Privacy-Preserving Analytics on Decentralized Social Graphs: The Case of Eigendecomposition

Songlei Wang, Yifeng Zheng, Xiaohua Jia, Fellow, IEEE, and Xun Yi

Abstract—Analytics over social graphs allows to extract valuable knowledge and insights for many fields like community detection, fraud detection, and interest mining. In practice, decentralized social graphs frequently arise, where the social graph is not available to a single entity and is decentralized among a large number of users, each holding only a limited local view about the whole graph. Collecting the local views for analytics of decentralized social graphs raises critical privacy concerns, as they encode private information about the social interactions among individuals. In this paper, we design, implement, and evaluate PrivGED, a new system aimed at privacy-preserving analytics over decentralized social graphs. PrivGED focuses on the support for eigendecomposition, one popular and fundamental graph analytics task producing eigenvalues/eigenvectors over the adjacency matrix of a social graph and benefits various practical applications. PrivGED is built from a delicate synergy of insights on graph analytics, lightweight cryptography, and differential privacy, allowing users to securely contribute their local views on a decentralized social graph for a cloud-based eigendecomposition analytics service while gaining strong privacy protection. Extensive experiments over real-world social graph datasets demonstrate that PrivGED achieves accuracy comparable to the plaintext domain, with practically affordable performance superior to prior art.

Index Terms—Decentralized social graph analytics, cloud computing, security, privacy preservation

1 INTRODUCTION

Analytics over information-rich social graphs allows the extraction of valuable and impactful knowledge and insights for many fields like community detection, fraud detection, and interest mining [1], [2]. Social graph analytics, however, becomes quite challenging when the social graph is not available to a single entity and presented in a decentralized manner, where each user only holds a limited local view about the whole social graph, and the complete social graph is formed by their collective views. Decentralized social graphs can arise in many practical applications [3], [4], [5], [6], [7]. For example, in a phone network, each user has his own contact list and the collective contact lists of all users form a social graph in a decentralized manner [4].

Collecting individual users’ local views for analytics in the setting of decentralized social graphs can raise critical privacy concerns, as these local views encode sensitive information regarding the social interactions among individuals [4], [8]. Users thus may be reluctant to be engaged in such analytics if their local views gain no protection. Thus, it is of critical importance to ensure that security must be embedded in analytics over decentralized social graphs from the very beginning, so that valuable knowledge and insights can be extracted without compromising the privacy of individual users. Among others, one popular and fundamental graph analytics task is eigendecomposition which we focus on as a concrete instantiation in this paper. Eigendecomposition-based social graph analytics works on the adjacency matrix associated with a social graph to yield eigenvalues/eigenvectors, and can benefit various applications, such as community structure detection [9], important members finding [10], and social graph partitioning [9] (see Section 2.1 for more details on applications).

In the literature, little work [8], [11], [12] has been done regarding privacy-preserving eigendecomposition on graphs. Some works [11], [12] focus on publishing adjacency matrices with differential privacy while preserving their eigenvalues/eigenvectors. Yet these works operate with centralized social graphs, where the social graph is available to a single entity and processed in the plaintext domain. The most related (state-of-the-art) work to ours is due to Sharma et al. [8], who propose a method PrivateGraph that works under a decentralized social graph setting and aims to provide privacy protection for individuals’ local views.

However, PrivateGraph is not quite satisfactory due to the following downsides in functionality and security. First, PrivateGraph only supports eigendecomposition on undirected graphs (via the Lanczos method [13]), but many social graphs in practice are directed [14], which cannot be supported via the Lanczos method. Second, it requires some users to expose
the number of their friends, posing a threat to their privacy [15]. Third, it requires frequent online interactions between the cloud that coordinates the eigendecomposition based analytics task and the entity that requests the eigenvalues/eigenvectors. Therefore, how to achieve privacy-preserving eigendecomposition-based analytics over decentralized social graphs is still challenging and remains to be fully explored.

In light of the above, in this paper, we design, implement, and evaluate PrivGED, a new system that allows privacy-preserving analytics over decentralized social graphs with eigendecomposition. Leveraging the emerging cloud-empowered graph analytics paradigm, PrivGED allows a set of users to securely contribute their local views on a decentralized social graph for an eigendecomposition analytics service empowered by the cloud, while ensuring strong protection on individual local views.

We start with considering how to enable the individual local views to be securely collected so as to form the (encrypted) adjacency matrix adequately for eigendecomposition on the cloud. Each row vector in the adjacency matrix stores information of the local view of a user. Targeting security assurance as well as high efficiency, PrivGED resorts to a lightweight cryptographic technique—additive secret sharing (ASS) [16], for fast encryption of the elements in local view vectors. However, simply applying ASS over each user’s complete vector is inefficient because social graphs are usually large-scale and sparse [14], leading to many zero elements in local view vectors that incur undesirable performance overheads. To tackle this problem, PrivGED develops techniques that allow to exploit the benefits of graph sparsity for efficiency while protecting the privacy of users’ private social relationships, through a delicate synergy of sparse representation, local differential privacy (LDP) [4], and function secret sharing (FSS) [17], [18] techniques. As opposed to PrivateGraph [8], PrivGED does not reveal any users’ exact node degree information.

Subsequently, we consider how to enable eigendecomposition to be securely performed on the cloud over the formed encrypted adjacency matrix. We first make two important practical observations: i) Usually only the top-k eigenvalues/eigenvectors \( k \ll N \); \( N \) is the number of users and determines the size of the adjacency matrix) are needed in practice [9], [10], [19], [20], [21], [22], [23]; ii) The desired top- \( k \) eigenvalues/eigenvectors can be derived from the complete eigenvalues/eigenvectors of a smaller matrix reduced from the original matrix under adequate dimension reduction methods, among which the most popular ones are the Arnoldi method [24] (for general (possibly non-symmetric) matrices) and the Lanczos method [13] (for symmetric matrices).

Therefore, PrivGED first introduces effective techniques, which tackle the challenging squared root and division operations in the secret sharing domain, to securely realize the Arnoldi method and the Lanczos method. This allows PrivGED to flexibly work on both undirected and directed graphs. PrivGED then further provides techniques for secure realization of the widely used QR algorithm [25] over the dimension-reduced matrix so as to produce the encrypted desired eigenvalues/eigenvectors. We reformulate the plaintext QR algorithm to ease computation in the ciphertext domain as well as optimize the processing of secret-shared matrix multiplications for high performance. We highlight our main contributions below:

- We present PrivGED, a new system supporting privacy-preserving analytics over decentralized social graphs with eigendecomposition.
- We develop techniques for secure collection of individual local views on the decentralized social graph, which exploit the benefits of graph sparsity for efficiency while protecting the privacy of individual’s social relationships.
- We develop techniques for securely realizing the Arnoldi/Lanczos methods and the QR algorithm, so as to fully support the processing pipeline of secure eigendecomposition on the cloud.
- We formally analyze the security of PrivGED, implement it with \( \sim2000 \) lines of Python code, and conduct an extensive evaluation over three real-world datasets. The results demonstrate that PrivGED achieves accuracy comparable to the plaintext domain, with practically affordable performance superior to prior art.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 introduces preliminaries. Section 4 presents the problem statement. Sections 5 and 6 give the detailed design. The privacy and security analysis is presented in Section 7. We present experiment results in Section 8 and conclude this paper in Section 9.

2 RELATED WORK

2.1 Graph Analytics via Eigendecomposition

Graphs can characterize the complex inter-dependency among entities, and are used in various applications, such as social networks [14] and webpage networks [20]. As one popular and fundamental graph analytics task, eigendecomposition works on the adjacency matrix to yield eigenvalues/eigenvectors, and can benefit various applications [9], [10], [19], [20], [21], [22], [26]. For example, eigendecomposition-based graph analytics can greatly benefit community detection through the following ways: 1) finding the community structure based on the eigenvectors [19]; 2) identifying and characterizing nodes importance to the community according to the relative change in the eigenvalues after removing them [10]; 3) partitioning a social graph based on its top-2 eigenvectors [9]. Another important application is PageRank [20], [21], [22], [26], which is one of the best-known ranking algorithms in web search. PageRank measures the importance of website pages by computing the principal eigenvector of the matrix describing the hyperlinks of the website pages. In addition, the second eigenvector can be used to detect a certain type of link spamming [23]. However, all of them consider the execution of eigendecomposition in the plaintext domain without privacy protection.

2.2 Privacy-Preserving Graph Analytics

There exist a variety of designs that aim to securely perform certain graph analytics tasks. Some works [27], [28], [29] focus on privacy-preserving training of graph neural networks (GNN) based on the federated learning paradigm [30], which aim to train GNN models across multiple clients.
holding local datasets (e.g., spatio-temporal datasets [27] or graphs [28], [29]) in such a way that the datasets are kept local. The work [27] focuses on GNNs over decentralized spatio-temporal data, and has the clients exchange model updates with the server in cleartext. In contrast, the works [28], [29] focus on graph datasets and design privacy-preserving mechanisms to protect the individual model updates.

There has been a line of work [31], [32], [33], [34], [35] aimed at the support for graph analytics with cryptographic methods like secure multi-party computation techniques and searchable encryption. The main focus of this line of work has been on supporting different kinds of graph queries under different scenarios in a secure manner. Wu et al. [31] propose a protocol for privacy-preserving shortest path query in a two-party setting, where a client holds a query and a server holding a plaintext graph, based on cryptographic techniques including private information retrieval, garbled circuits, and oblivious transfer. In contrast, the works [34], [35] focus on the support for privacy-preserving shortest path queries in an outsourcing setting, where private queries need to be executed over encrypted graphs outsourced to the cloud. The work [32] designs protocols that can support privacy-preserving shortest distance queries and maximum flow queries over outsourced encrypted graphs. These works rely on the combination of searchable encryption (e.g., order-revealing encryption), homomorphic encryption, and/or garbled circuits. In [33], Araki et al. consider a scenario where all nodes and edges of a graph are secret-shared between three servers and devise protocols for breadth-first search and maximal independent set queries, based on secret sharing and secure shuffling. The above works all target graph analytics tasks that are substantially different from the one considered in this paper.

There is another line of work [11], [12] focuses on publishing graph matrices with differential privacy while preserving their eigenvalues/eigenvectors. They work under the setting of centralized social graphs where the social graph is held by a single entity and processed in the plaintext domain. Some works consider the scenario of decentralized social graphs, and focus on the privacy-preserving support for different tasks with differential privacy, such as estimating subgraph counts [3] and generating representative synthetic social graphs [4].

The state-of-the-art design that is most related to ours is PrivateGraph [8], which is also aimed at eigendecomposition analytics over decentralized social graphs with privacy protection. However, as mentioned above, PrivateGraph is subject to several crucial downsides in terms of functionality and security, which greatly limit its practical usability. In light of this gap, we present a new system design PrivGED for privacy-preserving eigendecomposition analytics on decentralized social graphs. Compared to PrivateGraph, PrivGED is much advantageous in that it (i) supports both directed and undirected social graphs (via secure realizations of both the Arnoldi method and the Lanczos method), (ii) does not reveal any users’ exact degree information, and (iii) fully exploits the cloud to free the analyst from staying online for active and frequent interactions and conducting a large amount of local intermediate and post processing. In particular, as reported in [8], to obtain the eigenvalues/eigenvectors, the analyst must spend 0.2 hours as well as communicate 10 GB with the cloud. In contrast, PrivGED allows the analyst to directly receive the final eigenvalues/eigenvectors. Table 1 summarizes the prominent advantages of our PrivGED over PrivateGraph.

| Property                           | PrivateGraph [8] | PrivGED |
|------------------------------------|------------------|---------|
| Undirected graph supported         | ✔                | ✔       |
| Directed graph supported           | ✗                | ✔       |
| All users’ degrees protected       | ✗                | ✔       |
| Analyst allowed to stay offline    | ✗                | ✔       |
| Lightweight cryptography           | ✗                | ✔       |

3 PRELIMINARIES

3.1 Eigendecomposition-Based Graph Analytics

A graph comprises a set of nodes with a corresponding set of edges which connect the nodes. The edges may be directed or undirected and may have weights associated with them as well. Eigendecomposition works on the adjacency matrix associated with a graph to yield eigenvalues/eigenvectors. A complete eigendecomposition on an $N \times N$ matrix poses a considerable time complexity of $O(N^3)$, which indeed also results in unnecessary cost for a large $N$ since only top-$k$ ($k < N$) eigenvalues/eigenvectors are used in most eigendecomposition-based graph analysis tasks [9], [10], [19], [20], [21], [22], [23]. In practice, given a large-scale adjacency matrix $A$, to calculate its top-$k$ eigenvalues/eigenvectors, the first step is to reduce its dimension from $N \times N$ to $M \times M$ ($M$ is usually slightly larger than $k$), producing a new matrix $\tilde{A}$ for further processing. The most popular dimension reduction methods are the Arnoldi method (Algorithm 1) [24] and the Lanczos method (Algorithm 2) [13], which work on general (possibly non-symmetric) matrices and symmetric matrices, respectively. After dimension reduction, the QR algorithm [25] is usually used to efficiently calculate the complete eigenvalues/eigenvectors of $\tilde{A}$. Finally, the top-$k$ eigenvalues of $\tilde{A}$ are used to represent the top-$k$ eigenvalues of $A$, and the corresponding eigenvectors $\tilde{V}$ can be transformed to the eigenvectors $V$ of $A$ by $V = \tilde{P}\tilde{V}$ where $P$ is determined by line 11, Algorithm 1 or line 11, Algorithm 2.

3.2 Local Differential Privacy

Compared to the traditional differential privacy model [36] that assumes a trusted data collector which can collect and see raw data, the recently emerging LDP model [37] considers the data collector to be untrusted, in which each user only reports perturbed data with calibrated noises added. The formal definition of $(\epsilon, \delta)$-LDP is as follows.

**Definition 1.** A randomized mechanism $M$ satisfies $(\epsilon, \delta)$-LDP, if and only if for any inputs $x$ and $x'$, we have: $\forall y \in \text{Range}(M)$,

$$Pr[M(x) = y] \leq e^\epsilon \cdot Pr[M(x') = y] + \delta.$$
Algorithm 1. The Arnoldi Method

Input: A non-symmetric matrix $A$; the target dimension $M$.
Output: A new matrix $\tilde{A}$ with dimension $M \times M$ and $P$.
1: Start with an arbitrary vector $p_1$ with $L^2$ norm 1.
2: for $k \in [2, M]$ do
3: $p_k = A p_{k-1}$.
4: for $j \in [1, k-1]$ do
5: $\tilde{A}[j, k-1] = p^T_j \cdot p_k$. # $T$ denotes transposition.
6: $p_k = p_k - \tilde{A}[j, k-1] \cdot p_j$.
7: end for
8: $\tilde{A}[k, k-1] = ||p_k||$. # $|| \cdot ||$ denotes $L^2$ norm.
9: $P_k = \tilde{A}^{k-1}$.
10: end for
11: $P = [p_1, \ldots, p_M]$.

where $\text{Range}(M)$ denotes the set of all possible outputs of $M$, $\epsilon$ is the privacy budget, and $\delta$ is a privacy parameter.

Laplace distribution is a widely popular choice to draw the noises, which is formally defined as follows.

Definition 2. A discrete random variable $x$ follows $\text{Lap}(\epsilon, \delta, \Delta)$ distribution if its probability density function is [38]

$$Pr[x] = \frac{e^{x} - 1}{e^{\epsilon} + 1} \cdot e^{-\frac{|x|}{\epsilon}}, \forall x \in \mathbb{Z}.$$ 

where $\mu$ is the mean of the Laplace distribution:

$$\mu = -\Delta \cdot \ln\left(\frac{e^{\epsilon} + 1}{(1 - (1 - \delta)\epsilon)}\right).$$

(1)

$\Delta$ is the sensitivity of a function $f$

$$\Delta = \max\{|f(x) - f(x')|\},$$

which captures the magnitude by which a single entity’s data can change the output of $f$ in the worst case [36].

3.3 Additive Secret Sharing

Given a private value $x \in \mathbb{Z}_{p^2}$, ASS in a two-party setting works by splitting it into two secret shares $\langle x \rangle_1$ and $\langle x \rangle_2$ such that $x = \langle x \rangle_1 + \langle x \rangle_2$ [16]. Each share alone reveals no information about $x$. We denote by $[x]$ the ASS of $x$ for short. It is noted that if $k = 1$, we say the secret sharing is binary sharing, denoted as $[x]^B$, and otherwise arithmetic sharing, denoted as $[x]^A$. Given a public constant $c$, and the secret sharings $[x]$ and $[y]$, addition/subtraction $[x \pm y] = [x] \pm [y]$ and scalar multiplication $[c \cdot x] = c \cdot [x]$ can be performed without interaction among the two parties that hold the shares respectively, while multiplication $[x \cdot y] = [x] \cdot [y]$ requires the two parties to have one round of online communication with the use of Beaver triples which can be prepared offline. It is noted that addition and multiplication over $\mathbb{Z}_2$ are equivalent to XOR and AND respectively.

3.4 Function Secret Sharing

FSS [39] is a low-interaction secret sharing for secure computation, presenting prominent advantages in online communication and round complexity compared to other alternative techniques, such as garbled circuits [40] or ASS. From a high-level point of view, a two-party FSS-based approach to a private function $f$ consists of a pair of probabilistic polynomial time (PPT) algorithms: (i) $(k_1, k_2) \sim \text{Gen}(1^\lambda, f)$: given a security parameter $\lambda$ and a function description $f$, output two succinct FSS keys $k_1, k_2$, each for one party. (ii) $(f(x)), \alpha_1 \sim \text{Eval}(k_1, x)$: given an FSS key $k_1$ and an evaluation point $x \in \{0, 1\}^n$, output the share $(f(x))$, of the function evaluation result. The security of FSS ensures that an adversary learning only one FSS key $k_i, i \in \{1, 2\}$ learns no information about the function $f$ and output $f(x)$.

4 System Overview

4.1 Architecture

Fig. 1 illustrates PrivGED’s system architecture. There are three kinds of entities: the users $U_i (i \in [1, N])$, the cloud, and the analyst. The users and the social relationship (friendship for simplicity) among them constitute a (decentralized) social graph, where each $U_i$ represents a graph node and the friendship between any two users indicates the existence of an edge, and the number of each $U_i$’s associated friends is the degree of the corresponding graph node. Consider as a concrete example a phone network [4], where each $U_i$ has a limited phone numbers in his contact list and thus a limited local view about the social graph.

The decentralized social graph can be characterized by an adjacency matrix $A$ of size $N \times N$, where each row $A[i, :](i \in [1, N])$ indicates $U_i$’s local view. For example, in an unweighted social graph, $A[i, j] = 1$ may indicate that $U_i$ and $U_j$ are friends; in a weighted social graph, $A[i, j] = v$ may indicate the degree of intimacy between $U_i$ and $U_j$ by a value $v$. The users are willing to allow analytics over their federated data of local views to produce analytical results...
for the analyst. Our focus in this paper is the analytics task of eigendecomposition, which plays a vital role in graph analytics and has many applications as aforementioned. However, due to privacy concerns, each $U_i$ is not willing to disclose his private social relationship throughout the analytical process over the data federation, so the enforcement of data protection is demanded.

The graph analytics service is empowered by the cloud for well-understood benefits. In PrivGED, the power of the cloud is split into two cloud servers from different trust domains which can be hosted by different service providers in practice. Such multi-server model is getting increasingly popular in recent years for security designs in various domains, including both academia [32], [41], [42], [43], [44], [45], [46], [47] and industry [48], [49]. The adoption of such model in PrivGED follows this trend. The two cloud servers in PrivGED, denoted as $CS_{\{1,2\}}$, collaboratively perform the analytics task of eigendecomposition without seeing each $U_i$’s data, and produce encrypted results (the top-$k$ eigenvalues/eigenvectors) which are delivered to the analyst on demand. At a high level, PrivGED proceeds through two phases: (i) Secure collection of decentralized social graph data. In this phase, each user $U_i$’s local view data $A[i,:]$ regarding the social graph is collected by the cloud servers in encrypted form so as to form the matrix for eigendecomposition, through sparsity-aware and privacy-preserving techniques developed in PrivGED. (ii) Secure eigendecomposition. After the encrypted adjacency matrix is adequately formed at the cloud, the cloud servers collaboratively perform secure eigendecomposition, through a suite of customized secure protocols developed in PrivGED.

4.2 Threat Model
Along the system workflow in PrivGED, we consider that the primary threats come from the cloud entity empowering the eigendecomposition-based graph analytics service. Similar to most prior security designs in the two-server setting [41], [50], [51], we assume a semi-honest and non-colluding adversary model where $CS_{\{1,2\}}$ honestly follow the protocol specification of PrivGED, but may individually attempt to learn private information about individual users’ social connections in the decentralized social graph and the analytics result. Note that each user’s private social connections with other entities in the graph are reflected by the non-zero elements in her local view vector, as introduced above. So, for the privacy of individual users, PrivGED aims to conceal the sensitive information regarding the non-zero elements in each user $U_i$’s local view vector $A[i,:]$, which includes the positions, values, and number.

5 Secure Collection of Decentralized Social Graph Data
5.1 Overview
In this phase, each user $U_i$ provides his local view data $A[i,:]$ in protected form to the service. For high efficiency, PrivGED resorts to the lightweight ASS technique for encryption of the elements in $A[i,:]$. A simple method is to have each $U_i$ directly apply ASS over his complete $A[i,:]$ of length $N$. However, such simple method is clearly inefficient due to the sparsity of the (decentralized) social graph. According to Facebook’s statistics [14], on average a user has 130 friends in a social network, which is far less than $N$ (e.g., hundreds of thousands). This leads to high sparsity, indicating that the complete $A[i,:]$ will be filled with many zeros. So the simple method will incur significant cost on the user side as well as pose unnecessary workload in the subsequent secure eigendecomposition process.

To remedy this, a plausible approach is to leverage sparse representation. Specifically, $U_i$ applies ASS only over each nonzero element at location $j$ (denoted as $\{(i,j, A[i,j])\}$), and sends $\{(i,j, [A[i,j]]^\dagger)\}$ to $CS_{\{1,2\}}$. Such approach yet leads to prominent privacy leakages: (i) The number of nonzero elements (i.e., the degree) indicates the number of $U_i$’s friends, which can be used in inferring $U_i$’s privacy [15]. (ii) The presence of a nonzero element $(i,j, [A[i,j]]^\dagger)$ implies the presence of the relationship between $U_i$ and $U_j$, which can also be used in inference attacks [15]. (iii) If the social graph is unweighted (i.e., each element is 0 or 1), the presence of a nonzero element $(i,j, [A[i,j]]^\dagger)$ implies $A[i,j] = 1$, revealing the data to $CS_{\{1,2\}}$.

Therefore, the challenge here is how to preserve the benefits of sparsity while protecting the privacy of individual user’s social relationships. Meanwhile, the effectiveness of subsequent eigendecomposition should not be affected.

Our key insight is to delicately trade off (node degree) privacy for efficiency, by having $U_i$ blend in some dummy edges with zero weights at random empty locations in $A[i,:]$, and apply ASS over the weights of both true and dummy edges, inspired by [8]. Under ASS, even encrypting the same (zero) value multiple times will result in different shares (ciphertexts) indistinguishable from uniformly random values.

Therefore, the dummy edges cannot be distinguished from the true edges, as well as do not affect the effectiveness of the subsequent secure eigendecomposition process.

What remains challenging here is how to appropriately set the number of dummy edges so as to delicately balance the trade-off between efficiency and privacy. Too many dummy edges will impair the sparsity and increase the system overhead, while too few dummy edges will result in weaker privacy protection. Therefore, a tailored security design is demanded to provide a theoretically sound approach by which $U_i$ can select the adequate number of dummy edges to achieve a balance between efficiency and privacy. Our main idea is to rely on LDP so as to make the leakage about the node degrees differentially private. In what follows, we start with some basic approaches and discuss their limitations. Then we present our tailored solution.

5.2 Basic Approaches
A basic approach based on LDP can work as follows. As aforementioned, each $U_i$ draws a noise $n_i$ from the discrete Laplace distribution, and then blends $n_i$ dummy edges with zero values at random empty locations in $A[i,:]$ followed by encrypting the weights of both true and dummy edges by ASS. In this way, differential privacy guarantee on the node degree can be achieved, and the presence of edges is disguised as well [8]. Here, the sensitivity $\Delta$ needs to be set to $\Delta = (d_{\text{max}} - d_{\text{min}})$, where $d_{\text{max}}$ and $d_{\text{min}}$ are the possible maximum and minimum degree in the decentralized social graph, respectively. The above basic approach, however, would
result in a very large $\Delta$ (up to $N$ in theory) to be applied, leading to a large number of dummy edges to be added and heavily impairing the sparsity. To illustrate why this is the case, Fig. 2 shows the probability density function of the discrete Laplace distribution with different $\Delta$. It is revealed that a larger $\Delta$ leads to the shape of the density function being more uniform. This indicates that the larger the sensitivity $\Delta$ is, the larger probability that $U_i$ draws a large $|n_i|$ will be. In contrast, a small $\Delta$ will make the probability density function concentrated (e.g., $\Delta = 50$ in Fig. 2), which means that $U_i$ will draw a small $|n_i|$ with a large probability.

In order to achieve better efficiency, an alternative approach as proposed by [8] is to use a bin-based mechanism that provides bin-wise differential privacy rather than graph-wise differential privacy. The main idea is to partition users into several bins, each of which contains an approximately equal number of users whose degrees are within a small interval $[d_{p}, d_{q}]$. As such, all users in the same bin can use a much smaller sensitivity $\Delta = d_{q} - d_{p}$. We note that while the idea proposed by Sagar et al. [8] is useful, their design to instantiate such idea is not satisfactory and has crucial downsides. First, it requires some (sampled) users to open their degrees to estimate the degree histogram of the social graph so as to split users into bins, harming the privacy of the sampled users. Second, to avoid edge deletion when the drawn noise is negative, it lets all users add a large offset to the drawn noise, making a notable impact on the sparsity. In addition, no formal analysis regarding the differential privacy guarantee is provided.

### 5.3 Our Approach

Based on the aforementioned idea of bin-based LDP, we design a new protocol for secure collection of decentralized social graph data, which does not expose any users’ exact node degree information, as opposed to PrivateGraph [8]. Meanwhile, our protocol does not require users to add a large offset to the noises, and thus is much more advantageous in maintaining the sparsity. We also later provide formal analysis on the differential privacy guarantee. Our protocol for secure collection of decentralized social graph data is comprised of the following ingredients: (i) secure degree histogram estimation, (ii) secure binning map generation, and (iii) local degree data encryption.

#### 5.3.1 Secure Degree Histogram Estimation

It is challenging for $CS_{[1,2]}$ to obliviously estimate the degree histogram without knowing any users’ degrees. In particular, given a public degree $d_t$ and a specific user’s degree $d_j$, $CS_{[1,2]}$ need to efficiently check whether $d_j = d_t$ with both $d_j$ and the checking result being encrypted. To counter this challenge, we devise a tailored construction based on function secret sharing [39], a recently developed cryptographic primitive that allows the generation of compact function shares and secure evaluation with the function shares. In particular, we take advantage of distributed point function (DPF) [17], an FSS scheme for the point function $f_{a,b}(x)$ which outputs $\beta$ if $x = \alpha$ and otherwise outputs 0. Formally, a two-party DPF, parameterized by a finite Abelian group $G$ ($G = \mathbb{Z}_q$ in our design), consists of the following algorithms: (i) $(k_1, k_2) \leftarrow \text{Gen}(1^\lambda, \alpha, \beta)$: given a security parameter $1^\lambda$, a string $\alpha \in \{0, 1\}^\lambda$, and a value $\beta \in G$, outputs two succinct DPF keys representing the function shares. (ii) $(f_{a,b}(x))_1 = \text{Eval}(k_1, x)$: given a DPF key $k_1$ and an input $x \in \{0, 1\}^\lambda$, outputs $(f_{a,b}(x))_1 \in G$.

Building on top of DPF, we develop a technique for secure degree histogram estimation, as presented in Algorithm 3. As directly engaging all users for building the degree histogram will lead to high performance overhead, PrivGED adopts a sampling strategy following [8], where $S$ users, denoted by $\{SU_j\}_{j \in [1, S]}$, are randomly sampled to participate. Each $SU_j$ generates DPF keys based on his degree $d_j$ (line 1 in Algorithm 3), where $\alpha$ and $\beta$ are set to $d_j$ and 1 respectively. Each $SU_j$ then sends $k_{j,1}$ to $CS_1$ and $k_{j,2}$ to $CS_2$. Utilizing the DPF keys $\{k_{j,t}\}_{j \in [1, S]}$, $CS_t$ ($t \in \{1, 2\}$) can evaluate $\text{Eval}(k_{j,t}, i)$ for all possible degrees $i \in [1, d_{\text{max}}]$. By summing these evaluation results (line 6 in Algorithm 3), $CS_t$ can obtain exactly the encrypted number $\langle D_i \rangle_t$ of sampled users whose degree is equal to $i$. Correctness holds since

$$\langle D_i \rangle_1 + \langle D_i \rangle_2 = \sum_{j=1}^{S} \text{Eval}(k_{j,1}, i) + \sum_{j=1}^{S} \text{Eval}(k_{j,2}, i)$$

$$= \sum_{j=1}^{S} \left[ \text{Eval}(k_{j,1}, i) + \text{Eval}(k_{j,2}, i) \right]$$

$$= \sum_{j=1}^{S} 1 \{d_j = i\}.$$
Algorithm 4. Secure Binning Map Generation

**Input:** The encrypted estimated degree histogram \([D_i]^A, i \in [1, d_{max}]\), the number of bins \(B\), and the number of sampled users \(S\).

**Output:** The encrypted binning map \([\text{Inter}]^B \in \{[0]^B, [1]^B\}^{d_{max}}\).

1. Calculate the bin size \(sizeB = \frac{S}{B}\).
2. Initialize the accumulator \(\text{accu}^A = 0\).
3. for \(i \in [1, d_{max}]\) do
   4. \(\text{accu}^A + = [D_i]^A\).
   5. \(\text{Inter}[i]^B = \text{accu}^A \geq sizeB^B\).
   6. \(\text{accu}^A = \text{Inter}[i]^B \cdot \text{accu}^A\).
4. end for

5.3.2 Secure Binning Map Generation

With the encrypted degree histogram produced on the cloud side, we now introduce how to enable \(CS_{[1,2]}\) to generate an encrypted binning map, based on which each user can get the sensitivity value for his use in drawing the noise. The binning strategy underlying our secure design follows the prior art [8], leading to each bin containing an approximately equal number of users whose degrees are within a small interval. It is noted that unlike our design operating in the ciphertext domain, [8] operates in plaintext domain with exposed raw degrees of sampled users for building the histogram and producing the binning map.

Our secure design is shown in Algorithm 4, which inputs the encrypted degree histogram \([D_i]^A, i \in [1, d_{max}]\), the number \(B\) of bins, and the number \(S\) of the sampled users, and then outputs the encrypted binning map \([\text{Inter}]^B \in \{[0]^B, [1]^B\}^{d_{max}}\) (i.e., an encrypted bit-string under binary secret sharing), where \(i\) indicates the boundary of a bin. For example, given \(d_{max} = 10, Inter = 0001000001\) indicates that the users are partitioned into two bins: users whose degree \(\in [1, 4]\) and users whose degree \(\in [5, 10]\). After obliviously computing \([\text{Inter}]^B, CS_{[1,2]}\) send it to all users, and then each of them can judge to which bin he belongs based on his own degree locally.

As shown in Algorithm 4, \(CS_{[1,2]}\) first calculate the public bin size \(sizeB\) based on \(B\) and \(S\) (line 1), and then initialize an accumulator \(\text{accu}^A\). After that, \(CS_{[1,2]}\) add \([D_i]^A\) \((i \in [1, d_{max}])\) to \(\text{accu}^A\) in turn (line 4). After each addition, \(CS_{[1,2]}\) obliviously evaluate \(\text{accu}^A \geq sizeB^B\) and add the comparison result (in binary secret sharing) to \([\text{Inter}]^B\).

Specifically, if \(\text{accu} \geq sizeB\), \(\text{Inter}[i] = 1\), indicating a bin boundary; otherwise, \(\text{Inter}[i] = 0\). After that, based on \([\text{Inter}]^B\), \(CS_{[1,2]}\) obliviously evaluate whether to reset the accumulator \(\text{accu}^A\) or not. Specifically, if \(\text{Inter}[i] = 1\), a bin boundary appears, and thus \(\text{accu}\) needs to be reset to 0 for the next bin; otherwise, \(\text{accu}\) remains unchanged. This step is given in line 6, where “!” denotes not operation which can be achieved by letting one of \(CS_{[1,2]}\) flip its share \([\text{Inter}[i]^B\) or \([\text{Inter}[i]^B\) locally. Finally, \(CS_{[1,2]}\) output \([\text{Inter}]^B \in \{[0]^B, [1]^B\}^{d_{max}}\).

In Algorithm 4, the addition operations are easily supported with secret sharing, but the operation \(\text{accu} \geq sizeB^B\) is not directly supported. What we need here essentially is secure comparison in the secret sharing domain. From the very recent works [18], [43], we identify two primitives suited to allow secure comparison in the secret sharing domain, and introduce two approaches accordingly to allow the realization of secure binning map generation. The first approach is based on FSS [18], which is more suited for high-latency network scenarios because it requires minimal rounds of interactions (at the cost of more local computation). The second approach is based on ASS [43], which requires a small amount of local computation but higher online communication and round complexities, fitting better into low-latency network scenarios.

**FSS-Based Approach.** We identify the state-of-the-art construction of FSS-based secure comparison from [18], which is referred to as distributed comparison function (DCF). DCF is an FSS scheme for the function \(g\alpha(x)\) which outputs \(\beta\) if \(x < \alpha\) and 0 otherwise. Formally, a two-party DCF, parameterized by two finite Abelian groups \(G', G''\), consists of the following algorithms: (i) \((k_1, k_2, r_1, r_2) \leftarrow \text{Gen}(\alpha, \beta)\), which takes as input a security parameter \(\alpha\), \(\beta \in G''\), and \(\beta \in G''\), and outputs two keys \(k_1, k_2\) and two random values \(r_1, r_2 \in G''\) \((r_1 + r_2 = \text{random})\), each for one party. (ii) \(g\alpha(x) \leftarrow \text{Eval}(k_1, x, r_1 + r_2 = \text{random})\), which takes as input a key \(k_1\) and a (masked) input \(x + r_1 = \text{random}\) in \(G''\), and outputs \(g\alpha(x)\) in \(G''\). The evaluation process in DCF only requires one round of offline communication, in which the two parties send \((x), r_1, r_2 \in \{1, 2\}\) to each other to reveal \(x + r_1\) without leaking \(x\). The security of DCF states that if an adversary only learns one of \(k_1, r_1, r_2 \in \{1, 2\}\), it learns no information about the private input \(x\) and output \(g\alpha(x)\).

We now show how PrivGED builds on the DCF for secure evaluation of \(\text{accu}^A \geq sizeB^B\). To use the DCF in PrivGED, the related public parameters can be set as follows: \(\alpha = sizeB, \beta = 1, G' = Z_{p'}\), and \(G'' = Z_2\). With these parameters, the DCF keys \(k_1, k_2\) and the random values \(r_1, r_2 \in Z_{p'}\) can be prepared offline and distributed to \(CS_{[1,2]}\) respectively. Note that such offline work can be done by a third-party server in practice [18]. For the secure online evaluation of \(\text{accu}^A \geq sizeB^B\), \(CS_{[1,2]}\) first exchanges \(\text{accu}^A\) and \(r_1, r_2\) to each other to reveal \(\text{accu} + r_1\), and then evaluates \(\text{Eval}(k_1, \text{accu} + r_1)\), which will output \((1)^A\), if \(\text{accu} < sizeB\) and \((0)^A\) otherwise. However, PrivGED requires \(CS_{[1,2]}\) to output \([1]^B\) if \(\text{accu} \geq sizeB\), and thus PrivGED further lets one of \(CS_{[1,2]}\) flip its share locally.

**ASS-Based Approach.** This approach is based on the idea of secure bit decomposition in the secret sharing domain [43]. Specifically, given two fixed-point integers \(X, Y \in Z_{p}\) under two’s complement representation, the most significant bit (MSB) of \(X - Y\) can indicate the relationship between \(X - Y\) and 0. Namely, if \(X - Y < 0\), \(msb(X - Y) = 1\) and otherwise \(msb(X - Y) = 0\). Secure extraction of the MSB in the secret sharing domain can be achieved via secure realization of a parallel prefix adder (PPA) [43], as illustrated in Fig. 3. It is observed that only XOR and AND operations need to be performed in the secret sharing domain. PrivGED leverages the construction from [43], which takes the secret sharings of two values \(X\) and \(Y\) as input, and outputs \(\text{msb}(X - Y)^B\). With that construction, \(CS_{[1,2]}\) are able to obtain \(\text{msb}(\text{accu} - sizeB)^B\), i.e., \([0]^B\) if \(\text{accu} \geq sizeB\) and \([1]^B\) otherwise. However, what is needed in PrivGED instead is that \(CS_{[1,2]}\) output \([1]^B\) if \(\text{accu} \geq sizeB\) and \([0]^B\) otherwise. PrivGED further lets one of \(CS_{[1,2]}\) flip its share \(\text{msb}(\alpha)\), \((t \in \{1, 2\})\) locally.
It is noted that the output $\|\text{Inter[}\cdot]\|B = \|\text{accu}\|^A \geq \text{sizeB}^B$ of the above two approaches are in binary secret sharing. However, for the computation in line 6 of Algorithm 4, the binary secret-shared value needs to be multiplied by an arithmetic secret-shared value, i.e., $\|\text{Inter[}\cdot]\|B \cdot \|\text{accu}\|^A$. We note that this can be achieved based on an existing technique from [52], through the following steps: (i) $CS_1$ randomly generates $r \in Z_p$ ($k$ is the value length of accu), and sends two messages to $CS_2$: $m_0 := (b \oplus \langle \text{accu} \rangle_1 - r, b \in \{0, 1\}$. (ii) $CS_2$ chooses $m_0$ if $\|\text{Inter[}\cdot]\|_2 = 0$, otherwise $CS_2$ chooses $m_1$. Therefore, the share held by $CS_2$ is $m_\text{Inter[}\cdot]_2 = \langle \text{accu} \rangle_1 - r$ and the share held by $CS_1$ is $r$. (iii) For the other secret share $\langle \text{accu} \rangle_2$, $CS_2$ acts as the sender and $CS_1$ acts as the receiver to perform step 1, 2 again. At the end of the process, $CS_1$ holds $\langle \text{Inter[}\cdot] \cdot \langle \text{accu}\rangle, t \in \{1, 2\}$. The resulting secret sharing of the binning map $\|\text{Inter[}\cdot]\|B \in \{0\}^B, \{1\}^B$ can then be sent to all users for use.

5.3.3 Local View Data Encryption

With the binning map $\text{Inter}$ recovered at the user side, each $U_i$ then encrypts his local view data, as shown in Algorithm 5. The main problem is that the drawn noise $n_i$ could be negative (recall Fig. 2), which means that $U_i$ should delete some edges. Obviously, this will seriously impair the accuracy of subsequent secure eigendecomposition. To tackle this problem, PrivGED lets $U_i$ truncate $n_i$ to 0 inspired by [38], i.e., $n_i' = \max(n_i, 0)$. In Section 7.1, we will formally prove that all users’ degrees still satisfy $(\epsilon, \delta)$-LDP although they may truncate $n_i$ to 0. Let $\mathcal{L}$ denote the set of locations of both true edges and dummy edges, and $\{(i, j, A[i, j])\}_{i \in \mathcal{L}}$, the resulting data of $U_i$ after adding dummy edges with zero weights. Finally, $U_i$ applies ASS over each $A[i, j]$ and sends $\{(i, j, A[i, j])\}_{i \in \mathcal{L}_i}$ to the cloud.

6 SECURE EIGENDECOMPOSITION

6.1 Overview

The encrypted data collected from the users forms the encrypted adjacency matrix under sparse encoding for eigendecomposition at the cloud. We denote it by $[A^*]^A := \{(i, j, [A[i, j]]^A)\}_{i \in [1, N], j \in \mathcal{L}}$, where $\mathcal{L}$ is a multi-set that contains all the locations from $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_N$. Upon deriving $[A^*]^A$, PrivGED needs to enable $CS_{[1, 2]}$ to obliviously perform eigendecomposition on $[A^*]^A$, producing the encrypted top-$k$ eigenvalues/eigenvectors.

First, PrivGED introduces techniques (Section 6.2) to enable $CS_{[1, 2]}$ to obliviously transform $[A^*]^A$ to $[A]^A$ (in dense encoding) of a much smaller size $M \times M$, with respect to the Arnoldi method (for general (possibly non-symmetric) matrices) and Lanczos method (for symmetric matrices) as introduced before in Section 3.1. Then, PrivGED provides secure realizations of the QR algorithm so as to achieve secure eigendecomposition over the $[A]^A$ to produce encrypted top-$k$ eigenvalues/eigenvectors. We give a basic design (Section 6.3) for the secure QR algorithm as a starting point, followed by a delicate optimized design (Section 6.4) that further achieves a performance boost.

6.2 Secure Matrix Dimension Reduction

We first consider how to enable $CS_{[1, 2]}$ to obliviously execute the Arnoldi method on $[A]^A$ to output $[A]^A$. Looking into the operations in Algorithm 1, we note that the operations in line 1-7, which are comprised of addition and multiplication, can be naturally and securely realized in the arithmetic secret sharing domain by $CS_{[1, 2]}$. However, it remains challenging for $CS_{[1, 2]}$ to obliviously perform the operations in lines 8, 9, because square root and division are not directly supported in the secret sharing domain.

Our solution is to approximate the square root and division operations using basic operations (i.e., $+, \times$), so that they can be securely supported in the secret sharing domain. For secure square root $\sqrt{x}$, inspired by the very recent work [53], PrivGED utilizes a roundabout strategy to approximate the inverse square root $1/\sqrt{x}$ by the iterative Newton-Raphson algorithm [54]

$$y_{n+1} = \frac{1}{2}y_n(3 - xy_n^2),$$

which will converge to $y_n \approx \frac{1}{\sqrt{x}}$. Obviously, both subtraction and multiplication operations are naturally supported in the secret sharing domain. After that, PrivGED lets $CS_{[1, 2]}$ multiply $\|y_n\|^A$ by $\|x\|^A$ to derive $\|\sqrt{x}\|^A$. For secure division $\left[\frac{a}{b}\right]^A$ in the secret sharing domain, we note that the main challenge is to compute the reciprocal $\left[\frac{1}{a}\right]^A$. However, we also observe that the reciprocal of division in Algorithm 1 is $\left[\frac{1}{a}\right]^A$, which is exactly the inverse square root computed in line 8. Therefore, PrivGED lets $CS_{[1, 2]}$ directly perform the operation in line 9 by multiplying the inverse square root computed in line 8 by $\|q_k\|^A$.

We give our construction for securely realizing the Arnoldi method in Algorithm 6. Regarding the Lanczos method, we note that the operations required to be securely
Algorithm 6. Secure Arnoldi Method

Input: $[A]^4$ and the target dimension $M$.

Output: $[A]^4$ with dimension $M \times M$ and $[P]^4$.

1. Start with an arbitrary vector $[p_1]^4$ with $L^2$ norm 1.
2. for $k \in [2, M]$ do
3. \begin{align*}
    [p_k]^4 &= ([A]^4 [p_{k-1}]^4)^T.
\end{align*}
4. for $j \in [1, k-1]$ do
5. \begin{align*}
    [A][j, k-1]^4 &= [p_j]^4 \cdot [p_k]^4.
\end{align*}
6. \begin{align*}
    [p_k]^4 &= [p_k]^4 - [A][j, k-1]^4 \cdot [p_j]^4.
\end{align*}
7. end for
8. \begin{align*}
    [x_k]^4 &= [p_k]^4 \cdot [p_k]^4.
\end{align*}
# Calculate the inverse square root of Eq. (4):
9. for $n \in [1, \Omega]$ do
10. \begin{align*}
    [y_{n+1}]^4 &= \frac{1}{2} \cdot [y_n]^4 \cdot (3 - [x_k]^4 \cdot [y_n]^4 \cdot [y_n]^4).
\end{align*}
11. end for
12. \begin{align*}
    [A][k, k-1]^4 &= [x_k]^4 \cdot [y_{n+1}]^4.
\end{align*}
13. \begin{align*}
    [p_k]^4 &= [y_{n+1}]^4 \cdot [p_k]^4.
\end{align*}
14. end for
15. \begin{align*}
    [P]^4 &= [p_1]^4, \ldots, [p_M]^4.
\end{align*}

supported are identical to the Arnoldi method, so we omit the algorithm description for the secure Lanczos method. It is noted that the encrypted adjacency matrix $[A]^4 := \{(i, j, [A][i, j]^4)\}_{i,j \in [1, N], j \neq i}$ under sparse encoding can significantly save the cloud-side cost. For example, the secure matrix-vector product between $[A]^4$ and $[v]^4$ can be efficiently performed by only securely multiplying the elements $[A][i, j]^4$ and the corresponding $[v[j]^4$. Obviously, the number of multiplications in this example is independent of the number of columns in the original complete $A$ and only depends on the number of rows and the number of nonzero elements in $A$.

6.3 Secure QR Algorithm

We now introduce, as a basic design, how to enable $CS_{[1,2]}$ to obliviously calculate the complete encrypted eigenvalues/eigenvectors of $[A]^4$, through a protocol for securely realizing the widely used QR algorithm. The QR algorithm works in an iterative manner, consisting of a series of QR decomposition. Formally, given matrix $L$ and $T_0 = L$, in the $k$th ($k \in [1, K]$) iteration, on input $T_{k-1}$, we compute QR decomposition $T_{k-1} = Q_{k-1} R_{k-1}$, where $Q_{k-1}$ is an orthogonal matrix (i.e., $Q^T = Q^{-1}$), $R_{k-1}$ is an upper Hessenberg matrix, and then output $T_k = R_{k-1} Q_{k-1}$. At the end of QR algorithm, the diagonal elements of $T_K$ are $L$'s eigenvalues and $S = Q_1 \cdots Q_K$ are $L$'s eigenvectors.

In PrivGED, we perform the QR decomposition utilizing the Givens rotations [55]. Formally, given an $M \times M$ upper Hessenberg matrix $T_{k-1}$, we create the orthogonal Givens rotation matrix $G_i$, $i \in [1, M - 1]$

\begin{align*}
    G_i = \begin{bmatrix}
    1 & \cdots & 0 & \cdots & 0 \\
    \vdots & \ddots & \vdots & \ddots & \vdots \\
    0 & \cdots & c_i & -s_i & 0 \\
    0 & \cdots & s_i & c_i & 0 \\
    \vdots & \ddots & \vdots & \ddots & \ddots \\
    0 & \cdots & 0 & 0 & 1
\end{bmatrix},
\end{align*}

where

\begin{align}
    c_i &= \frac{H(i)[i, i]}{\sqrt{H(i)[i, i]^2 + H(i)[i + 1, i]^2}}, \\
    s_i &= \frac{H(i)[i + 1, i]}{\sqrt{H(i)[i, i]^2 + H(i)[i + 1, i]^2}},
\end{align}

and $H(i) = G_{i-1}^T H(i - 1) H(1) = T_{k-1}$. At the end of this iteration, $T_k = G_{M-1}^T \cdots G_1^T T_{k-1} G_1 \cdots G_{M-1}$, and $Q_{k-1} = G_1 \cdots G_{M-1}$.

Next, we consider how to securely and efficiently perform the above process in secret sharing domain. The main challenging part of the secure QR algorithm is to let $CS_{[1,2]}$ obliviously calculate square root and division in secret sharing. This can be effectively addressed based on the techniques introduced in Section 6.2. In addition, we note that securely realizing the QR Algorithm consists mainly of matrix multiplications in the secret sharing domain.

A straightforward method is to perform secret-shared value-wise multiplication, which would require $M^3$ multiplications and online communication of $2 M^3$ ring elements. We note that a better choice is to work with vectorization [16], where the Beaver triples needed for secure multiplication are represented in a vectorized form, i.e., $[X, Y, Z]$, where $Z = XY$, and $X$ and $Y$ play the role in masking the input matrices during secure multiplication. Based on such vectorization trick, the online communication is reduced to only $2 M^2$ ring elements. PrivGED chooses to work with vectorization for secret-shared matrix multiplication.

Algorithm 7 presents our basic design for securely realizing the QR algorithm. With $[A]^4$ as input, it outputs $[T_K]^4$.
Fig. 4. Illustration of the optimized QR decomposition on a $4 \times 4$ upper Hessenberg matrix utilizing a series of Givens rotations.

and $[S]^{A}$. Note that the top-k diagonal elements of $T_{K}$ represent the desired top-k eigenvalues of $A$. For the secret-shared eigenvectors $[S]^{A}$ of $[X]^{A}$, they can be obliviously transformed to the corresponding eigenvectors of $[A]^{A}$ by $[V]^{A} = [P]^{A}[S]^{A}$, where $[P]^{A}$ is the output matrix of our secure Arnoldi method (or Lanczos method).

6.4 Optimizing the Secure QR Algorithm

We now show how to further optimize the basic design introduced above to achieve an efficiency boost. Our key insight is to first reformulate the plaintext QR algorithm by simplifying the Givens rotation matrix (i.e., Eq. (3)), and then identify the correlated multiplications and extract repetitive multiplicands to further save the cost. We first simplify the Givens rotation matrix based on the observation: in each Givens rotation $G_{i}^{j}H(i)$ or $H(i)G_{i}$, only the $i$th and $(i + 1)$th rows of $H(i)$ are updated (recall Eq. (3)). Therefore, to save computation, we can reduce the Givens rotation matrix $G_{i}$ from Eq. (3) to $\begin{bmatrix} c_{i} & -s_{i} \\ s_{i} & c_{i} \end{bmatrix}$ (denoted as $g_{i}$).

Fig. 4 illustrates the optimized QR decomposition on a $4 \times 4$ upper Hessenberg matrix utilizing a series of Givens rotations. Similarly, when calculating $[S]^{A} = [S]^{A}[G]^{A}$ (i.e., line 15 in Algorithm 7), we can also reduce $[G]_{i}^{A}$ to $[g]_{i}^{A}$.

After the above simplification, we have a new observation: $[g]_{i}^{A}$ will be used repeatedly in several matrix multiplications. In a recent study [56], Kelkar et al. point out that when one of the multiplicands is a number of (secret-shared) matrix multiplications stays constant, the constant multiplicant can be masked and then opened only once so as to achieve cost savings on communication. For example, suppose we need to multiply $[U]^{A}$ with $[V_{1}]^{A}$, $[V_{2}]^{A}$, $\ldots$, $[V_{k}]^{A}$ in the secret sharing domain. We only need a single matrix sharing $[X]^{A}$ for $[U]^{A}$, rather than $k$ matrices as in directly using the Beaver’s trick in vectorized form. Therefore, PrivGED can have $CS_{[1,2]}$ only mask $[g]_{i}^{A}$ once for all secret-shared multiplications. In addition, the masked $[g]_{i}^{A}$ can be directly achieved by letting $CS_{[1,2]}$ locally transpose the masked $[g]_{i}^{A}$.

Remark. Our basic design for the secure QR algorithm requires $CS_{[1,2]}$ to online communicate $4K(M - 1)M^{2} + 2K(M - 1)M^{2} = 6K(M - 1)M^{2}$ ring elements. In contrast, our optimized secure QR algorithm only requires $CS_{[1,2]}$ to online communicate $K(M - 1)(4M + 4) + 2K(M - 1)M = K(M - 1)(6M + 4)$ ring elements. As later shown in our experiments, the optimized secure QR algorithm can save up to 99% online communication as well as 9% computation as compared to the basic design.

6.5 Complexity Analysis

The design of PrivGED is comprised of several subroutines: 1) secure degree histogram estimation $secEst$; 2) secure binning map generation $secBin$; 3) local view data encryption $secEnc$; 4) secure matrix dimension reduction $secRedu$; 5) secure QR algorithm $secQR$; 6) optimized secure QR algorithm $secOptiQR$. Therefore, we analyze their theoretical performance complexities as follows.

1) $secEst$. Algorithm 3 describes $secEst$ running between the cloud servers and some sampled users, which requires each of $CS_{[1,2]}$ to evaluate $S$ sampled users’ DPF keys for all possible degrees $i \in [1, d_{max}]$, and thus its computation complexity is $O(S \cdot d_{max})$. $secEst$ does not require $CS_{[1,2]}$ to communicate with each other, but requires each of the $S$ sampled users to send a pair of DPF keys to $CS_{[1,2]}$, and thus its communication complexity is $O(S \cdot K)$, where $K$ is the size of DPF keys.

2) $secBin$. Algorithm 4 describes $secBin$ running at the cloud side, which requires $CS_{[1,2]}$ to securely evaluate for all possible degrees $i \in [1, d_{max}]$, and thus its computation complexity is $O(d_{max})$. For communication, it is noted that we provide two secure comparison approaches (FSS-based and ASS-based). The FSS-based approach only requires $CS_{[1,2]}$ to send a masked value with bit length $l$ to each other, but the ASS-based one requires $CS_{[1,2]}$ to communicate $O(l \cdot \log l)$ bits. Therefore, the communication complexity of FSS-based $secBin$ is $O(l \cdot d_{max})$, and that of ASS-based $secBin$ is $O(l \cdot \log l \cdot d_{max})$.

3) $secEnc$. Algorithm 5 describes $secEnc$ running at the users side, which requires each user $U_{i}$ to encrypt his local view with length $|L_{i}|$, and thus its computation complexity is $O(|L_{i}|)$ and communication complexity is $O(|L_{i}| \cdot l)$, where $l$ is the bit length of weight $A[i, j]$.

4) $secRedu$. Algorithm 6 describes $secRedu$ running at the cloud side, whose computation complexity is $O(N \cdot M)$, where $N$ is the width (or height) of the original matrix and $M$ is the width (or height) of the matrix after dimension reduction. Its communication complexity is $O(N \cdot M \cdot l)$. It is noted that the asymptotic computational and communication complexity of secure Lanczos method are also $O(N \cdot M)$ and $O(N \cdot M \cdot l)$ because its algorithm is similar to secure Arnoldi method (recall Algorithms 1 and 2).

5) $secQR$. Algorithm 7 describes $secQR$ running at the cloud side, whose computation complexity is $O(K \cdot M^{3})$ and communication complexity is $O(K \cdot M^{3} \cdot l)$ where $K$ is the number of QR decomposition.

6) $secOptiQR$. It is noted that the asymptotic computation complexity of $secOptiQR$ is identical $secQR$, i.e., $O(K \cdot M^{3})$, but its communication complexity is only $O(K \cdot M^{3} \cdot l)$ (recall the Remark in Section 6.4).
7 Privacy and Security Analysis

We now provide analysis regarding the protection PrivGED offers for the users. Note that PrivGED aims to conceal the sensitive information regarding the non-zero elements in each user’s local view vector, which includes the positions, values, and number (i.e., the degree). As PrivGED builds on LDP to perturb the exact degree of users and cryptographic techniques to safeguard the confidentiality of data values, we present our analysis in two parts. The first part is privacy analysis, which is to prove the differential privacy guarantee PrivGED offers for users. Note that blending in dummy edges in each user’s local view vector as per the differential privacy mechanism obfuscates not only the number of non-zero elements but also their positions. The second part is security analysis, where we follow the standard simulation-based paradigm to prove data confidentiality against the cloud servers. Similar treatment also appears in prior works [38], [57] using both DP and cryptography.

7.1 Privacy Analysis

Theorem 1. PrivGED can achieve \((\epsilon, \delta)\)-LDP for the degree of users in the same bin according to Definition 1.

Proof. Given the degrees \(d_i, d_j\) of \(U_i\) and \(U_j\), who are in the same bin, and the sensitivity of the bin is \(\Delta\). If both the noises drawn from \(Lap(\epsilon, \delta, \Delta)\) by \(U_i\) and \(U_j\) are non-negative, the probability of them outputting the same noisy degree \(\tilde{d}\) is bounded by

\[
\frac{Pr[\tilde{d} = d_i]}{Pr[\tilde{d} = d_j]} = \frac{Pr[\tilde{d} - d_i]}{Pr[\tilde{d} - d_j]} = \frac{\epsilon^{i/2} \cdot |d_i - d_j|}{\epsilon^{j/2} \cdot |d_i - d_j|} = \frac{\epsilon^{i/2}}{\epsilon^{j/2}} \cdot \frac{|d_i - d_j|}{|d_i - d_j|} = \frac{\epsilon^{i/2}}{\epsilon^{j/2}} \leq \epsilon^\delta.
\]

In addition, we note that the probability to draw a negative noise from \(Lap(\epsilon, \delta, \Delta)\) is [38]

\[
Pr[x < 0] = \sum_{x=-1}^{\infty} \frac{\epsilon^{|x|} - 1}{\epsilon^{x+1}} = \frac{\epsilon^{|x|}}{\epsilon^{x+1}} = \frac{\epsilon^{-x}}{\epsilon^{-x+1}} = \frac{\epsilon^{x}}{\epsilon^{x+1}}.
\]

Given Eq. (1) and \(\Delta \geq 0\), we have

\[
Pr[x < 0] = 1 - (1 - \delta)^\frac{1}{2} \leq \delta,
\]

which means that the maximum probability of truncation is \(\delta\). Therefore, with \(1 - \delta\), the probability to output the same noisy degree \(\tilde{d}\) from \(U_i\) and \(U_j\) is bounded by \(\epsilon^\delta\), which satisfies \((\epsilon, \delta)\)-LDP in Definition 1.

7.2 Security Analysis

We follow the standard simulation-based paradigm to analyze the security of PrivGED. In this paradigm, a protocol is secure if the view of the corrupted party during the protocol execution can be generated by a simulator given only the party’s input and legitimate output. Let \(\Pi\) denote the protocol in PrivGED for secure eigendecomposition. Recall that the cloud servers \(CS_{1,2}\) neither provide input nor obtain output in PrivGED.

Definition 3. Let \(\pi^{C_i}_{CS_i}\) denote each \(CS_i\)’s view during the execution of \(\Pi\). We say that \(\Pi\) is secure in the semi-honest and non-colluding setting, if for each \(CS_i\) there exists a PPT simulator which can generate a simulated view such that \(Sim_{CS_i} \approx \pi^{C_i}_{CS_i}\). That is, the simulator can simulate a view for \(CS_i\), which is indistinguishable from \(CS_i\’s\) view during the execution of \(\Pi\).

Theorem 2. PrivGED is secure according to Definition 3.

Proof. It is noted that PrivGED invokes the subroutines \(secEst, secBin, secEnc, secRedu\) and \(secQ\) in order. If the simulator for each subroutine exists, then our complete protocol is secure [58], [59], [60]. Since the roles of \(CS_{1,2}\) in these subroutines are symmetric, it suffices to show the existence of simulators for \(CS_1\).

1) Simulators for \(CS_1\) in \(secEnc, secRedu\) and \(secQ\). It is easy to see that the simulators for \(CS_1\) in these subroutines must exist, because they only require basic addition and multiplication over additive secret shares (which can be simulated by random values as per the security of additive secret sharing [16]).

2) Simulator for \(CS_1\) in \(secEst\). \(CS_1\) only has the public number \(S\) of sampled users at the beginning, and later receives \(DPF\) keys \(\{k_{j,1}\}_{j=1,S}\) from the sampled users. Since \(CS_1\) does not receive any other information apart from the \(DPF\) keys, the simulator for \(secEst\) can be trivially constructed by invoking the \(DPF\) simulator. Therefore, from the security of \(DPF\) [17], the simulator for \(secEst\) exists.

3) Simulator for \(CS_1\) in \(secBin\). In the \(secBin\) subroutine (Algorithm 4), the steps that require \(CS_{1,2}\) to interact are in lines 5 and 6. Since each of them is invoked in order as per the processing pipeline and their inputs are secret shares, if the simulator for each of them exists, the simulator exists for \(CS_1\) in \(secBin\). We first analyze the simulated view about line 5. As aforementioned, we provide two approaches (i.e., FSS-based and ASS-based) to perform the secure comparison operation in line 5. For the FSS-based approach, at the beginning of its each execution, \(CS_1\) has a DCF key \(k_1\), a random value \(r_1\), a share \(\{accu\}_1\), and later receives a masked share \(\{accu\}_2 + r_2\) followed by outputting \(Val(k_1, accu, r_2)\). Since these information \(CS_1\) receives is all legitimate in FSS-based DCF, the simulator for \(secBin\) can be trivially constructed by invoking the simulator of DCF. From the security of FSS-based DCF [18], the simulator for \(secBin\) exists. We then analyze the simulator for ASS-based approach. It is noted that the ASS-based approach is built on realization of a PPA in the secret sharing domain, which consists of basic secret-shared \(\oplus\) and \(\odot\), and thus the simulator for it exists. Similarly, line 6 also consists of basic multiplication, and thus the simulator for it exists [52].

4) Simulator for \(CS_1\) in \(secOptQR\). Our optimized secure QR algorithm builds on the technique from [36]. Specifically, when computing \(k\) multiplications of the form \([U^A V_j]^A, j \in [1, k]\), instead...
of using \( k \) independent beaver triples \( \{Z_i\}_i^A = [X_j^A][Y_j^A] \), we can use \( k \) correlated beaver triples \( \{Z_i\}_i^A = [X_j]^A[Y_j]^A \), where a single matrix sharing \( [X]^A \) is used to mask the constant multiplicand \( [U]^A \). Next, we analyze the existence of the simulator for the technique. We first show that the view of \( CS_1 \) on the correlated Beaver triples can be simulated. It is noted that the shares of each element in the correlated beaver triples \( \{Z_i\}_i^A = [X]^A[Y]^A \) is generated based on the standard secret sharing. This means that the distribution of the Beaver triple shares received by \( CS_1 \) is identically distributed with random values in the view of \( CS_1 \). We then show that the view of \( CS_1 \) on the masked constant multiplicand can be simulated. In this phase, \( CS_1 \) first receives \( \langle U \rangle_2 - \langle X \rangle_2 \) and \( \langle Y \rangle_2 - \langle Z \rangle_2 \) from \( CS_2 \), and then outputs \( \{Z_i\}_i^A \). It is noted that \( \langle Y \rangle_2 - \langle Z \rangle_2 \) is a function of the correlated Beaver triple shares, \( U = X, \) and \( Z \). Therefore, these matrices are uniformly random and independent from one another, and their joint distribution in both real view and simulated view is identical [56]. The simulator for secOpt.iQR exists. \( \square \)

### 8 Experiments

#### 8.1 Setup

We implement a prototype system of PrivGED in Python. Our prototype implementation comprises ~2000 lines of code (excluding the code of libraries). We also implement a test module with another 500 lines of code. All experiments are performed on a workstation with 16 Intel 17-10700K cores, 64 GB RAM, 1 TB SSD external storage running Ubuntu 20.04.2 LTS. The network bandwidth and latency are controlled by the local socket protocol.

**Social Graph Datasets.** We use three social graph datasets: Facebook, Twitter, and Google+. Their statistics are summarized in Table 2. For each node in a tested graph dataset, a vector is extracted representing the node’s local view on the whole graph, based on its social connections with other nodes. Recall that for privacy protection, PrivGED aims to conceal the values, positions, and number of non-zero elements in each node’s local view vector, as such information captures a node’s private social interactions. In particular, in the context of these social graph datasets, the value information reflects two users are connected (e.g., they are friends), the position information reflects with which other users a user is connected, and the number information may reflect how many friends/followers a user has (indicating the user’s social skills).

**Protocol Instantiation.** Eigendecomposition usually works on real numbers, while cryptographic computation needs to work with integers. Following priors works on secure computation [16], [52], we use a common fixed-point encoding of real numbers. Specifically, for a private real number \( x \), we consider a fixed-point encoding with \( t \) bits of precision: \( [x \cdot 2^t] \). In our experiments, we use the ring \( \mathbb{Z}_{2^32} \) in the phase of secure degree histogram estimation and secure binning map generation, and the ring \( \mathbb{Z}_{2^{64}} \) with \( t = 32 \) bits of precision in the phase of secure eigendecomposition. The number of iterations of Eq. (2) is set to 25. For DPF and DCF, we set the security parameter \( \lambda \) to 128. The size of the output matrix of Arnoldi method and Lanczos method is set to 15*15, and the top-3 eigenvalues/eigenvectors are used to verify the accuracy of eigendecomposition, because only top-2 eigenvalues/eigenvectors are used in most community detection tasks [9], [10], [19].

**Ground Truth.** We use the standard Python library scipy.sparse.coo_matrix to store the large-scale social graphs in sparse encoding (plaintext and ciphertext), and then use its standard eigendecomposition library scipy.sparse.linalg.eigh and scipy.sparse.linalg.eigsh to calculate the eigenvalues/eigenvectors on symmetric matrix (i.e., Facebook) and non-symmetric matrices (i.e., Twitter and Google+), respectively. Subsequently, we will use the outputs of the standard library as the ground truth.

#### 8.2 Evaluation on Accuracy

**Secure Binning Map Generation.** To obtain an appropriate sampling rate when securely estimating the degree histogram, we compare the accuracy of the binning map with varying sampling rates. Fig. 5 summarizes the experiment results, where we set the number of bins to 10 and use the results of sampling rate = 100% as the ground truth. It is observed that when the sampling rate is set to 10%, we can obtain a satisfactory accuracy (about 85% - 98%).

**Secure Eigendecomposition.** We first implement the plaintext Arnoldi method, Lanczos method, and QR algorithm to compute the plaintext eigenvalues/eigenvectors, and then calculate the RMSE in the top-3 eigenvalues/eigenvectors between the plaintext and the ground truth (output by the standard Python library). We then execute PrivGED to compute the ciphertext eigenvalues/eigenvectors, and calculate the RMSE in the top-3 eigenvalues/eigenvectors between the PrivGED’s result and the ground truth. We summarize the results in Figs. 6 and 7. It is observed that the RMSE of

| Dataset  | Type    | Nodes | Edges  |
|----------|---------|-------|--------|
| Facebook | Undirected | 3,959 | 170,174 |
| Twitter  | Directed | 76,244 | 1,768,149 |
| Google+  | Directed | 102,100 | 13,673,453 |

![Fig. 5. Accuracy of the binning map under different sampling rates.](image-url)

---

2. [http://snap.stanford.edu/data/ego-Facebook.html](http://snap.stanford.edu/data/ego-Facebook.html)
3. [http://snap.stanford.edu/data/ego-Twitter.html](http://snap.stanford.edu/data/ego-Twitter.html)
4. [http://snap.stanford.edu/data/ego-Gplus.html](http://snap.stanford.edu/data/ego-Gplus.html)
PrivGED is slightly higher than that of plaintext (about 0.1% - 5%), but they exhibit consistent behavior.

8.3 Evaluation on Performance

Secure Degree Histogram Estimation. We first report PrivGED’s cost in securely estimating the degree histogram. Fig. 8 illustrates the running time and Fig. 9 shows the key size of DPF involved in the process, where we set the number of bins to 10 and the maximum degree to $N = 20$. It is noted that Algorithm 3 does not require communication between $CS_{1,2}$. Over the three social graphs, the total key size of all sampled users only ranges from 0.1 MB to 7.8 MB.

Secure Binning Map Generation. We then report PrivGED’s performance in securely generating the binning map. Recall that in Section 5.3.2, we provide an FSS-based approach and ASS-based approach to support the secure comparison operation under different network environments. Therefore, we first evaluate the efficiency under different network delays, and report the results in Fig. 10. Additionally, we report the benchmark cost of the two secure comparison approaches under different network delays and bit lengths, in Figs. 11 and 12. It is observed that when the network delay is low (about 0 ms-2 ms), the ASS-based approach faster than the FSS-based approach. As the network delays grows (i.e., >2 ms), the FSS-based approach becomes faster.

Local View Data Encryption. We now report the size of the encrypted social graph to show the storage saving of PrivGED, which is shown in Fig. 13. It is observed that compared to direct encryption of all matrix elements by ASS, our protocol achieves a considerable storage saving (up to 90% under $\epsilon = 1, \delta = 10^{-5},$ and 10 bins). In addition, compared to way of standard LDP-based encryption (i.e., 1 bin), our encryption protocol (with 10 bins) still achieves a considerable storage saving (up to 70%).

Secure Eigendecomposition. We now report PrivGED’s computation and communication performance in secure eigendecomposition. We first examine the performance gain of our optimized secure QR algorithm, and summarize the results in Table 3, where we set the matrix dimension delay is low (about 0 ms-2 ms), the ASS-based approach faster than the FSS-based approach. As the network delays grows (i.e., >2 ms), the FSS-based approach becomes faster.

Secure Binning Map Generation. We then report PrivGED’s performance in securely generating the binning map. Recall that in Section 5.3.2, we provide an FSS-based approach and ASS-based approach to support the secure comparison operation under different network environments. Therefore, we first evaluate the efficiency under different network delays, and report the results in Fig. 10. Additionally, we report the benchmark cost of the two secure comparison approaches under different network delays and bit lengths, in Figs. 11 and 12. It is observed that when the network delay is low (about 0 ms-2 ms), the ASS-based approach faster than the FSS-based approach. As the network delays grows (i.e., >2 ms), the FSS-based approach becomes faster.

Local View Data Encryption. We now report the size of the encrypted social graph to show the storage saving of PrivGED, which is shown in Fig. 13. It is observed that compared to direct encryption of all matrix elements by ASS, our protocol achieves a considerable storage saving (up to 90% under $\epsilon = 1, \delta = 10^{-5},$ and 10 bins). In addition, compared to way of standard LDP-based encryption (i.e., 1 bin), our encryption protocol (with 10 bins) still achieves a considerable storage saving (up to 70%).

Secure Eigendecomposition. We now report PrivGED’s computation and communication performance in secure eigendecomposition. We first examine the performance gain of our optimized secure QR algorithm, and summarize the results in Table 3, where we set the matrix dimension
We show in Fig. 16 the overall running time and online communication at the cloud, for varying number of bins ($\epsilon = 1, \delta = 10^{-6}$). "No bin" refers to the use of standard LDP.

TABLE 3

| M | Time (ms) | Gain | Online Comm. (KB) | Gain |
|---|-----------|------|-------------------|------|
| 15 | Basic | Optim. | Gain | Basic | Optim. | Gain |
| 341 | 339 | 0.6% | 65 | 0.2 | 91% |
| 726 | 698 | 3.9% | 19 | 0.6 | 96% |
| 1172 | 1062 | 9.3% | 65 | 1.5 | 97% |

Fig. 14. Running time and online communication of secure eigendecomposition at the cloud, for varying number of bins ($\epsilon = 1, \delta = 10^{-6}$). "No bin" refers to the use of standard LDP.

Fig. 15. Running time and online communication of secure eigendecomposition at the cloud, for varying privacy budget $\epsilon$ ($\delta = 10^{-6}$, 10 bins).

Fig. 16. Overall running time and online communication at the cloud, for varying number of bins ($\epsilon = 1, \delta = 10^{-6}$). "No bin" refers to the use of standard LDP.

Compared to the way of encryption without sparse representation, PrivGED can save up to 80% overall online communication as well as 40% overall computation time. Meanwhile, compared to the way of standard LDP based encryption, PrivGED can still save up to 65% overall online communication and 30% overall running time.

8.4 Comparison With the State-of-The-Art Prior Work

A fair comparison between PrivGED and the state-of-the-art prior work PrivateGraph [8] does not exist due to its downsides analyzed in Section 2.2. Specifically, PrivateGraph has limited security since it generates the binning map in plaintext domain. Besides, PrivateGraph requires frequent interactions between the cloud and the analyst who undertakes processing workload. As reported in PrivateGraph, to obtain the eigenvalues/eigenvectors of the dataset Google+, the analyst must spend 0.2 hours as well as communicate 10 GB with the cloud. In contrast, PrivGED allows the analyst to directly receive the final eigenvalues/eigenvectors. We also note that under the same privacy budget $\epsilon = 1$, the ciphertext size of PrivateGraph is up to 6.3 TB, but that of PrivGED is only 7 GB.

9 CONCLUSION

We present PrivGED, a new system allows privacy-preserving analytics over decentralized social graphs with eigendecomposition. PrivGED leverages the emerging paradigm of cloud-empowered graph analytics paradigm and enables the cloud to collect individual user’s local view in a privacy-friendly manner and perform eigendecomposition to produce desired encrypted eigenvalues/eigenvectors that can be delivered to an analyst on demand. PrivGED delicately builds on the advancements on lightweight cryptographic techniques (ASS and FSS) and local differential privacy to securely embrace the operations required by eigendecomposition analytics, yielding a customized security design. Extensive experiments demonstrate that PrivGED achieves accuracy comparable to the plaintext domain, with practically affordable performance superior to prior work.

REFERENCES

[1] S. Tabassum, F. S. F. Pereira, S. Fernandes, and J. Gama, “Social network analytics: An overview,” Wiley Interdiscip. Rev. Data Min. Knowl. Discov., vol. 8, no. 5, 2018, Art. no. e1256.
[2] U. Can and B. Alatas, “A new direction in social network analysis: Online social network analysis problems and applications,” Physica A: Statistical Mechanics Applied, vol. 535, 2019, Art. no. 122372.
[3] H. Sun et al., “Analyzing subgraph statistics from extended local views with decentralized differential privacy,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., 2019, pp. 703–717.
[4] Z. Qin, T. Yu, Y. Yang, I. Khalil, X. Xiao, and K. Ren, “Generating synthetic decentralized social graphs with local differential privacy,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., 2017, pp. 425–438.
[5] M. Xue, B. Carminati, and E. Ferrari, “P3D - privacy-preserving path discovery in decentralized online social networks,” in Proc. IEEE 35th Annu. Comput. Softw. Appl. Conf., 2011, pp. 48–57.
[6] L. Zhang, X. Li, K. Liu, T. Jung, and Y. Liu, “Message in a sealed bottle: Privacy preserving friending in mobile social networks,” IEEE Trans. Mobile Comput., vol. 14, no. 9, pp. 1888–1902, Sep. 2015.
[7] X. Ma, J. Ma, H. Li, Q. Jiang, and S. Gao, “ARMOR: A trust-based privacy-preserving framework for decentralized friend recommendation in online social networks,” Future Gener. Comput. Syst., vol. 79, pp. 82–94, 2018.

[8] S. Sharma, J. Powers, and K. Chen, “PrivateGraph: Privacy-preserving spectral analysis of encrypted graphs in the cloud,” IEEE Trans. Knowl. Data Eng., vol. 31, pp. 989–993, 2019.

[9] M. E. Newman, “Spectral methods for community detection and graph partitioning,” Phys. Rev. E, vol. 88, no. 4, 2013, Art. no. 042822.

[10] Y. Wang, Z. Di, and Y. Fan, “Identifying and characterizing nodes important to community structure using the spectrum of the graph,” PLoS One, vol. 6, no. 11, 2011, Art. no. 27418.

[11] Y. Wang, X. Wu, and L. Wu, “Differential privacy preserving spectral graph analysis,” in Proc. Pacific-Asia Conf. Knowl. Discov. Data Mining, 2013, pp. 329–340.

[12] F. Ahmed, A. X. Liu, and R. Jin, “Publishing social network graph eigenspectrum with privacy guarantees,” IEEE Trans. Netw. Sci. Eng., vol. 7, no. 2, pp. 892–906, Second Quarter 2020.

[13] C. Lanzos, “An iteration method for the solution of the eigenvalue problem of linear differential and integral operators,” J. Res. Nat. Bureau Standards, vol. 45, no. 4, pp. 255–282, 1950.

[14] M. Curtiss et al., “Unicorn: A system for searching the social graph,” Proc. VLDB Endowment, vol. 6, no. 11, pp. 1150–1161, 2013.

[15] B. Zhou, J. Pei, and W. Luk, “A brief survey on anonymization techniques for privacy preserving publishing of social network data,” ACM SIGKDD Explorations, vol. 10, no. 2, pp. 12–22, 2008.

[16] P. Mohassel and Y. Zhang, “SecureML: A system for scalable privacy-preserving machine learning,” in Proc. IEEE Symp. Secur. Privacy, 2017, pp. 19–38.

[17] E. Boyle, N. Gilboa, and Y. Ishai, “Function secret sharing: Improvements and extensions,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., 2016, pp. 1292–1303.

[18] E. Boyle et al., “Function secret sharing for mixed-mode and fixed-point secure computation,” in Proc. Annul. Int. Conf. Theory Appl. Cryptos. Techn., 2021, pp. 255–279.

[19] M. E. Newman, “Finding community structure in networks using the eigenvectors of matrices,” Phys. Rev. E, vol. 74, no. 3, 2006, Art. no. 036104.

[20] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub, “Extrapolation methods for accelerating PageRank computations,” in Proc. 12th Int. Conf. World Wide Web, 2003, pp. 261–270.

[21] S. Kamvar, T. Haveliwala, and G. Golub, “Adaptive methods for the computation of PageRank,” Linear Algebra Appl., vol. 386, pp. 51–65, 2004.

[22] T. Charalambous, C. N. Hadjicostis, M. G. Rabbat, and M. Johansson, “Totally asynchronous distributed estimation of eigenvector centrality in digraphs with application to the PageRank problem,” in Proc. IEEE 55th Conf. Decis. Control, 2016, pp. 25–30.

[23] A. Sangers and M. B. van Gijzen, “The eigenvectors corresponding important to community structure using the spectrum of the matrix eigenvalue problem,” Comput. J., vol. 41, no. 1, pp. 17–29, 1951.

[24] W. E. Arnoldi, “The principle of minimized iterations in the solution of the matrix eigenvalue problem,” Quart. Appl. Math., vol. 9, no. 1, pp. 17–29, 1951.

[25] J. G. F. Francis, “The QR transformation - Part 2,” Comput. J., vol. 4, no. 4, pp. 332–345, 1962.

[26] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank citation ranking: Bringing order to the web,” Stanford InfoLab, Tech. Rep., pp. 1–17, 1999.

[27] C. Meng, S. Ramchandran, and Y. Liu, “Cross-node federated graph neural network for spatio-temporal data modeling,” in Proc. ACM SIGKDD Conf. Knowl. Discov. Data Mining, 2021, pp. 1202–1211.

[28] J. Zhou et al., “Vertically federated graph neural network for privacy-preserving node classification,” J. Comput. Appl. Math., vol. 277, pp. 192–201, 2015.

[29] W. E. Arnoldi, “The principle of minimized iterations in the solution of the matrix eigenvalue problem,” Quart. Appl. Math., vol. 9, no. 1, pp. 17–29, 1951.

[30] J. G. F. Francis, “The QR transformation - Part 2,” Comput. J., vol. 4, no. 4, pp. 332–345, 1962.

[31] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank citation ranking: Bringing order to the web,” Stanford InfoLab, Tech. Rep., pp. 1–17, 1999.

[32] C. Meng, S. Ramchandran, and Y. Liu, “Cross-node federated graph neural network for spatio-temporal data modeling,” in Proc. ACM SIGKDD Conf. Knowl. Discov. Data Mining, 2021, pp. 1202–1211.

[33] J. Zhou et al., “Vertically federated graph neural network for privacy-preserving node classification,” J. Comput. Appl. Math., vol. 277, pp. 192–201, 2015.

[34] C. Wu, F. Wu, Y. Cao, Y. Huang, and X. Xie, “FedGNN: Federated graph neural network for privacy-preserving recommendation,” in Proc. Int. Workshop Federated Learn. User Privacy Data Confidentiality, 2021, pp. 1–7.

[35] Z. Xiang, X. Chen, J. K. Liu, and S. Gao, “DeepPAR and deepDPA: Privacy preserving and asynchronous deep learning for industrial IoT,” IEEE Trans. Ind. Informat., vol. 16, no. 3, pp. 2081–2090, Mar. 2020.

[36] D. A. Kuncicky, J. Plattner, and I. Pratt, “Privacy-preserving shortest path computation,” in Proc. Netw. Distrib. Syst. Secur. Symp., 2016, pp. 1–15.

[37] M. Du, S. Wu, Q. Wang, D. Chen, P. Jiang, and A. Mohaisen, “GraphShield: Dynamic large graphs for secure queries with forward privacy,” IEEE Trans. Knowl. Data Eng., vol. 34, no. 7, pp. 3295–3308, Jul. 2022.
[60] M. Curran, X. Liang, H. Gupta, O. Pandey, and S. R. Das, “ProCSA: Protecting privacy in crowdsourced spectrum allocation,” in Proc. Eur. Symp. Res. Comput. Secur., 2019, pp. 556–576.

Songlei Wang is currently working toward the PhD degree with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His research interests include cloud computing security and secure machine learning.

Yifeng Zheng is an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His publications appeared in prestigious journals (e.g., IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Information Forensics and Security, IEEE Transactions on Multimedia, and IEEE Transactions on Services Computing) and conferences (e.g., ESORICS, DSN, PERCOM, and AsiaCCS). His current research interests are focused on security and privacy related to cloud computing, IoT, machine learning, and multimedia.

Xiaohua Jia (Fellow, IEEE) is a chair professor with the Department of Computer Science, City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks and mobile computing.

Xun Yi is a professor with the School of Computing Technologies, RMIT University, Australia. His research interests include applied cryptography, computer and network security, mobile and wireless communication security, and privacy-preserving data mining.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.