottom: running time as a function of D; as a function of ℓ with $D=3$; as a function of $|V|$ with $D=3$.

7. CONCLUSIONS AND FUTURE WORK

We study a setting in which several social network hosts wish to compute a centrality score for each user in the multi-layer graph induced by their private graphs. We define three centrality scores and devise secure multiparty protocols for computing them. For π_3^D – the most sensitive measure – we devise a protocol which has computational and communication costs that are linear in the number of users, whence it is scalable to very large graphs.

This study is one of very few studies to date that present secure multiparty protocols on distributed multi-layer graphs, where different players hold different layers of links. As future work we intend to explore the applicability of the methods presented here to compute the PageRank score [5] when the underlying graph data is distributed among several hosts. The PageRank score is closely related to our π_2^D; the only difference between the two scores is that while π_2^D is based on powers of the adjacency matrix A, PageRank is based on powers of the matrix $(K^{-1}A)^T$, where K is a diagonal matrix that records the out-degrees.

Acknowledgement. The first author was supported by a Junior Fellow award from the Simons Foundation; some of the work was done while the author was a postdoc at the IBM Research, supported by NSF Grant No. 1017660, and in the Hebrew University of Jerusalem, supported by Israeli Centers of Research Excellence (I-CORE) Program (Center No. 4/11). The other authors received funding from the EU Horizon 2020 innovation action programme under grant agreement No 653449 (TYPES project).

8. REFERENCES

[1] G. Asharov and Y. Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *J. Cryptology*, 30(1):58–151, 2017.

[2] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Conference on Computer and Communications Security*, pages 535–548, 2013.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10, 1988.

[4] P. Boldi and S. Vigna. Axioms for centrality. *Internet Mathematics*, 10(3-4):222–262, 2014.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[6] M. Burkhart, M. Strasser, D. Many, and X. A. Dimitropoulos. SEPIA: privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX*, pages 223–240, 2010.

[7] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[8] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *STOC*, pages 11–19, 1988.

[9] M. De Domenico, A. Solé-Ribalta, S. Gómez, and A. Arenas. Navigability of interconnected networks under random failures. *Proceedings of the National Academy of Sciences*, 111(23):8351–8356, 2014.

[10] D. Demmler, T. Schneider, and M. Zohner. Ad-hoc secure two-party computation on mobile devices using hardware tokens. In *USENIX Security Symposium*, pages 893–908, 2014.

[11] K. C. Foster, S. Q. Muth, J. J. Potterat, and R. B. Rothenberg. A faster katz status score algorithm. *Computational & Mathematical Organization Theory*, 7(4):275–285, 2001.

[12] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and fact-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.

[13] O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[14] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[15] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[16] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[17] A. Halu, R. J. Mondragón, P. Panzarasa, and G. Bianconi. Multiplex pagerank. *PloS one*, 8(10):e78293, 2013.

[18] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *Transactions on Knowledge and Data Engineering*, 16(9):1026–1037, 2004.

[19] L. Katz. A new status index derived from sociometric index. *Psychometrika*, pages 39–43, 1953.

[20] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter. Multilayer networks. *Journal of Complex Networks*, 2(3):203–271, 2014.

[21] N. Korula and S. Lattanzi. An efficient reconciliation algorithm for social networks. *PVLDB*, 7(5):377–388, 2014.

[22] D. Liben-Nowell and J. M. Kleinberg. The link prediction problem for social networks. In *CIKM*, pages 556–559, 2003.

[23] X. Lin, C. Clifton, and M. Zhu. Privacy-preserving clustering with distributed EM mixture modeling. *Knowl. Inf. Syst.*, 8(1):68–81, 2005.

[24] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Crypto*, pages 36–54, 2000.

[25] Y. Lindell and B. Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[26] Z. Lu, B. Savas, W. Tang, and I. S. Dhillon. Supervised link prediction using multiple sources. In *ICDM*, pages 923–928, 2010.

[27] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[28] M. J. Rattigan and D. Jensen. The case for anomalous link discovery. *SIGKDD Explor. Newsl.*, 7(2):41–47, 2005.

[29] P. Sarkar and A. W. Moore. A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In *UAI*, pages 335–343, 2007.

[30] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[31] A. Solé-Ribalta, M. De Domenico, S. Gómez, and A. Arenas. Centrality rankings in multiplex networks. In *WebSci*, pages 149–155, 2014.

[32] D. R. Stinson. *Cryptography - theory and practice*. Discrete mathematics and its applications series. Chapman & Hall/CRC, 2006.

[33] T. Tassa. Secure mining of association rules inhorizontally distributed databases. *Transactions on Knowledge and Data Engineering*, 26(4):970–983, 2014.

[34] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *KDD*, pages 639–644, 2002.

[35] C. Wang, V. Satuluri, and S. Parthasarathy. Local probabilistic models for link prediction. In *ICDM*, pages 322–331, 2007.

[36] A. Yao. Protocols for secure computation. In *FOCS*, pages 160–164, 1982.

[37] A. C. Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167, 1986.

[38] J. Zhan, S. Matwin, and L. Chang. Privacy preserving collaborative association rule mining. In *Data and Applications Security*, pages 153–165, 2005.