Privacy-Preserving Spectral Analysis of Large Graphs in Public Clouds

Sagar Sharma
Data Intensive Analysis and Computing(DIAC) Lab
Kno.e.sis Center
Wright State University, Dayton, OH
sharma.74@wright.edu

James Powers
Data Intensive Analysis and Computing(DIAC) Lab
Kno.e.sis Center
Wright State University, Dayton, OH
powers.4@wright.edu

Keke Chen
Data Intensive Analysis and Computing(DIAC) Lab
Kno.e.sis Center
Wright State University, Dayton, OH
keke.chen@wright.edu

ABSTRACT

Large graph datasets have become invaluable assets for studying problems in business applications and scientific research. These datasets, collected and owned by data owners, may also contain privacy-sensitive information. When using public clouds for elastic processing, data owners have to protect both data ownership and privacy from curious cloud providers. We propose a cloud-centric framework that allows data owners to efficiently collect graph data from the distributed data contributors, and privately store and analyze graph data in the cloud. Data owners can conduct expensive operations in untrusted public clouds with privacy and scalability preserved. The major contributions of this work include two privacy-preserving approximate eigendecomposition algorithms (the secure Lanczos and Nyström methods) for spectral analysis of large graph matrices, and a personalized privacy-preserving data submission method based on differential privacy that allows for the trade-off between data sparsity and privacy. For a N-node graph, the proposed approach allows a data owner to finish the core operations with only $O(N)$ client-side costs in computation, storage, and communication. The expensive $O(N^2)$ operations are performed in the cloud with the proposed privacy-preserving algorithms. We prove that our approach can satisfactorily preserve data privacy against the untrusted cloud providers. We have conducted an extensive experimental study to investigate these algorithms in terms of the intrinsic relationships among costs, privacy, scalability, and result quality.

CCS Concepts

•Security and privacy → Privacy-preserving protocols; Management and querying of encrypted data; •Computing methodologies → *MapReduce algorithms;*

Keywords

ACM proceedings;Cloud computing,Privacy preserving computations, Spectral analysis, Approximate eigen decomposition

ASIA CCS '16, May 30-June 03, 2016, Xi'an, China

ISBN 978-1-4503-4233-9/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2897845.2897857

1. INTRODUCTION

The continuous and rapid expansion of social networks, mobile and web applications, and biomedical research has led to generation of big graph datasets, which have great value in both business and scientific research as shown by recent studies [1]. These graph datasets are often continuously updated, making it expensive to store and analyze. It is thus appealing for data owners to use cloud infrastructures for computation and data storage, as it mitigates the need to establish, operate, and maintain expensive in-house infrastructures. More importantly, the elasticity of cloud computing infrastructures allows users to adapt rapidly to the changes in storage and computing requirements.

However, data privacy becomes a significant concern once data is moved to the cloud. Recent studies show that adversaries can explore infrastructure vulnerabilities to gain unauthorized accesses [31], and curious employees of the cloud provider may snoop private data [9]. Two well-known generic approaches have been developed to preserve privacy while computing with untrusted public clouds: fully homomorphic encryption (FHE) [16] and secure multi-party garbled circuits (GC) [17]. Theoretically, they can be used to construct the privacy-preserving versions of most data mining algorithms. However, these schemes are too expensive to have practical applications. The current best implementation of FHE schemes [8] will result in large ciphertext (e.g., a 4-byte integer will become about 100 kilobytes(KB) - 25,000 times of size increase) and expensive homomorphic multiplication (e.g., about 10 milliseconds (ms) for single multiplication, and it is more expensive with multiple levels of multiplications). The major problem with garbled circuits is the communication cost, as each gate of a circuit will incur a communication cost between two parties. A recent study on the gradient-decent-based matrix factorization algorithm for recommendation systems [28] that used garbled circuits shows the communication cost being around 40 gigabytes (GB) for a small-size 4096×4096 matrix in just one iteration.

Thus, efficient approaches that are particularly designed for a specific application (or a specific set of applications) are highly demanded.

In this paper, we develop a cloud-centric privacy-preserving graph spectral analysis framework that enables data owners to privately store and analyze big graphs with untrusted public clouds. We consider graph spectral analysis as a particular area that can possibly benefit from more efficient privacy-preserving methods for computing with public clouds. Graph spectral analysis is important to many applications such as community detection [27], PageRank [5], and spectral clustering [15].

In our framework, data contributors, who are distributed over the

Internet (e.g., users using mobile apps provided by the data owner), agree to contribute to a graph database in the cloud. Typical examples that fit in this framework include social interactions through instant messengers and social media forming person-person interaction graphs, location updates that form person-location graphs, and user ratings of products that form person-product preference graphs. These graph data are encrypted with a certain additive homomorphic encryption (AHE) scheme, such as Paillier scheme [29] and pairing scheme [7] that have much lower costs than the current FHE schemes. The data owner interacts with the cloud to run our proposed privacy-preserving graph spectral analysis algorithms and get analytic results. The data owner may authorize trusted model consumers (e.g., a data mining team in the data owner's organization) to use the analytic results via a service interface. The graphs might be periodically updated and so do the analytic results.

There are three major challenges to this proposed framework.

- How to develop the cloud-side algorithms that solely relies on the partially homomorphic operations allowed by an AHE scheme?
- How to design *practical* cloud-client algorithms that preserve cloud economics and utilize cloud-side scalability?
- How to define data privacy in cloud-centric storage and computation, and design mechanisms to protect it?

We focus on the core operation of graph spectral analysis: the approximate eigendecomposition of graph matrix, and address the challenges with the following key ideas.

Our approach uses a *pseudo homomorphic multiplication* and random data masking mechanism to enable cloud-side homomorphic matrix computations. Specifically, an AHE scheme already enables $E(\alpha+\beta) = f(E(\alpha), E(\beta))$, where $f()$ is some algorithm that works on the encrypted version of two integer operands: $E(\alpha)$ and $E(\beta)$ without decrypting them. The pseudo homomorphic multiplication for $E(\alpha\beta)$ can be achieved by applying additions β times, e.g., $E(\sum_{i=1}^{\beta} \alpha)$, with one of the operands, say β, unencrypted. With these two fundamental operations, we can derive pseudo homomorphic matrix-vector operations. The key problem with this technique becomes protecting the unencrypted operands in pseudo homomorphic operations, for which we design random data masking algorithms that work with the approximate eigendecomposition algorithms.

For an N-node graph, we consider a practical cloud-client collaborative algorithm which should enable the data owner to conduct expensive core operations of complexity $O(N^2)$ in the cloud with scalable algorithms, while limiting the client-side communication and computation costs to $O(N)$. The presented secure Lanczos algorithm and Nyström algorithm satisfy these requirements. We also reveal the intricate relationships between the quality of results, and the computation and communication costs for the two algorithms, in a typical application of spectral analysis - graph spectral clustering.

Our framework addresses the major privacy threat from curious-but-honest cloud providers, who may try to learn private information, such as an interaction between two nodes, the presence of a specific node in the graph, and the eigen-structure of the graph, by snooping on the graph data or the intermediate results from the computations done in the cloud. It protects data privacy in static storage with the AHE encryption schemes and the differential privacy mechanism, and the privacy of the intermediate computing results with the pseudo homomorphic computation and random data masking algorithms.

We also design an efficient privacy-preserving data submission algorithm to simultaneously preserve the sparsity of graph matrices and allow the contributors to achieve differential privacy [13]. It allows the data contributor to disguise node degrees and edges by injecting randomly selected fake links. The fake links are encrypted so that they are indistinguishable from the real links. The number of fake links are selected via $\epsilon-$differential privacy, with the user-tunable privacy parameter ϵ. Our algorithm completely preserves data authenticity and provides differential privacy while keeping low costs. The preserved sparsity helps reduce the cloud storage cost and the costs of the Nyström algorithm.

In summary, our work has the following unique contributions.

- We develop the secure Lanczos algorithm and the secure Nyström algorithm to allow practical privacy-preserving graph spectral analysis on untrusted public clouds. The pseudo homomorphic operations and random data masking algorithms are designed to protect data privacy in computation.
- We design a privacy-preserving distributed sparse graph data submission algorithm that allows the data contributors to employ personalized differential privacy mechanism, while fully preserving data authenticity and sparsity. The preserved sparsity help reduces the costs of cloud storage and privacy-preserving graph analysis algorithms.
- We have studied the intricate relationships among data privacy, costs, scalability, and result quality, with real graph datasets and using graph spectral clustering as the application.

The remaining part of the paper is organized as follows: Section 2 gives background knowledge about the proposed approach, including a brief description on differential privacy, the AHE methods such as the Paillier encryption system, and graph spectral analysis. Section 3 describes the design of cloud-centric data service framework and the threat model we consider. Section 4-6 describes the core algorithms and their properties. Section 7 presents the results of experimental evaluation. Section 8 shows some related work on privacy-preserving computation with the cloud.

2. PRELIMINARIES

In the following, we briefly describe our notations, and present background knowledge on the applicable encryption schemes, differential privacy, the approxiamte eigendecomposition methods of Lanczos and Nyström, and the basic graph spectral analysis algorithm. For clear presentation, we will use Greek characters to represent scalars; lower case letters for vectors; indexed lower case letters for the vector elements; capital letters for matrices, submatrices, or sets; and indexed capital letters for matrix rows.

A graph is represented as $G(V, E)$, where V is the set of vertices and E is the set of edges. We use N and $|E|$ to represent the set sizes of V and E respectively. In addition, $\mathbb{Z}_q^N$, which is used by the encryption schemes, represents the group of N-dimensional vectors of integers modulo q.

Additive Homomorphic Encryption Schemes. Our approach will use additive homomorphic encryption (AHE) schemes to encrypt graph matrices and enable homomorphic computation in the cloud. There are several AHE schemes such as the Paillier encryption [29], the pairing-based encryption [7], and the Ring-LWE based encryption [8]. The additive homomorphic property is described as follows. If $E()$ denote the encryption function, then for

two integers α and β, the homomorphic addition is notated as:

$$E(\alpha + \beta) = E(\alpha) + E(\beta)$$

For simplicity, we re-use '+' in $E(\alpha) + E(\beta)$ to represent the additive homomorphic operation, which may be implemented as some algorithm $f(E(\alpha), E(\beta))$. We consider that $E(\alpha\beta)$ can be implemented as $\sum_{i=1}^{\beta} E(\alpha)$ if one of the operands, say β is unencrypted - we call it *pseudo homomorphic multiplication*. For n-bit integer β, encoded as $b_{n-1} \dots b_0$, this can be implemented efficiently with the double-and-add algorithm, i.e., recursively computing $E(2\alpha), \dots, E(2^{n-1}\alpha)$ and then $\sum_{i=0}^{n-1} E(2^i\alpha)$, which requires $O(\log n)$ additions. For Paillier encryption, it can be more efficient as

$$E(\alpha\beta) = E(\alpha) \text{ mod\_power } \beta,$$

where *mod_power* means the modulo power operation [21]. For simplicity of presentation, we use $E(\alpha)\beta$ to represent this operation. Based on these notations, we can extend partial homomorphic operations to vector dot-product. For two k-element vectors x and y, we have

$$E(x^T y) = E(x^T)y = x^T E(y), \tag{1}$$

with one of the operand unencrypted.

It can be further extended to matrix-vector multiplication. For a k by k matrix A and a k-element column vector x, we have

$$E(Ax) = E(A)x, \tag{2}$$

with x unencrypted. This property will be used in developing the secure Lanczos algorithm. It can also be extended to matrix-matrix multiplication. Assume R is an unencrypted matrix. We have

$$E(AR) = E(A)R, \tag{3}$$

which will be used in developing the secure Nyström algorithm. For this set of operations, we name them *pseudo homomorphic operations*.

Because one operand in these operations has to be unencrypted, finding a way to maintain the confidentiality of the unencrypted operand will be one key problem in our algorithms.

We will use the Paillier encryption in our implementation due to its efficiency in ciphertext size and homomorphic operations. It is a public-key system, which enables more practical uses for a large number of distributed data contributors. Paillier Encryption has proven strong security guarantees. It is indistinguishable under an adaptive chosen-plaintext attack (IND-CPA), and the scheme is probabilistic, meaning that two equal values are mapped to different ciphertexts uniformly at random. Table 1 summarizes the basic costs for Paillier encryption. In our experiment setup, we use 1024-bit Paillier encryption key, the security of which is equivalent to 80-bit AES encryption.

Differential Privacy. Differential privacy [13] is a standard notion in data privacy which protects any individual record from privacy breach in databases via database querying. For two datasets A_1 and A_2 that differ in exactly one record, $M(A_i)$ be the mechanism that outputs noisy statistics $r \in R$ of the datasets, then ϵ-differential privacy is satisfied if the following condition given in [13] holds:

$$Pr[M(A_1) = r] <= \exp^{\epsilon} Pr[M(A_2) = r] \tag{4}$$

It is popularly applied in interactive databases to preserve data privacy while allowing anybody to query the database. The mechanism M is defined as the additive perturbation of a specific query function, such as the COUNT function, i.e., the query result plus a random noise. The noise in the output is engineered so as to approximately preserve the utility of the query function, yet prevent the distinguishability of any individual records in the database. Laplacian noise is of popular choice, where a noise is drawn from the Laplace distribution Laplace(0, b), the PDF of which is $\frac{1}{2b}\exp(-\frac{|x|}{b})$. The parameter b is determined by the user-specified parameter ϵ and the sensitivity of query function: $\Delta = \max|M(A_1) - M(A_2)|$, i.e., $b = \Delta/\epsilon$. For example, for the COUNT function, the sensitivity Δ is 1 and thus, the parameter b is set to $1/\epsilon$. In general, smaller the ϵ setting (i.e., the stronger privacy preferences) and larger the Δ value, higher the level of noise that is added to achieve the desired level of privacy.

In the non-interactive setting where the dataset is open to public eyes, a synthesized database is published that preserves both the desired data utility and differential privacy [6]. The resilience of non-interactive method depends on whether all possible attack queries are considered when designing the synthesized database. The non-interactive setting fits our case. We will design a fake edge addition method to achieve differential privacy against estimations of node degrees and identification of real links, while completely preserving data authenticity for outsourced graph data in the cloud.

Graph Spectral Analysis. When graphs are represented in matrix forms, e.g., adjacency matrix or Laplacian matrix [15], the spectral properties of the matrices play an important role in many analytics tasks such as PageRank [5], community detection [27], and clustering [15]. The key operation of spectral analysis is eigendecomposition [14], which is generally expensive: the complete eigendecomposition of a $N \times N$ matrix has a time complexity of $O(N^3)$. Therefore, approximate algorithms, such as the power iteration methods (e.g. Lanczos method) [11], and the sampling based Nyström method [15, 22], are often used to reduce the time complexity for large graphs. These algorithms typically return approximate top-k eigenvalues and eigenvectors. The core and most expensive operation in these algorithms are matrix-vector multiplication (for iterative methods) and matrix-matrix multiplication (for Nyström methods). See algorithm 1 and 2 for the fundamental steps of Lanczos and Nyström methods respectively.

Algorithm 1 Lanczos Method for Eigen-approximation

1: $b_0 \leftarrow$ random N-dimensional vector;
2: **for** $i \leftarrow 1$ **to** t **do**
3: $\quad b_i \leftarrow Ab_{i-1}$
4: $\quad$ other vector-based processing
5: **end for**
6: Post-processing with the vectors $\{b_i, i = 0..t\}$ to get top-k approximate eigenvectors

Algorithm 2 Nyström Method for Eigen-approximation

1: $s \leftarrow$ generate random index set such that $\|s\| = m < N$
2: $C_{N\times m} \leftarrow$ get the corresponding m column vectors from A
3: $W_{m\times m} \leftarrow$ submatrix of A with rows and column indices in s
4: decompose $W_{m\times m}$ to get top-k eigenvalues $\Lambda_{k\times k}$ and eigenvectors $U_{m\times k}$
5: compute $C_{N\times m}U_{m\times k}\Lambda_{k\times k}^{-1}$

These algorithms reduce the complexity with some sacrifice in accuracy. Greater number of Lanczos iterations, and greater sampling rate for Nyström account for better accuracy, however, increase the computation cost. In subsequent sections, we present our framework that privately processes user-submitted matrices with untrusted public cloud platforms to conduct eigen-anlaysis. We will develop the secure versions of these approximate algorithms for working with encrypted graph matrices in the cloud.

Table 1: Parameter setting and costs for Paillier encryption. b: bits, B:bytes, ms: milliseconds

Keysize(b)	Cipher-size(B)	Enc(ms)	Add(ms)	Mult(ms)	Dec(ms)
1024	256	3.173	0.005	0.052	2.096
2048	512	16.481	0.014	0.160	15.165

3. FRAMEWORK AND SECURITY OVERVIEW

In this section we describe the primary elements of the proposed framework, the threat model, and security guarantees.

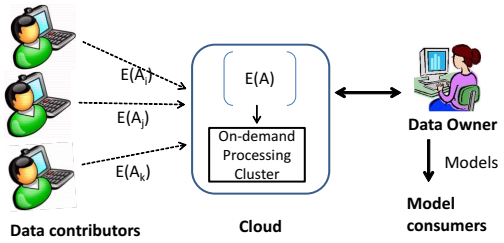

Figure 1: Framework for privacy-preserving graph spectral analysis with the cloud.

Figure 1 shows our framework, which involves four parties: the public cloud, the data owner, data contributors, and authorized model consumer. We also use the client side to represent either the data owner or the data contributors that may involve in cloud-client interactions. The data owner will generate a public key and a private key, and distribute the public key to the data contributors. Data contributors then upload the encrypted data to the cloud to gain the data owner's incentives using a web application or a mobile App provided by the data owner. The graph data can be the data contributor's personal interactions with other contributors in an online social network, or relationships with other items such as locations and products, which make bipartite graphs. Alternatively, the data owner can also upload any in-house graph datasets that are collected via other channels. Data consumers are authorized by the data owner to use the graph analysis service. The data owner will provide analytic results to the model consumers.

The core spectral analysis algorithms work on the encrypted data and guarantee that the most expensive computations (e.g., in $O(N^2)$ complexity) are performed in the cloud, while both the storage and computation costs on the client-side stay low (e.g., in $O(N)$ complexity). Upon updated graphs, the data owner will periodically start the cloud-client spectral analysis algorithms to generate updated models.

3.1 Threat Model

We make practical threat assumptions and focus on confidentiality breaching by honest-but-curious cloud providers - 1) The data contributors operate through secure systems and no information is leaked by themselves. 2) The data owner's in-house infrastructures are secure. An adversary who can compromise data owner's firewall cannot be protected by our framework. 3) All communication channels are secure and data in transit is always encrypted. 4) Our framework is not meant to ensure integrity, freshness, or completeness of results returned to data owners, but to efficiently address the secure storage and processing of large datasets in cloud. 5) The data owner is a trusted party who will manage all they keys and can read all data submitted by data contributors.

Threat 1: Data contributors' privacy.

Social graphs often include sensitive interactions between data contributors. Although data contributors' real IDs have been removed, attackers can still utilize node degrees [37] to de-anonymize and then reveal sensitive links between the de-anonymized data contributors [36]. Our design will protect node degree and links[1] with encryption and a bin-based randomized anonymization scheme that satisfies ϵ-differential privacy.

Threat 2: Privacy of data in computation.

Since the cloud provider is untrusted, not only the data in storage, but also the data in computation needs to be protected. Our framework lets the cloud perform heavy tasks of eigendecomposition over protected data, with no additional information about the matrix or the analytic results revealed to the cloud provider. Any intermediate data generated during computations and interactions are in encrypted form or masked by randomization.

4. DENSE VECTOR DATA SUBMISSION

In this section, we first present our graph data encoding method and basic dense vector submission that breaches no information, but demands high storage costs. Later, in Section 6 we will introduce the personalized privacy-preserving data submission algorithm that allows the contributor to reduce the size of submission with personalized privacy setting.

Representation of Data. In order to use the AHE schemes all values need to be converted to *non-negative integers*. For matrices in the real value domain, a valid method to preserve the desired precision, e.g., d decimal places, is to multiply the original value by 10^d to scale up the values and drop the remaining decimal places. In addition, we also need to shift the values so that the domain is positive. The conversion can be represented as an affine transformation: for a real value α,

$$g(\alpha) = 10^d \alpha + \tau, \tag{5}$$

where τ is a large positive value that converts the smallest negative value to non-negative. This process is clearly reversible for the result of the discussed matrix-vector operations.

Encoding Graph as Dense Matrix. We first present the method that encodes the whole graph as a dense matrix. It will fully preserve the confidentiality of the entire graph, but the storage and computation costs will be much higher than the sparse method we will discuss later. We use the *normalized graph Laplacian matrix* for spectral clustering [32] as an example. Other graph matrices can use a similar method to encode. Below we describe the encoding method for a Laplacian matrix.

For an undirected graph, let D be the diagonal matrix with node degrees on its diagonal - D_{ii} represents the degree of node i, $i = 1..N$. Let W be the adjacency matrix with W_{ij} =1 if and only if the edge (i, j) exists, and $W_{ij} = 0$ otherwise. For undirected graphs, W is a symmetric matrix, where each row(column) of W represents the corresponding node's adjacency edges. The normalized graph Laplacian matrix L is $L = I - D^{-1}W$,

where I is the N by N identity matrix. Specifically, for each row of the Laplacian matrix, say L_i, its element $l_{ij}, j = 1..N$ is

[1]Compared to links, the no-link relationship is less sensitive and thus not protected in our method

$$l_{ij} = \begin{cases} -1/D_{ii} & \text{if } W_{ij} = 1 \text{ and } i \neq j \\ 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Each data contributor may represent one node (or a few nodes) in the graph, and contribute the corresponding row(s) of the Laplacian matrix. To encode the entire vector in the dense format, it is straightforward to apply Eq. 5 to convert each element of the row.

5. PRIVACY-PRESERVING APPROXIMATE EIGENDECOMPOSITION ALGORITHMS

A major motivation for using the cloud to store and process graph data is to move the expensive processing algorithms to the cloud for good scalability and cost-saving. Specifically, we expect the cloud-side handles the $O(N^2)$ computations in a scalable manner, while the client-side costs are limited to $O(N)$. Meanwhile, the algorithm should not reveal any information additional to whatever the adversary already knows, e.g., the information disclosed by the personalized data submission algorithm.

We design our algorithms based on two major methods for approximate eigendecomposition: the Lanczos method [11] and the Nyström method [22]. These two methods have been used to effectively reduce the computational costs for eigendecomposition of large matrices. The challenge is to design the secure cloud-client collaborative algorithms to meet the cost and privacy requirements. In the following, we will briefly describe the original algorithms and then present the secure versions working on encrypted graph matrices.

5.1 Privacy-preserving Lanczos Algorithm

The Lanczos method is based on matrix power iteration[11]. The Lanczos iteration for a matrix $A_{N\times N}$ initiates with the generation of a random N dimensional vector b_0, then the operation $b_i = Ab_{i-1}/\|Ab_{i-1}\|$ is performed in each iteration i. The most expensive computation is the matrix-vector multiplication, and it is also essential to preserve the privacy of the vector b_i because the eigenvectors are derived from the set $\{b_i, i = 0..t\}$ for t iterations.

With an AHE scheme, we use pseudo homomorphic matrix-vector multiplication to compute $E(Ab_{i-1})$, where b_{i-1} has to be in plaintext. The challenge here is to efficiently protect b_{i-1} and recover Ab_{i-1} afterwards.

We describe our random data masking algorithm for AHE schemes. Paillier encryption is used as the representative AHE scheme for its fast computation speed and small ciphertext size.

The core data masking and result recovery algorithm consists of three key steps: preparation, random masking, and recovery.

Preparation. This step consists of two substeps. (1) The data owner selects h N dimensional random vectors, $\{s_i\}$, where h is small which is related to complexity for estimates with high confidence as described in [30], and send them to cloud. These random vectors will be used to protect the vectors b_i in each iteration. The cloud will compute $\mathrm{E}(As_i)$ and send back the results to the data owner. (2) The data owner also generates a random vector b_0 and distribute $\mathrm{E}(b_0)$ to data contributors. Each data contributor i computes $\mathrm{E}(A_i b_0) = \sum_{j=1}^{N} E(A_{ij} b_{0j})$ using pseudo homomorphic multiplication with A_{ij} in plaintext, and sends back the single encrypted scalar $\mathrm{E}(A_i b_0)$ to the data owner.

Random Masking. The perturbed vector $\bar{b_i}$ is given as

$$\bar{b_i} = b_i + r_i \mod q \tag{6}$$

where q is a big random prime large enough to contain all values and computation results in the application domain and r_i is designed from the seed vectors s_i as:

$$r_i = \sum_{l=1}^{h} \alpha_{il} s_l + \sum_{j=0}^{i-1} \beta_{ij} b_j \mod q, \tag{7}$$

where α_{il} and β_{ij} are randomly drawn from Z_q . This approach protects b_i and the security depends on the randomness of r_i. Knowing s_l, $l = 1..h$ to reveal information about b_j vectors is a learning with error (LWE) problem, and hence our perturbation is secure under the intractability of the LWE problem, which will be discussed in more detail later.

Recovery. Recovery of Ab_i is performed at data owner as $A\bar{b}_i$ from the cloud is the result of $Ab_i + Ar_i$. Because

$$Ar_i = \sum_{k=1}^{h} \alpha_{ik} As_k + \sum_{j=0}^{i-1} \beta_{jk} Ab_j \mod q, \tag{8}$$

The set of vectors As_1, As_h are precomputed by cloud. Let $c_j = As_j$. Then computation of $Ar_i = \sum_{j=1}^{h} \alpha_j c_j + \sum_{j=0}^{i-1} \beta_j b_j$ involves only $h + i$ vector-scalar multiplications and $h + i$ vector additions. Ab_i can be recovered with only a $O(N)$ cost: $Ab_i = A\bar{b}_i - Ar_i \mod q$.

Algorithm 3 gives the detail. The client side communication and computation costs are limited to $O(tN)$. The cloud-side pseudo homomorphic computation of $E(A\bar{b}_{i-})$ with the encrypted $E(A)$ and the plaintext $\bar{b}_{i-1}$ can be easily cast to a matrix-row based parallel algorithm. For example, with MapReduce processing [12], each map function handles $E(A_j\bar{b}_{i-1})$ for a row A_j, which generates one element in the result vector; the reduce function simply assembles the elements by their indices.

Algorithm 3 Privacy-preserving Lanczos Algorithm with AHE schemes

1: $b_0 \leftarrow$ random N-dimensional vector and encrypt $E(b_0)$ // data owner
2: download $E(b_0)$, compute $E(A_i b_0)$, and send back to data owner // each data contributor
   for $i \leftarrow 1$ to t do:
3: $\quad \bar{b}_{i-1} \leftarrow$ perturbation based on $\{b_0, .., b_{i-2}\}$ and seed vectors $\{s_1, .., s_h\}$ and upload $\bar{b}_{i-1}$ // data owner
4: $\quad$ compute $E(A\bar{b}_{i-1})$ // cloud
5: $\quad$ download and decrypt $E(A\bar{b}_{i-1})$ // data owner
6: $\quad$ recover b_i from $A\bar{b}_{i-1}$ // data owner
7: $\quad \alpha_{i-1} \leftarrow b_i^T b_{i-1}$ // data owner
8: $\quad w_{i-1} \leftarrow b_i - \alpha_i b_{i-1} - \beta_{i-1} b_{i-2}$ where $b_i = 0$ for $i < 0$ // data owner
9: $\quad \beta_i \leftarrow \|w_{i-1}\|$ // data owner
10: $\quad b_i \leftarrow w_{i-1}/\beta_i$ // data owner
11: end for

Privacy Analysis. Since the adversary only sees the plaintext $\bar{b}_i$ in the cloud-side computation, while the rest part of cloud-side computation does not leak any additional information, the key is how well $\bar{b}_i$ protects b_i. Note that the adversary also knows the seed random vectors $\{s_j\}$, for $j = 1..h$, $s_j \in \mathbb{Z}_q^N$, as they were sent in plaintext in the preparation step. We want to show the following proposition.

PROPOSITION 1. *$\{\bar{b}_i\}$, $i = 0..t$, cannot be distinguished from uniformly random vectors, with the known $\{s_j, j = 1..h\}$.*

PROOF. We will prove the b_0 case. Other cases are similar. Let $a_i = (\alpha_{i1}, \ldots, \alpha_{ik})^T$ be the random parameter vector, and $C = (s_1, ..., s_h)$ be the matrix consisting of s_i as the column vectors. We represent the Equation 6 with matrix operations for $i = 0$:

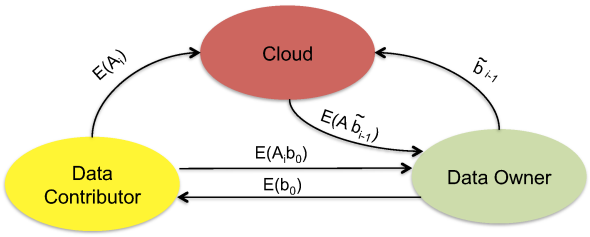

Figure 2: Lanczos for AHE scheme

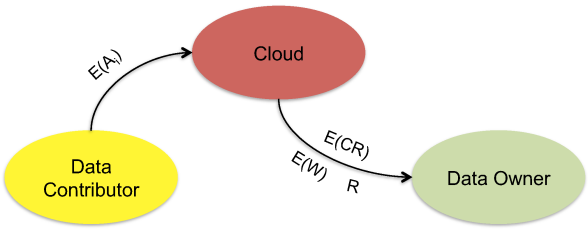

Figure 3: Nyström for AHE scheme

$\bar{b}_0 = b_0 + r_0 = Ca_0 + b_0 \bmod q$, with known C and unknown a_0 and b_0. If C, a_0, and b_0 are drawn uniformly at random, the problem of distinguishing $< C, \bar{b}_0 >$ from uniformly random samples over $\mathbb{Z}_q^{N\times m} \times \mathbb{Z}_q^N$ is exactly the decision version of the *Learning with Errors* (LWE) problem [30]. The existing results on the LWE problem have shown that $\bar{b}_0$ cannot be distinguished from any uniformly random vectors if b_0 is a random vector and a_0 is secret [30]. Therefore, b_0 is securely protected. The same conclusion can be extended to the cases $i \geq 1$ with more unknowns included. □

5.2 Privacy-preserving Nyström Algorithm

The Nyström method has been applied to handle large matrices for spectral analysis [15, 22]. It uses a small sample submatrix of a large matrix to compute approximate eigenvectors. Let $C_{N\times m}$ be m ($m \ll N$) column vectors randomly sampled from the original matrix A. Let S be the set of the sampled column indices and $W_{m\times m}$ be a reduced matrix with the corresponding rows of the matrix C with row indices in S. The W matrix represents the sampled subgraph consisting of the nodes in S. Let the eigendecomposition of W be $W = U\Lambda U^T$, where U, an orthogonal matrix and Λ, a diagonal matrix contain the eigenvectors and eigenvalues of W, correspondingly. Let $U_{m\times k}$ be the top-k eigenvectors and $\Lambda_{k\times k}$ be the corresponding eigenvalues. The Nyström method uses $CU_{m\times k}\Lambda_{k\times k}^{-1}$ to approximate the top-k eigenvectors of the original matrix A.

Let's consider a basic version without encryption and privacy protection. The cloud gets the matrix C and W, and sends W to the client. The client decomposes W and sends back $U_{m\times k}$ and $\Lambda_{k\times k}$. Finally, the cloud computes the result $V = U_{m\times k}\Lambda_{k\times k}^{-1}$ and CV, and sends it back. This algorithm moves the most expensive computations to the cloud. Basically, the cloud-side computation cost is $O(Nmk)$, the client-side computation cost is $O(km^2)$, and the communication cost is $O(kN + m^2)$. The costs are acceptable only when m^2 is $O(N)$. However, this cannot be simply transformed to AHE-encrypted data, as the cloud-side computation of $CU_{m\times k}\Lambda_{k\times k}^{-1}$ will require $U_{m\times k}$ and $\Lambda_{k\times k}$ in plaintext, which reveals the eigen-structure of W and thus significantly breaches privacy.

The key idea of the privacy-preserving version is to replace the step $E(CV)$ in the cloud with $E(CR)$, where R is a $m \times k$ random matrix. The problem is then to recover CV with the result of CR. This can be done by the following operation:

$CR(R^TR)^{-1}R^TV$, if $(R^TR)^{-1}R^TV$ is provided. Since R is a random matrix, it can be the unencrypted operand in pseudo homomorphic multiplication. Furthermore, R can be transferred in plaintext, or encrypted with AES in the same size as the plaintext.

Algorithm 4 shows the detail. Let $\Phi = CR$. The client downloads $E(W)$, R and $E(\Phi)$ in one batch; there is no uploading cost. The cloud side $E(CR)$ is computed with pseudo homomorphic matrix-matrix multiplication and can be done in a scalable manner with a MapReduce program.

Algorithm 4 Secure Nyström Method for AHE schemes

1: preset parameters k, and the sampling number m //data owner
2: $s \leftarrow$ generate random index set of m items from $[1..N]$ //cloud
3: $E(C_{N\times m}) \leftarrow$ get the corresponding m column vectors from $E(A)$ //cloud
4: $E(W_{m\times m}) \leftarrow$ a reduced matrix from C with rows and column indices in s // cloud
5: generate a random matrix $R_{m\times k}$ // cloud
6: compute $E(\Phi) = E(CR)$; // cloud
7: download matrices $E(W_{m\times m})$, $R_{m\times k}$, and $E(\Phi_{N\times k})$ // data owner
8: decrypt $E(W)$ and $E(\Phi)$ // data owner
9: decompose W to get $W = U_{m\times m}\Lambda_{m\times m}U_{m\times m}^T$// data owner
10: get top k eigenvalues $\Lambda_{k\times k}$ and eigen vectors $U_{m\times k}$ of $W_{m\times m}$ //data owner
11: $V_{m\times k} \leftarrow U_{m\times k}\Lambda_{k\times k}^{-1}$ //data owner
12: $M \leftarrow (R^TR)^{-1}R^TU_k\Lambda_k^{-1}$; //data owner
13: compute ΦM; // data owner

It is straightforward to analyze the involved costs in this algorithm. The cloud-to-client transmission consists of three matrices: $E(W)$, $E(\Phi)$, and R, which is $O(kN + m^2)$. The algorithm involves only this download cost. The client-side computation costs consist of $O(kN + m^2)$ decryption operations, and $O(k^2(m + N) + k^3)$ on matrix computation. Since both $m \ll N$ and $k \ll m$, the client side computation cost is highly acceptable. The cloud-side pseudo homomorphic matrix operations can be easily implemented with a MapReduce program with scalability preserved.

Privacy Analysis. Because this algorithm does not ask the client to upload any information, and the cloud side works with either the encrypted data or the random matrix, R, it does not leak any additional information.

5.3 Transmission-Cost Comparison and Trade-off

As we have shown, for both algorithms the client-side computation costs are all linear to the number of nodes N, which can be comfortably handled by the data owner. The key cost differences are on cloud-client data transmission and data decryption.

The secure Lanczos method has both upload and download costs. The total transmission cost is linear to the number of Lanczos iterations, t. The upload stream contains only the plaintext vectors, which incurs much lower costs than the download stream that contains the encrypted vectors. The overall transmission cost is tN encrypted elements.

In comparison, the Nyström method has only the download cost. The key factor for the Nyström method is the sampling rate $r = m/N$, which is intrinsically related to the density of the graph. Let $\Delta = 2|E|/N^2$ be the density of the graph, which is the percentage of the non-empty items in the graph matrix. Let's consider the cost of the transmitted matrices $E(W)$, $E(\Phi)$, and R. The cost of the plaintext matrix, R, is ignorable. The average number of elements in $E(W)$ is around $(rN)^2\Delta$, and $E(\Phi)$ has a fixed cost

kN. Therefore, the total transmission cost of encrypted elements is $(rN)^2\Delta + kN = 2r^2|E| + kN$.

The decryption costs are similar: tN for the lanczos method and $kN + 2r^2|E|$ for the Nyström method.

Since k has to be smaller than t in the Lanczos method, the Nyström method saves the costs if and only if

$$2r^2|E| < (t-k)N. \quad (9)$$

As we will show in experiments, around certain sampling rate the result quality for the Nyström reaches a stable level; similarly there is an approximate range for t to reach a stable level in Lanczos method. The Nyström method can be cheaper than the Lanczos method with lower sampling rates and low graph density, especially when the Lanczos method needs a large number of iterations to converge. However, we also observed that the Nyström method often converges to a lower-quality result. Thus, when selecting the algorithm to use, the user needs to consider the complicated trade-offs between costs and result quality.

6. PRIVACY-PRESERVING SPARSE DATA SUBMISSION

For a big graph with many nodes, each row of the matrix is normally very sparse. An appealing solution is to skip some of the zero entries but still provide enough protection to privacy. We design an algorithm to protect node degrees and links with differential privacy, while also preserving the sparsity of data and the authenticity of data. The problem setting and data encoding method distinguish our method from previous studies [24, 37] on privacy-preserving graph publishing in two aspects. (1) Previous studies aim to share data and models with curious parties. In contrast, we prevent data and models sharing with curious parties. (2) Most existing methods change the authenticity of graph data by adding or removing nodes or edges. Our method will preserve the authenticity of data completely. (3) In our framework, data disguising is done individually by each data contributor who only knows a little bit of global information (i.e., a histogram of node degree distribution generated from sample nodes and distributed by the data owner). Many existing methods work on the entire graph to determine the graph perturbation scheme [37], which is impractical for big data hosting in the cloud.

Regardless how the elements are encrypted, the sparse representation of graphs reveals node degrees and edge existence ? the non-zero entries represent the edge weights from which we can derive node degrees. The known node degrees and edges open doors to several effective privacy attacking methods [4, 37]. Our basic idea to address the issue is by adding randomly selected fake links (encrypted zero entries) to disguise node degrees and real links, which does not change the authenticity of the graph. The goal is to make groups of nodes that have close node degrees indistinguishable under the definition of differential privacy, after adding the fake links. Furthermore, by using a probabilistic encryption scheme such as the Paillier encryption, the encrypted zero entries cannot be distinguished from non-zero ones.

Compared to the dense vector submission method, our sparse method will disclose some additional information. Let d_i be the node degree of node i, and k_i be the number of added zero entries. (1) The actual node degree d_i must be smaller than the number of submitted entries, i.e., $d_i + k_i$. (2) The probability of one submitted entry to be a non-zero entry is increased from $1/N$ to $1/(d_i + k_i)$. If this is acceptable to the data contributor, their cost of submission can be reduced from $O(N)$ to $O(d_i + k_i)$. In the following, we will first discuss the sparsification technique and then describe the selection of k_i to satisfy differential privacy.

6.1 Data Sparsification

First, we use a transformation to preserve the eigen-structure, which makes it possible to use a sparse encoding method. Let γ be a sufficiently large positive integer such that $\lfloor \gamma/D_{ii} \rfloor$, where D_{ii} is the degree of node i, for all $i = 1..N$ are also positive integers with necessary precision preserved. Let $H = \gamma(I - L)$ and let *top-k (or bottom-k) eigenvectors* of H be the eigenvectors corresponding to the largest (or smallest) k eigenvalues, $k = 1..N$. We have the following Proposition.

PROPOSITION 1. *The top-k eigenvectors of H are the same as the bottom-k eigenvectors of L.*

PROOF. Let eigendecomposition of L be $L = U\Lambda U^T$, where U is the eigenvector matrix and Λ is the diagonal eigenvalue matrix: Λ_{ii} is the i-th largest eigenvalue λ_i. Then $\gamma(I - L) = U(\gamma(I - \Lambda))U^T$. Therefore, U is also H's eigenvector matrix. However, H's eigenvalue matrix, $\gamma(I - \Lambda)$ reverses the order of L's eigenvalues.

□

Now, let H_i be the i-th row of H and its element $h_{ij}, j = 1..N$ is

$$h_{ij} = \begin{cases} \gamma - \lfloor \gamma/D_{ii} \rfloor & \text{if } W_{ij} = 1 \text{ and } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

It allows us to develop a concise sparse encoding scheme with H: for data contributor i and every element j, $j = 1..N$, if $W_{ij} = 1$ and $i \neq j$, it outputs $(i, j, E(\gamma - \lfloor \gamma/D_{ii} \rfloor))$; otherwise, with some probability p_i (to be described) it outputs $(i, j, E(0))$ for $i \neq j$.

6.2 Disguising Node Degrees and Edges with Differential Privacy

We propose a bin-based graph disguising algorithm. We will first describe the method to protect node degrees and then discuss the protection on edges. Specifically, we sort the nodes by their node degrees and then partition the distribution by bins. The nodes in the same bin add randomly selected fake edges to achieve node-level differential privacy.

The node degree distribution can be estimated with the node degrees of randomly sampled nodes. This can be achieved by asking some randomly selected data contributors to submit encrypted node degrees before them submitting the graph data. The data owner can build a histogram to approximate the node degree distribution. Apparently, this additional cost is quite low.

Let's generate an equi-height histogram with the sample node degrees, e.g., for a 100-bin histogram, each bin contains about 1% of the nodes. The number of bins is chosen so that each bin contains a moderate number of nodes, for example, a value in (50, 100) to provide satisfactory indistinguishability. Let U_i be the maximum node degree in the i-th bin, and L_i be the minimum degree in the i-th bin. Our purpose is to make the nodes in this bin indistinguishable in terms of node degree, which is implemented via $\epsilon-$differential privacy. Let the query function $F()$ about node degree be quite general, say finding the node degree ranked at k in the bin. Let A and A' be the *neighboring graph* which differ from each other by only one node in the bin. We can derive the sensitivity $\Delta_i = \max\{F(A) - F(A')\} = U_i - L_i$. Note that without bucketization Δ_i can be very large, possibly up to the number of nodes in the graph. Thus, the bucketization mechanism helps reduce function sensitivity and results in much less fake links to better preserve the sparsity.

According to the noise design of differential privacy, we derive that the parameter b of Laplace noise to be $(U_i - L_i)/\epsilon$. This noise can be negative, which suggests we remove some edges and thus destroy the authenticity of data. To avoid this problem, we add an offset to the noise to make it positive. For a specific b, we can always identify the bound q for $Pr(x < q) <= 0.01$. That means, if we add an offset $|q|$ to the distribution, we can make sure the majority of population ($> 99\%$) positive. With such an offset, the number of fake links, $k_{i,j}$ is chosen as follows

$$k_{i,j} = |q_i| + \delta_{i,j},$$

where $|q_i|$ is the offset and $\delta_{i,j}$ is a random integer drawn from Laplace$(0, (U_i - L_i)/\epsilon)$ to make $k_{i,j} > 0$. With such a noise design, the nodes in the same bin satisfy ϵ-differential privacy.

By preserving node-degree differential privacy, edge differential privacy is also satisfied. We define A and A' as a pair of neighboring graphs, if they only differ by one edge. The problem of checking the existence of an edge can be transformed to an edge counting query function. Let's look at any arbitrary edge counting functions. Clearly, the sensitivity of such a function is 1. Thus, Laplace$(0, 1/\epsilon)$ is used to generate the noisy edges. Since the parameter $(U_i - L_i)/\epsilon$ used for disguising node degrees is no less than $1/\epsilon$, the fake links generated for protecting the privacy of node degrees also protect edge privacy.

Algorithm 5 gives the details of our privacy preserving sparse submission algorithm. Here, we only discuss two types of functions for querying node degrees and edges that are already used to design privacy attacks. However, our result can be easily extended to any new types of query functions.

Algorithm 5 Privacy preserving sparse submission (H, ϵ, $d_{i,j}$).

1: input: H: histogram provided by the data owner. ϵ: user selected parameter for ϵ-differential privacy. $d_{i,j}$: the actual node degree.
2: find the bin that contains $d_{i,j}$, whose upper bound and lower bound are U_i and L_i, respectively;
3: $b \leftarrow (U_i - L_i)/\epsilon$;
4: $q \leftarrow b * 3.912$;// for $b = 1$ the q value is at 3.912, which scales with b;
5: draw a value $\delta_{i,j}$ from the distribution Laplace $(0, b)$;
6: $k_{i,j} \leftarrow |q| + \delta_{i,j}$;
7: add the $d_{i,j}$ real links to the list with the sparse encoding;
8: randomly choose $k_{i,j}$ edges from the rest $N - d_{i,j}$ edges and encode them as the encrypted zero entries;
9: submit the $d_{i,j} + k_{i,j}$ items.

7. EXPERIMENTS

To show the practicalness of the proposed approach, we evaluate the computation, storage, and communication costs associated with the data contributors, the cloud, and the data owner. We use spectral clustering as an application of graph spectral analysis to show the result quality and related trade-offs of the two eigendecomposition methods.

Table 2: Statistics of the graph datasets.

Dataset	N	$\|E\|$	AvgDegree	Density
Facebook	3,959	84,243	42	0.0107
Twitter	76,244	1,242,390	32	0.0004
Gplus	102,100	12,113,501	237	0.0023

7.1 Setup

Resources. The client machine is configured with 128 GB of RAM and four quad-core AMD processors. The cloud-side MapReduce program is tested on an in-house Hadoop cluster configured with 14 slave nodes running Apache Hadoop version 1.0.3. Each slave node is configured with 16 GB of RAM, four quad-core AMD processors, 16 map slots, 12 reduce slots, and a 64MB HDFS block size. The MapReduce program is implemented with Java and Java native library and accesses the GMP library (gmplib.org) for fast encryption/decryption. We also implement a customized input/output MapReduce format classes for efficiently handling the encrypted sparse matrix. Local algorithms for proxy side are implemented in the Linux environment with C++ and various libraries such as Armadillo (arma.sourceforge.net)for matrix computation, and the GMP for encryption.

Datasets.

We use three graph datasets from the SNAP database (snap.stanford.edu). Table 2 shows some statistics of these datasets after some pre-processing. The original datasets were used to study some kind of social circles in the three popular social networks [26]. Only the original Facebook data is undirected while the other two are directed. We did some preprocessing to convert the directed graphs to undirected ones and removed the dangling nodes that do not link to any other node. Note that this conversion is only meaningful for testing our algorithms.

We define the graph density as $\frac{2|E|}{N^2}$ for undirected graphs, which is the proportion of non-empty elements in the graph matrix. Density is one of the factors affecting the cost and effectiveness of the Nyström algorithm. We also give the average degree $2|E|/N$ as it is related to the data contributor's cost.

Evaluation Method. Both the Lanczos and Nyström methods involve the trade-off between costs and result quality. Since we focus on graph spectral analysis, we use the results of a typical application, graph spectral clustering, to measure the quality. Although there are several variations of the graph spectral clustering algorithm [32], they all follow the same general steps as depicted in Algorithm 6.

Algorithm 6 Graph spectral clustering algorithm

1: Input: a normalized/unnormalized Laplacian L;
2: Compute the last k eigenvectors $u_1, \ldots u_k$ of L;
3: Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing vectors $u_1, \ldots u_k$ as columns;
4: For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the i-th row of U;
5: Cluster the row vectors $y_i, i = 1, \ldots, n$ with the k-means algorithm into clusters $C_1, \ldots, C_k$.

It is clear that the most expensive step is the eigendecomposition of the Laplacian matrix. The remaining steps can be done locally by the client with $O(N)$ complexity.

We will use the quality of graph spectral clustering as the quality measure, which is computed based on the matching between the baseline cluster labels generated by using precise eigendecomposition, and the labels generated by using the Lanczos or Nyström method. Note that the quality of the result is affected not only by the result of eigendecomposition, but also by the k-means algorithm. In particular, k-means results can be significantly affected by the selection of the initial centroids. To eliminate this uncertainty, we use the kmeans++ algorithm [2] in experiments.

7.2 Data Contributor's Costs

Each data contributor in the framework will generate one row of the graph matrix, encrypt the elements, and transmit them to the cloud. In the dense format, the submitted vector consists of N elements. In the sparse format, the element will be encoded in the sparse format $(i, j, E(.))$, where $E(.)$ is the encrypted non-zero or zero items. The encryption and transmission costs are determined by the encryption method and the number of elements in the row.

Table 3: The perturbation parameters and results.

Dataset	nbins	nodes per bin	original $\|E\|$	|E| after perturbation	% increase
Facebook	100	40	84243	99965	18.66
Twitter	1000	76	1242390	1527286	22.93
GPlus	2000	52	12113501	13228599	9.21

In the sparse format, the total number of elements varies according to the personalized privacy parameter ϵ, as described in Section 4. We select the number of bins so that the number of nodes in each bin is in $[50, 100]$ to provide sufficient indistinguishability within the bin. With $\epsilon = 1.0$, we have the results in Table 3, which shows that the size of increased edges are quite manageable.

Table 4 shows a comparison between dense and sparse schemes using 1024-bit Paillier encryption for the three graphs in terms of the average cost for a data contributor. The actual results of each data contributor varies according to their original node degree. The sparse result is based on the settings in Table 3, which is much lower than the dense format. Even with the dense format, the costs are still acceptable.

Table 4: Contributor's Average Cost

Format	Encrypt A_i (s)			Upload E(A_i) (MB)		
	FB	Twitter	GPlus	FB	Twitter	GPlus
dense	12.60	241.92	324.00	0.97	18.61	24.93
sparse	0.08	0.06	0.41	0.01	0.01	0.03

7.3 Cloud-side Costs

The cloud side storage cost is the sum of all data contributors? contribution. Table 5 shows the storage costs for the dense and sparse formats respectively.

Table 5: The cloud storage and parallel computing costs. (MB: megabytes, GB: gigabytes, TB: terabytes)

Format	Facebook	Twitter	GPlus
dense	3.78GB	1.35TB	2.43TB
sparse	24.41MB	372.87MB	3.15GB

We also experiment with the MapReduce implementation of the cloud-side matrix-vector homomorphic multiplication. The matrix is partitioned by rows in the MapReduce processing. Each Map function handles the homomorphic dot products $E(A_j)b_i$, and the Reduce function sorts the results from the Map output. Clearly, the major computation cost occurs in the Map phase, which is determined by the number of Map waves, which in turn is determined by the amount of input data and the available resources. Since each Map process handles one data block, which is 64MB by default, the maximum number of parallel Map processes is $M/64MB$, where M is the total size (in MB) of the encrypted matrix $E(A)$. With sufficient processing resources, say $M/64MB$ Map slots, all the Map processes can be done in one wave. Figure 4 shows the time costs for matrix-vector multiplication with different dimensions in the in-house cluster for dense encoding of large graphs. With the increase of Map waves, the cost increases slightly non-linearly due to the additional scheduling cost [35]. Overall, the average per-wave cost is about 50-60 seconds. That says, with sufficient parallel processing resources, the MapReduce program can be done in pretty short time. Our in-house cluster has about 224 map slots. With the same block setting and similar worker node configuration, to handle 2.7TB for the dense GPlus matrix, about 2632 similar worker nodes are needed to achieve the lowest cost. In contrast, with the sparse format, our small in-house cluster is sufficient to handle 3.15GB of sparse GPlus data in one wave. The optimal setting of MapReduce system can be achieved with some optimization methods [18], which is out of the scope of our paper.

7.4 Costs for Data Owner

We consider the overall costs for the data owner to run the Lanczos and Nyström methods. These costs are determined by the parameters t, the number of Lanczos iterations and m number of samples in Nyström method. The parameters are intricately linked to the quality of approximation and the specific dataset. Therefore, we use the spectral clustering results to determine the satisfactory settings of t and m for each dataset and then derive the corresponding costs for the data owner.

Spectral Clustering Results. We use the *eigs_sym* function in the Armadillo library, which in turn calls the standard ARPACK library function [23], to find top-k eigenvectors of the unencrypted sparse matrix. Note that it is more numerically stable to find the top-k eigenvectors (corresponding to the largest k eigenvalues) than to find the bottom-k eigenvectors that are required by spectral clustering. Since we have done the transformation on Laplacian matrix $H = \gamma(I - L)$ for encoding (Section 4), the top-k eigenvectors of H are what we need in practice. With these eigenvectors returned by *eigs_sym* we conduct the spectral clustering algorithm to establish the standard labels. For simplicity, we choose $k = 10$ for all datasets[2]. Figure 5 and 6 shows the accuracy change over the number of iterations and the sampling rate for these two methods, respectively. Table 6 gives the appropriate parameter settings for each dataset to achieve the same level of accuracy. Note: Our encryption framework does not affect the quality of results for both Lanczos and Nyström methods.

Table 6: Number of iterations (t) for Lanczos and sampling size (m) for Nystrom to attain satisfactory clustering accuracy

Datasets	N	Accuracy	t	m
Facebook	3959	82%	30	396
Twitter	76244	90%	25	3050
Gplus	102100	92%	30	8168

Costs of the Lanczos method. According to the settings of t, we give the aggregated costs in t iterations for the Lanczos method. The major costs include uploading $\{\bar{b}_i\}$, downloading $\{E(A\bar{b}_i)\}$, and decrypting $\{E(A\bar{b}_i)\}$. Note that all these vectors are dense, regardless of the sparse or dense representation of the matrix $E(A)$. Table 7 shows the aggregated costs for t iterations. Overall, the communication costs are moderate while the decryption is expensive. It still allows the data owner to monitor the change of graph spectral structures in every couple of hours.

Costs of the Nyström method. The major client-side costs of the Nyström method consist of downloading $E(W_{m\times m})$, $E(\Phi_{N\times k})$, and R, and decrypting $E(W)$ and $E(\Phi)$. Other costs are comparably small and skipped in the discussion. Note that these costs are determined not only by m, but also by the sparsity of the matrix, which determines the size of $E(W)$. Table 8 shows the costs for the dense format and the sparse format with the settings

[2]The optimal k should depend on the actual graph clustering structures. However, the simplified setting is enough for our goals.

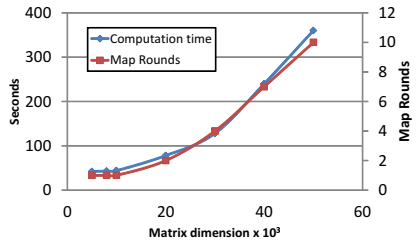

Figure 4: MapReduce processing for matrix-vector homomorphic computation. The cost is approximately determined by the number of Map waves.

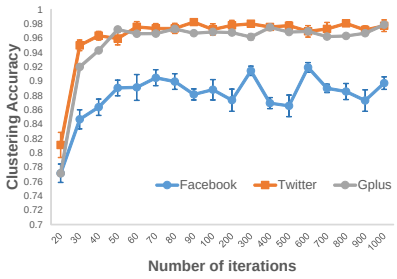

Figure 5: Clustering accuracy vs. the number of iterations for the Lanczos algorithm.

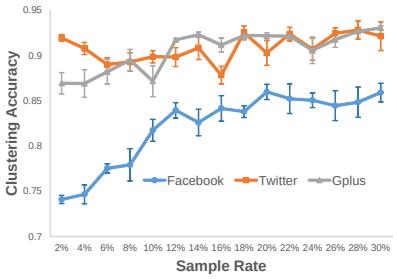

Figure 6: Clustering accuracy vs. the sampling rate for the Nyström algorithm.

Table 7: The aggregated costs of the lanczos method. mins: minutes

Datasets	t	Upload	Download	Decryption
Facebook	30	960KB	30MB	4 mins
Twitter	25	14MB	488MB	66 mins
Gplus	30	23MB	783MB	107 mins

given by Table 3. The result shows that the costs for dense matrices are very high, while the costs for sparse matrices are much lower.

Comparing these two methods, we find that to achieve the same level of model quality the Nyström method with sparse matrices yields lower costs than the Lanczos method. However, as Figure 5 and 6 show, with more Lanczos iterations, the model quality can improve further, while the Nyström method converges to a lower quality level with higher sampling rate. This represents a potential trade-off between the costs and model quality.

8. RELATED WORK

Numerous graph mining algorithms have been developed during the past decade for analyzing the web, social networks, software bugs, and biological networks [1]. With the emergence of big graphs, several scalable algorithms or toolkits have recently been developed dedicated to graph mining, for e.g. PEGASUS [19], Pregel [25], Giraph (giraph.apache.org), and parallel spectral clustering algorithms [10]. However, the data privacy problem for mining graph data in public clouds is not addressed yet.

Fully homomorphic encryption (FHE) schemes have been considered an ideal solution for privacy-preserving computation of outsourced data, but the current most efficient solutions [8] still result in large ciphertext size and expensive homomorphic multiplication. Another generic solution is the garbled circuits (GC) [17] for secure multiparty computation, which suffers from high communication costs. A recent study [28] uses the optimized implementation FastGC [17] to implement iterative privacy-preserving matrix factorization for recommender systems. It shows that one iteration of computation with a 4096×4096 matrix costs about 40GB in communication. Somewhat homomomorphic encryptions provide semantic security to the plaintext being masked, however their application alone is not enough for protecting sparse graph dataset as we describe in our paper.

Atallah et al. [3] present secure outsourcing solutions that are specific to large-scale systems of linear equations and matrix multiplication applications. These solutions fall short as they leak private information, depend on multiple non-colluding servers, and require a large communication overhead. Wang et al. [33] use an iterative approach for solving linear equations via client-cloud collaboration; however, their approach possesses several weaknesses. First, it requires that the entire unencrypted matrix be present at the client side. Secondly, the client side must perform a problem transformation step with a computation cost of $O(N^2)$. These weaknesses render Wang's approach impractical for big matrices.

Privacy-preserving graph data publishing [37] is somewhat related to our work. However, it has a totally different problem setting than ours. Graph data publishing wants to share the graph data but needs to address the privacy attacks from the curious data miners. It normally publishes perturbed graph structures by adding or removing edges and nodes to disguise some graph structures which puts dents to the graph authenticity. In our problem, we avoid sharing of the graph data and we ensure the graph authenticity is fully preserved.

The attacks to anonymized graph data have been extensively discussed [4, 37] and several methods such as k-degree anonymization [24] have been proposed to address the attacks. However, attacks with background knowledge cannot be thoroughly discovered and understood. Thus, differential privacy for graph analysis becomes popular in recent years. Differential privacy has been used in graph spectral analysis in [34] in the interactive setting, however fails to consider the use of cloud infrastructures for analytics. Kasiviswanathan et al. [20] studied the application of differential privacy in graph analysis about node and edge differential privacy in interactive setting. Our method differs from these work in several aspects. We apply the non-interactive setting of differential privacy which perfectly fits the cloud-client computing paradigm. We protect the data authenticity completely, and allow distributed data contributors to submit data individually, with a little information from the data owner about the distribution of overall node degrees.

9. CONCLUSION

In this paper, we present a cloud-centric privacy-preserving graph spectral analysis framework. The graph datasets are collected from distributed data contributors, encrypted with an additive homomorphic encryption scheme, and stored in the cloud. We develop two privacy-preserving eigendecomposition algorithms: the Lanczos and Nyström methods to generate approximate top-k eigenvectors and eigenvalues for conducting the spectral analysis. These two algorithms are designed to meet three requirements: (1) the cloud side processes the expensive operations of $O(N^2)$ complexity with great scalability, (2) the client side costs are linear to the number of nodes N, and (3) these algorithms do not leak data privacy to adversaries. We also design a personalized sparse data submission algorithm for data contributors to preserve data sparsity while still allowing the contributors to achieve personalized privacy with solid guarantees provided via differential privacy. Preserving sparsity significantly reduces the cloud storage costs and very im-

Table 8: Costs of the Nyström method

Dataset	m	Dense		Sparse	
		Download	Decryption	Download	Decryption
Facebook	396	48.0MB	7.0 mins	10.1 +/- 0.5MB	1.5 mins +/- 4.1 secs
Twitter	3050	2.4GB	351.6 mins	187.4 +/- 0.9MB	26.8 mins +/- 1.0 secs
Gplus	8168	16.2GB	2366.2 mins	290.3 +/- 1.6MB	42.5 mins +/-13.4 secs

portantly the costs for the Nyström method.

We have done extensive experiments with real graph datasets to show the storage and computation costs for the data contributors, the cloud, and the data owner. A comparative study has been done between the two privacy-preserving eigendecomposition algorithms to understand the trade-off between costs and result quality. The result shows that the Nyström method with sparse data submission may have lower costs, however with lower quality results, when compared to the Lanczos method. In contrast, the Lanczos method is not affected by data sparsity. Overall, we show that the costs for both proposed methods are practical.

This work can be extended to several directions. First, we may consider new methods to further reduce data transmission cost for both algorithms, especially for the Lanczos method. Second, we will explore into issues governing dynamically updated graphs. Finally, we will also extend the core operations of our framework to other graph mining algorithms.

10. REFERENCES

[1] C. C. Aggarwal and P. S. Yu. *Privacy-Preserving Data Mining: Models and Algorithms*. Springer, 2010.

[2] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, 2007.

[3] M. J. Atallah and K. B. Frikken. Securely outsourcing linear algebra computations. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 48–59, 2010.

[4] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x? anonymized social networks, hidden patterns, and structural steganography. In *International Conference on World Wide Web*, 2007.

[5] P. Berkhin. A survey on pagerank computing. *Internet Mathematics*, 2:73–120, 2005.

[6] A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pages 609–618, New York, NY, USA, 2008. ACM.

[7] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Proceedings of the Second International Conference on Theory of Cryptography*, TCC'05, pages 325–341, Berlin, Heidelberg, 2005. Springer-Verlag.

[8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[9] A. Chen. Gcreep: Google engineer stalked teens, spied on chats. *Gawker, http://gawker.com/5637234/*, 2010.

[10] W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99(PrePrints), 2010.

[11] J. K. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*. Cambridge University Press, 1985.

[12] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[13] C. Dwork. Differential privacy. In *International Colloquium on Automata, Languages andProgramming*. Springer, 2006.

[14] L. Elden. *Matrix Methods in Data Mining and Pattern Recognition*. SIAM, 2007.

[15] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2), 2004.

[16] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Annual ACM Symposium on Theory of Computing*, pages 169–178, New York, NY, USA, 2009. ACM.

[17] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 35–35, Berkeley, CA, USA, 2011. USENIX Association.

[18] D. Jiang, B. C. Ooi, L. Shi, and S. Wu. The performance of mapreduce: An in-depth study. In *Proceedings of Very Large Databases Conference (VLDB)*, 2010.

[19] U. Kang, C. E. Tsourakakis, and C. Faloutsos. Pegasus: Mining peta-scale graphs. *Knowledge and Information Systems (KAIS)*, 2010.

[20] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. *Theory of Cryptography (9783642365935)*, page 457, 2013.

[21] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC, 2007.

[22] S. Kumar, M. Mohri, and A. Talwalkar. Sampling methods for the nyström method. *J. Mach. Learn. Res.*, 13(1):981–1006, 2012.

[23] R. Lehoucq, D. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998.

[24] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 93–106, New York, NY, USA, 2008. ACM.

[25] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[26] J. Mcauley and J. Leskovec. Discovering social circles in ego networks. *ACM Trans. Knowl. Discov. Data*, 8(1):4:1–4:28, Feb. 2014.

[27] M. E. J. Newman. Spectral methods for community detection and graph partitioning. *Phys. Rev. E*, 88:042822, Oct 2013.

[28] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft,

and D. Boneh. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security*, pages 801–812, New York, NY, USA, 2013. ACM.

[29] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238. Springer-Verlag, 1999.

[30] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 84–93, New York, NY, USA, 2005. ACM.

[31] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[32] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[33] C. Wang, K. Ren, J. Wang, and K. M. R. Urs. Harnessing the cloud for securely solving large-scale systems of linear equations. In *Proceedings of ICDCS*, pages 549–558, Washington, DC, USA, 2011.

[34] Y. Wang, X. Wu, and L. Wu. Differential privacy preserving spectral graph analysis. In J. Pei, V. Tseng, L. Cao, H. Motoda, and G. Xu, editors, *Advances in Knowledge Discovery and Data Mining*, volume 7819 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013.

[35] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, 2009.

[36] X. Wu, X. Ying, K. Liu, and L. Chen. A survey of privacy-preservation of graphs and social networks. In C. C. Aggarwal and H. Wang, editors, *Managing and Mining Graph Data*, Advances in Database Systems. Springer US, 2010.

[37] B. Zhou, J. Pei, and W. Luk. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *SIGKDD Explor. Newsl.*, 10(2):12–22, Dec. 2008.