PrivateGraph: Privacy-Preserving Spectral Analysis of Encrypted Graphs in the Cloud

Sagar Sharma, James Powers, and Keke Chen
Data Intensive Analysis and Computing (DIAC) Lab
Ohio Center of Excellence in Knowledge-Enabled Computing
Department of Computer Science and Engineering
Wright State University, Dayton, OH 45435
{sharma.74,powers.4,keke.chen}@wright.edu

Abstract—Big graphs, such as user interactions in social networks and customer rating matrices in collaborative filters, possess great values for both businesses and research. They are not only big but often keep evolving, which requires a large amount of computing resources to maintain. With the wide deployment of public cloud resources, owners of big graphs may want to use cloud resources to obtain storage and computation scalability. However, privacy and ownership of the graphs in the cloud has become a major concern. In this paper, we study privacy-preserving algorithms for one of the important graph analysis techniques - graph spectral analysis for outsourced graph in the cloud. The core operation: eigendecomposition of large matrix is also important to many data mining algorithms. We consider a cloud-centric framework with three collaborative parties: data contributors, data owner, and cloud provider. Graphs are represented as matrices such as adjacency matrix and Laplacian matrix, the elements of which are encrypted and submitted by distributed contributors. The data owner then interacts with the cloud-side programs to conduct spectral analysis, while protecting data privacy from the honest-but-curious cloud provider. For a $N \times N$ graph matrix, we aim to design algorithms with the cloud handling expensive storage and computation in $O(N^2)$ complexity, while data owner and data contributors' algorithms take only $O(N)$. To achieve this goal, we develop the privacy-preserving versions of the two approximate eigendecomposition algorithms: the Lanczos algorithm and the Nyström algorithm, considering two encryption methods: additive homomorphic encryption (AHE) methods and somewhat homomorphic encryption (SHE) methods. Both dense and sparse matrices are studied, while sparse matrices also involve a differentially private data submission protocol to allow the trade-off between data sparsity and privacy. Experimental results show that the Nyströ algorithm with sparse encoding can dramatically reduce data owners' costs; SHE-based methods have lower computational time while AHE-based methods have lower communication/storage costs.

Index Terms—Graph Spectral Analysis, Outsourced Computation, Homomorphic Encryption, Differential Privacy, Cloud Computing

1 Introduction

With the rise of social networks, mobile applications, and sensors, there has been a rapid increase of data generation in areas of commerce, science, and health industries. Among them graph data has become an important resource [1]. Due to the sheer and continuously growing scale, data owners may want to adopt the popularly available cloud resources to achieve both storage and computational scalability. However, there are privacy concerns on collecting and mining graph data with the cloud. (1) These datasets are often sensitive, and thus users are reluctant to share them due to the lack of trust in data owners' ability to protect the data in the cloud. (2) On the other side, data owners also have great interests in preserving the ownership of these proprietary data, as data have become the determining factor in business operation or scientific research. The recent studies [29] and incidents [9] have shown that sensitive data in the cloud are subject to information theft, eavesdropping, and insider attacks. Thus, it is urgent to find means to address both users' and data owners' concerns in cloud-centric data mining.

Protecting data privacy in the cloud is not straightforward, as encryption alone can limit cloud's usage in computation. The traditional method is to store encrypted data (e.g., using Advanced Encryption Standard (AES)) in the cloud; to protect the privacy in computation, data owners need to download, decrypt, and process data locally. However, when the size of data increases to a certain scale, data owners cannot afford doing this anymore for non-linear complexity (e.g., $O(N^2)$ for N-node graphs) algorithms.

Another approach is to develop novel encryption methods to allow the cloud to work with encrypted data. Researchers have been studying the fully homomorphic encryption (FHE) [16] and garbled circuits (GC) [35] based methods. Theoretically, it is possible to construct any data mining algorithms to work with FHE or GC. However, implementing a FHE-based method for complex algorithms will require many levels of denoising/re-encryption to maintain the utility of encrypted data [7], which requires high computational costs and larger ciphertexts. In contrast, the GC-based methods require huge communication costs between data owners and the cloud [26] and thus are impractical for outsourcing large datasets.

A practical design is to let the cloud process the $O(N^2)$ (or higher complexity) operations in parallel, while the data owner contributes constant- or $O(N)$-cost computation to assist the cloud. To our knowledge, there is no general framework that achieves this practical cloud-client work allocation while providing strong security notions (e.g.,

semantic security) for complex data mining algorithms such as graph analysis.

Research Scope and Contributions. In this paper, we study a cloud-centric graph spectral analysis framework that protects privacy from curious cloud providers. Graph spectral analysis has many applications such as network partitioning [25], spectral clustering [15], and web ranking [4]. The basic operation - eigendecomposition of big matrices has even broader applications such as dimensionality reduction [19] and kernel-based learning methods [30].

The cloud-centric framework consists of three major parties: the cloud, the data contributors, and the data owner (or data curator), who collaboratively finish the mining task. Under certain incentives, the data contributors will be willing to contribute their sensitive data to the data owner and trust the data owner to properly protect their data privacy. On the other hand, the data owner wants to use the public cloud resources to manage and mine the ever-increasing user data, but does not trust that cloud providers will properly ensure data privacy. As we consider only the privacy issues, we exclude the case that malicious cloud providers will actively tamper the data or do not conduct the operations asked by the data owner, which are being addressed by the orthogonal line of research on data and computation integrity. Instead, we can assume that cloud providers will follow protocols to honestly conduct the computations they are asked for. Therefore, the assumption "honest-but-curious" cloud providers will be appropriate for our study.

The proposed framework aims to achieve the practical work partitioning between the cloud and the client sides while preserving the privacy of data and analysis result. The cloud will host the big encrypted graph matrix of $N \times N$ dimensions (dense or sparse). In mining data, the cloud will conduct the expensive (e.g., $O(N^2)$) operations with parallel algorithms, the data owner (and data contributors who may also participate) will take tasks of $O(N)$ complexity, which justifies the use of cloud for scalable and economical processing.

We consider two approximate algorithms for spectral analysis - the Lanczos method [10] and the Nyström method [15]. Both can be possibly re-designed and cast to the cloud-centric framework to achieve the practical work partitioning. The core of the Lanczos method: matrix-vector multiplication (an $O(N^2)$ operation), and the core of the Nyström method: matrix-matrix multiplication (an $O(mN^2)$ operation) should be done by the cloud. Both involve only one level of multiplication, i.e., any element of matrix and vector involves in only one multiplication, and multiple levels of additions. We study the application of two types of homomorphic encryption methods: additive homomorphic encryption (AHE) and somewhat homomorphic encryption (SHE), which satisfy this special requirement and are more efficient than the earlier mentioned FHE and GC.

Our research addresses several challenges in applying the SHE and AHE methods under the cloud-centric framework. (1) SHE allows one level of homomorphic multiplication and thus it is straightforward to implement the cloud-side operations. However, their costs are not fully understood yet. (2) AHE methods have smaller ciphertext size, which makes storage and communication efficient, but they maintain only handicapped homomorphic properties - one of the two operands have to be plaintext. Thus, the challenge to apply AHE is twofold: to protect the privacy of the exposed operands, while make sure that data owners' costs stay in $O(N)$. Specifically, our research has four unique contributions.

(1) Two data masking algorithms for protecting the exposed operands in matrix-vector and matrix-matrix multiplications, respectively, for AHE-encrypted data, while also maintaining the client-side cost to stay in $O(N)$. The first data masking algorithm is based on the security guarantee provided by the Learning-with-Error (LWE) problem [28], designed for the matrix-vector multiplication protocol. The second algorithm is based on matrix perturbation that is resilient to ciphter-text only attack, which is sufficient for one-time use of perturbation matrix.

(2) We designed the AHE-based and SHE-based Lanczos algorithms and Nyström algorithms that use the two data masking algorithms, respectively. The Nyström algorithms are specifically designed to benefit from sparse graphs. Both achieve the practical work allocation between the cloud and the client.

(3) We identify the privacy risk of submitting sparse graph matrices and design an efficient local differentially private method for inserting fake edges that have indistinguishable encrypted values $E(0)$ (an important benefit by probabilistic public-key encryption) and do not affect the accuracy of analysis.

(4) A thorough experimental evaluation has been conducted on AHE- and SHE-based Lanczos and Nyströ algorithms. Two most popular encryption schemes are used in the evaluation: the Paillier method [27] for AHE and the Ring-LWE (RLWE) method [7] for SHE. Our results show that the RLWE-based methods have lower computational costs for data owners, while the Paillier-based methods have lower communication costs.

The remaining part of this paper is organized as follows. In Section 2 we establish the background knowledge about spectral analysis of large graph matrices and introduce different AHE and SHE schemes. In Section 3 we give a detailed description of our framework and the AHE- and SHE-based algorithms for privacy-preserving outsourced graph spectral analysis, including the analyses on privacy guarantee and costs. We also present the differentially private solution for sparse graph data submission. Section 4 presents the comprehensive cost evaluation on three social graph datasets. Section 5 gives some related work.

2 Preliminary

This section will setup the notations and give the background knowledge about the eigendecomposition problem for large matrices, the homomorphic encryption schemes we will use, the integer conversion method, and differential privacy that will be used for privacy-preserving sparse matrix submission. For clear presentation, we will use Greek letters for scalars, lower-case letters for vectors, and capital letters for matrices, submatrices or sets.

2.1 Spectral Analysis

The core operation of graph spectral analysis is eigendecomposition of graph matrix, which yields eigenvalues and

corresponding eigenvectors. Eigenvalues and eigenvectors provide valuable information about the structure of matrices and have been used in many data mining algorithms such as social community detection [25], spectral clustering [15], web ranking [4], dimensionality reduction [19], and kernel-based methods [30]. Specifically, for a symmetric matrix A of size $N \times N$, we want to find the decomposition $A = U\Lambda U^T$, where the matrix U consists of the eigenvectors and Λ is a diagonal matrix with eigenvalues on the diagonal.

A complete eigendecomposition of a $N \times N$ matrix possesses a remarkable time complexity of $O(N^3)$. Hence, approximate eigendecomposition algorithms are often used for big N, including the power-iteration Lanczos [10] and matrix sampling based Nyström methods [15]. These algorithms reduce the cost to $O(kN^2), k \ll N$ and return only the top-k eigenvectors/values. The core and most expensive operation in these algorithms are matrix-vector multiplication (for power-iteration methods) and matrix-matrix multiplication (for sampling methods). See Algorithm 1 and 2 for the fundamental steps of Lanczos and Nyström methods respectively. 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, which however, increase the computation cost.

Algorithm 1 Lanczos Method

1: $b_0 \leftarrow$ random N-dimensional vector;
2: **for** $i \leftarrow 1$ to t **do**
3: $\quad b_i \leftarrow Ab_{i-1}$; // the most expensive step
4: $\quad \alpha_i \leftarrow b_i^T b_{i-1}$;
5: $\quad w_i \leftarrow b_i - \alpha_i b_i - \beta_{i-1} b_{i-2}, b_i = 0$ for $i < 0$;
6: $\quad \beta_i \leftarrow \| w_{i-1} \|$;
7: $\quad b_i \leftarrow w_{i-1}/\beta_i$;
8: **end for**
9: α_i and β_i form a tridiagonal matrix $T_{t\times t}$, the top-k eigenvalues and eigenvectors of which are the approximation of A's.

Algorithm 2 Nyström Method

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

2.2 Additive Homomorphic Encryption

Additive homomorphic encryption has the following property. For two integers α and β, the additive homomorphic operation is represented as below.

$$E(\alpha + \beta) = E(\alpha) + E(\beta) \tag{1}$$

As one of the most efficient implementations of AHE schemes, we will use Paillier encryption [27] as the representative to describe our AHE based protocols. The additive homomorphic encryption enables a set of *pseudo homomorphic operations* that make the foundation of our protocols. With one operand, either α or β, unencrypted, we have

$$E(\alpha\beta) = \sum_{i=1}^{\beta} E(\alpha) = \sum_{i=1}^{\alpha} E(\beta). \tag{2}$$

For Paillier encryption, multiplication can be implemented more efficiently as $E(\alpha\beta) = E(\alpha)^\beta \mod n^2$, where n is the public key. We name it *pseudo homomorphic multiplication* as one operand is not encrypted. Based on these two important properties, we can derive pseudo-homomorphic dot-product of vectors, matrix-vector multiplication, and matrix-matrix multiplication, all of which have one operand in plaintext. The key challenge of AHE-based applications is to protect the plaintext operand.

2.3 Somewhat Homomorphic Encryption

Somewhat homomorphic encryption (SHE) schemes have been developed during recent years to achieve one (or a few) levels of homomorphic multiplications in addition to AHE. For example, $(\alpha_1 + \alpha_2)(\alpha_3 + \alpha_4) + (\alpha_5 + \alpha_6)(\alpha_7 + \alpha_8)$ can be computed on encrypted values $E(\alpha_i)$ without decrypting them. Note that any one of the values involves in only one multiplication. In contrast, $\alpha_1\alpha_2\alpha_3$ involves two levels of multiplication. The SHE schemes are often used to compute the degree-2 functions in a homomorphic manner. There are three well-known SHE schemes: the Ring-LWE (RLWE) scheme [7] based on the ring learning-with-error problem [28], the Boneh-Goh-Nissim (BGN) scheme [5], based on pairing of groups and elliptic curves, and the Catalano et al. [8] based on an extension of AHE scheme.

Among them, we will adopt the RLWE scheme in evaluation, as the other two have cost issues. The BGN scheme's decryption depends on computing discrete log, which costs $O(\sqrt{q})$ for plaintext values in $[0, q]$. We find the cost of decrypting a 20-bit value has taken more than 1 second (using the *element_dlog_brute_force* function (crypto.stanford.edu/pbc/)), which can be certainly improved with some optimization [23]. The Catalano et al. [8] scheme was excluded because of its ciphertext expansion. For matrix-vector multiplication on a $N \times N$ encrypted matrix and a N dimensional encrypted vector, the result will include $O(N^2)$ encrypted elements, too expensive to be delivered to the client. Due to the space limitation, we exclude the details of these schemes.

2.4 Integer Conversion

For practical problems, the values are normally in floating-point representation. However, all the discussed encryption schemes work on big integers. Thus, there are additional steps to convert the data to integers for encryption and recover from the result integers to floating-point numbers within acceptable precision losses. Since the homomorphic matrix-vector operations we use include only multiple additions plus one multiplication at maximum, we consider the following simple conversion. For a floating point value x, $x \in \mathbf{R}$, if we would like to preserve the precision of n-digit after the decimal point, we have

$$v = \lfloor 10^n x \rfloor \mod q$$

where q is a large integer so that $10^n x \in (-q/2, q/2)$. The modulo operation maps the values to $[0, q)$, where the negative values are mapped to the range $(q/2, q)$. It is easy to check that x is recoverable: if $v > q/2$, $x \approx (v - q)/10^n$; otherwise, $x \approx v/10^n$. It's also easy to check that, if q is large enough to accommodate the operational results, the results,

including both sign and floating-point values, of addition and multiplication operations on the converted values followed by $\mod q$ are preserved, and thus recoverable. We skip the details. Aliasgari et al. [2] has a thorough discussion on this topic.

2.5 Differential Privacy

Differential privacy [14] is a standard notion in data privacy, which protects individual's privacy from the query-based privacy attacks. For two datasets A_1 and A_2 that differ in exactly one record, let $M(A_i)$ be the mechanism that outputs noisy statistics $r \in R$ of the datasets, then ϵ-differential privacy is satisfied if the following condition holds:

$$Pr[M(A_1) = r] <= \exp^{\epsilon} Pr[M(A_2) = r], \quad (3)$$

where ϵ is the privacy parameter - the smaller it is, the better the preserved privacy. It has been popularly applied to preserve data privacy in querying databases, where any users are allowed to submit limited types of queries and a limited number of repetitive queries subject to the ϵ setting. The mechanism M is defined as the additive perturbation of a specific query function, such as the COUNT function: $M(A)$ = COUNT(A) + random noise. The noise in the output is engineered to approximately preserve the utility of the query function, yet prevent distinguishing any individual records in the database. Laplacian noise is one of the popular choices, where a noise is drawn from the Laplace distribution $Lap(0, b)$, the density function 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: $\Delta = \max|M(A_1) - M(A_2)|$, and $b = \Delta/\epsilon$. For example, the COUNT function has the sensitivity $\Delta = 1$, and thus the parameter b is set to $1/\epsilon$. In general, the smaller the ϵ setting is, the larger the Δ value will be to provide a higher level of protection.

3 Framework and Core Algorithms

First, we will describe the privacy-preserving cloud-centric framework for graph spectral analysis, the threat model, and security expectations. Second, we describe the AHE-based Lanczos algorithm for dense matrices. Third, as many graphs are sparse, we study the privacy issues with sparse representation, and design the privacy-preserving sparse-graph submission protocol for data contributors. Fourth, we develop the AHE-based Nyström method to benefit from the sparse representation. Finally, we will also describe the Lanczos and Nyström algorithms based on the SHE schemes, and analyze the costs associated with all the schemes. The cloud-side parallel processing will be briefly discussed due to the simplicity of the related operations.

3.1 Framework

The involved parties in our framework are: 1) a data owner, denoted as "Owner", who owns the matrix data, 2) data contributors, denoted as "Contributors", who agree on the data owner's privacy declaration and provide private data voluntarily (with or without rewards from the data owner), 3) a public cloud provider, denoted as "Cloud", in a service

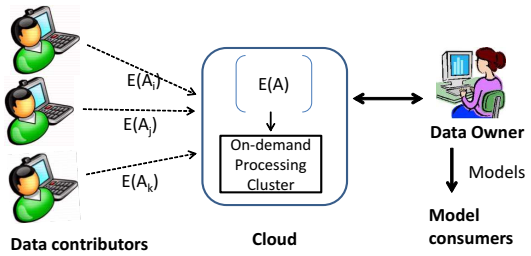


Fig. 1: A framework for cloud-centric privacy-preserving spectral analysis.

level agreement to provide scalable computation and storage, who is honest in providing services, but curious about observed data.

Our aim here is to design practical privacy-preserving eigendecomposition algorithms for graph matrices where the public cloud learns nothing from the stored data, the computations that occur within its infrastructure, and the interactions with other parties. One of the key ideas is to let Owner and Contributors take a small amount of computation and storage responsibility of $O(N)$ complexity, while Cloud takes more expensive $O(N^2)$ parts that can be implemented in parallel and scalable algorithms.

Specifically, Cloud stores the big encrypted matrix $E(A)$ and conducts the expensive homomorphic matrix-vector multiplication. Owner interacts with Cloud and assists in the computation. When collecting data, Owner employs an asymmetric AHE or SHE encryption scheme and publishes a public key for Contributors. Contributors encrypt their submissions and use a web service or a mobile app to upload the encrypted data. Examples of such user data may include interactions between social network users, which are used for detecting social network communities, or user ratings on products for training a recommender system. This cloud-centric framework is particularly important for handling continuously evolving matrix $E(A)$, for which the analytic models should be periodically updated, which are too expensive to be maintained locally by data owners.

3.2 Security Model

We make practical threat assumptions and only focus on the privacy threats from honest-but-curious cloud providers. 1) The data contributors operate through secure systems and no information is leaked to attackers. 2) The data owner's infrastructure is secure. Our framework cannot protect privacy from an insider attack issued by the data owner's organization. 3) All communication channels are secure and data in transit is always protected. 4) Our framework is not meant to ensure the integrity of data that is orthogonal to our work.

Let's model a graph spectral analysis algorithm as a secure protocol $\mathbf{GSA} = (\mathbf{Enc}, \mathbf{Prepare}, \mathbf{Query})$, consisting of three polynomial-time protocols. After the initial stages $\mathbf{Enc}$ and $\mathbf{Prepare}$, the main body is a series of Query-Answering interactions between Owner and Cloud: Owner queries Cloud and Cloud returns the result to Owner. Combined with Owner's local processing, it achieves the algorithmic goal.

$(K, EG) \leftarrow \mathbf{Enc}(1^h, G)$: is a multi-party protocol among three parties: Owner takes a security parameter h and

generates a key-pair $\mathcal{K} = (Pk, Sk)$; Contributors take Pk and output an encrypted graph EG to Cloud.

$(H) \leftarrow \mathbf{Prepare}(m)$: is a multi-party protocol among the three parties: Owner takes some parameter m and works with Cloud (and Contributors) to get a helper data H. It's a one-time setup for securely processing queries later. For some algorithms, this step might be skipped, or some parties may not participate.

$(R) \leftarrow \mathbf{Query}(K, q, EG)$: is a two-party protocol between Owner that holds the key $\mathcal{K}$ and a query q, and Cloud that holds the encrypted graph EG. Cloud processes q and returns the query result R, which can be encrypted vectors or matrices depending on the specific algorithm.

Security Definition. We define security guarantee as follows. (1) For graph encryption, the strongest definition is that given the encrypted graph, no adversary can learn any information about the graph, which is used by the dense-matrix encryption method. (2) We also define a weaker notion for the sparse-matrix encryption method, which does not exactly protect each edge's privacy, but uses differential privacy to protect each data contributor from re-identification [36]. (3) The **Prepare** procedure does not leak any information. (4) With either the dense or the sparse method, the protocol interactions do not leak any additional information. Specifically, given the view of a polynomial number of $Query$ executions for an adaptively generated sequence of queries $q = (q_1, \ldots, q_n)$, no adversary can learn any partial information about either G or q.

We adopt the idea of simulation-based security [12], [23] to formally define the $Query$ protocol security. The semi-honest adversary $\mathcal{A}$ who compromises Cloud observes the interactions between $\mathcal{A}$ (i.e., Cloud) and the challenger $\mathcal{C}$ (i.e., Owner), and tries to infer any useful information. $\mathcal{A}$ knows the encrypted graph EG and the public key, but not the private key of K. $\mathcal{S}$ is a simulator that simulates views of $\mathcal{A}$ in the ideal world corresponding to the views of $\mathcal{A}$ during the protocol execution in the real world. The following formalizes the security definition based on the $\underline{Ideal}$ and $\underline{Real}$ experiments.

$\underline{Ideal_{A,S}(1^h)}$: $\mathcal{A}$ possesses the encrypted graph EG received from Users. If $\mathcal{A}$ were malicious it may also generate a fake graph G, run $\mathbf{Enc}(1^h, G)$ with the public key provided by Owner and generate EG. However, we only consider the semi-honest scenario. $\mathcal{A}$ generates a polynomial number of adaptively chosen queries $(q_1, \ldots, q_m)$ with an intent to compromise the security of the $Ideal$ GSA's $Query$ functionality. We can envision a simulator $\mathcal{S}$ which runs **Prepare** to get the helper data. For each query q_i, $\mathcal{S}$ presents to $\mathcal{A}$ a view as the execution of $Query(K, q_i, EG)$.

$\underline{Real_{A,C}(1^h)}$: $\mathcal{A}$ possesses an encrypted graph EG. $\mathcal{C}$ runs **Prepare** protocol to get the helper data. $\mathcal{A}$ generates a polynomial number of adaptively chosen queries $(q_1, \ldots, q_m)$. For each q_i, $\mathcal{A}$ and $\mathcal{C}$ interactively execute $Query(K, q_i, EG)$.

In both the settings, $\mathcal{A}$ uses the observed views to compute a bit b that is the output by the experiment. We say that the protocol is adaptively semantically secure if for all adversaries with probabilistic algorithms running in polynomial time (i.e., PPT), there exists a PPT simulator S such that

$$|Pr(Real_{A,C}(1^h) = 1) - Pr(Ideal_{A,S}(1^h) = 1)| = negl(h).$$

where $negl(h)$ is a negligible function [21]. In proofs, we only need to show such a simulator exists for each proposed protocol.

3.3 AHE-based Lanczos Construction for Dense Matrix

We first present the AHE-based Lanczos method (Lan-AHE) for dense matrix. The core operation: **Query** implements the privacy-preserving matrix-vector multiplication with client-cost $O(N)$ and cloud-cost $O(N^2)$. Section 2 shows that the most expensive operation in the Lanczos iteration is $b_{i+1} \leftarrow Ab_{i1}$. Thus, the core of the algorithm is that Owner uses **Query** to compute Ab_{i1}, which is combined with some $O(N)$ local processing to implement the Lanczos algorithm. It also uses an IND-CCA secure AHE such as Paillier to encrypt each element of the graph matrix in the dense form. The **Prepare** procedure will generate some helper data for Owner quickly hiding b_i and recovering the result Ab_i, so that the desired security and efficiency goals are achieved. Cloud will take the expensive task of computing Ab_i on the encrypted A. One key challenge is to compute $E(Ab_i)$ with $E(A)$ and plaintext b_i that needs to be protected to achieve the security guarantee - leaking the set $\{b_i\}$ will allow the adversary to approximately reconstruct the matrix.

The basic idea is to submit a masked vector $\bar{b}_i$. The masking technique needs to address two goals: (1) the masked vector does not leak any information to adversaries, i.e., $\bar{b}_i$ cannot be distinguished from any uniformly random noise vector; (2) it is possible to recover Ab_i from the result of $A\bar{b}_i$ efficiently, i.e., no more than $O(N)$ complexity. The design of the noise vector to meet these two goals is the key of this protocol. We describe this protocol in detail as follows.

$\mathbf{Prepare_{Lan-AHE}(h)}$:. This step consists of two sub-steps. (1) The data owner selects h N-dimensional random vectors, $\{s_j, j = 1..h\}$, where h is a constant related to the security of the masking technique (e.g., $h = 80$), and sends them to cloud in plaintext. These random vectors will be used to protect the vector b_i in each iteration. The cloud will compute $E(As_j)$ and send back the results to the data owner, who will decrypt the results to get the vectors $c_j = As_j, j = 1..h$. After the initial setup, the masking results $\{c_j\}$ will be incrementally updated when A is evolving. (2) The data owner also generates a uniformly random vector b_0 and distributes $E(b_0)$ to data contributors. b_0 serves as a secret, which is critical to the security of the whole protocol as shown in Section 3.5. Each data contributor i computes $\mathrm{E}(A_i b_0) = \sum_{j=1}^{N} A_{ij} E(b_{0j}))$, where b_{0j} is the j-th element of b_0 and A_{ij} is the j-th element of the row vector A_i, using pseudo homomorphic multiplication, and sends back the single encrypted scalar $E(A_i b_0)$ to the data owner. It follows that the costs for data owner and data contributors in this step are $O(N)$.

$\mathbf{Query_{Lan-AHE}(K, q, EG)}$: this protocol has two steps: the LWE-based masking to generate the query q (i.e., $\bar{b}_i$), and the efficient recovery method to get Ab_i from the query result. Let q be the perturbed vector $\bar{b}_i$ given as

$$\bar{b}_i = b_i + r_i \mod p \tag{4}$$

where b_i is the vector to be protected, r_i is a noisy vector, and p is a big random prime large enough to contain all values and computation results in the application domain and guarantee the security of perturbed vectors (i.e., the brute-force enumeration is computational intractable). The key of this perturbation is to guarantee r_i cannot be distinguished from any uniformly random vectors and still allow the efficient recovery of Ab_i from the result of $A\bar{b}_i$.

We design r_i as follows to meet the two goals. Its security will be discussed later in Section 3.5 based on the intractability of the Learning-with-Error problem in lattice [28]. $r_i, i \geq 0$, is derived from the seed vectors $\{s_j, j = 1..h\}$ and existing $\{b_j, j = 0..i-1\}$ as:

$$r_i = \sum_{l=1}^{h} \alpha_{il} s_l + \sum_{j=1}^{i-1} \beta_{ij} b_j + b_0 \bmod p, \tag{5}$$

where α_{il} and β_{ij} are randomly drawn from $\mathbb{Z}_p$. This approach protects b_i and its security depends on the randomness of r_i. Note that $\{s_l\}$ is already known by the cloud in the preparation phase, which, however, does not compromise the security of r_i due to the learning-with-error (LWE) based security [28], as long as $b_j, j = 0..i-1$ are secret.

The recovery of Ab_i is performed at data owner as $A\bar{b}_i$ from the cloud is the result of $A\bar{b}_i = Ab_i + Ar_i$ since $\bar{b}_i = b_i + r_i$. Also, because we can compute

$$Ar_i = \sum_{k=1}^{h} \alpha_{ik}(As_k) + \sum_{j=1}^{i-1} \beta_{jk}(Ab_j) + (Ab_0) \bmod p$$

with known $c_k = As_k$ and $Ab_j, j < i$ in complexity $O(N)$, Ab_i can be recovered with a $O(N)$ cost. The correctness of this algorithm is easy to verify.

Algorithm 4 in Appendix gives the detail of the privacy-preserving Lanczos algorithm. The cost and security analysis will be discussed later.

3.4 Construction of Secure Nyström with Differential Privacy and AHE for Sparse Matrix

Many graphs are actually sparse, which has not been fully explored by the Lan-AHE algorithm yet. This sparsity can be utilized to reduce the cloud data storage, cloud-side computation, and the cost of contributors submitting data. For a matrix that has only M non-zero elements on average per row, where $M \ll N$, with sparse representation the submission cost is reduced to $O(M)$ for each contributor, the cloud storage is reduced to $O(MN)$ from $O(N^2)$, and the cost of the core matrix-vector computation is also reduced to $O(MN)$. This saving can be huge, as N is probably around millions while M is only hundreds. However, straightforward sparse encoding may leak private information for graphs. In the following, we will analyze this privacy risk, then present the specific **Enc** procedure for sparse matrix, and finally develop the secure Nyström algorithm to take advantage of the sparse matrix.

3.4.1 Privacy Leak on Sparse Graph Matrices and Our Protection Method

Let's consider a typical graph matrix for spectral analysis: the normalized Laplacian graph 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. The eigenvectors of L can be used for graph spectral clustering [32]. The matrix L is apparently sparse due to the sparsity of W. In traditional sparse encoding, the zero entries are skipped, while the non-zero ones are encoded as (i, j, v) for entry index (i, j).

However, simply encrypting the non-zero entries does not preserve the privacy of the matrix for several reasons. (1) The number of non-zero entries per row is the *node degree* of the corresponding node. (2) The presence of a non-zero entry also implies the existence of the corresponding edge. Both node degree and edge existence information can be used in privacy attacks on social graphs [36].

Our method is to blend in fake edges to disguise both exact node degrees and edge existence. As the encryption methods we use are all probabilistic, each time encrypting a value (or evan the same value encrypted multiple times) will result in a different ciphertext that cannot be distinguished from a uniformly random value. Therefore, the fake edges (i.e., the zero entries in the matrix) cannot be distinguished from other entries as well. Apparently, the added zero entries will not affect the result of matrix-vector operations. The key question is to design a theoretically justified method for users to select the number of fake edges, for which we apply the Laplace mechanism of differential privacy.

The problem setting and data encoding method distinguish our method from previous studies [36] on privacy-preserving graph publishing in several aspects. (1) Previous studies aim to share data and models with curious parties but preserve node and edge privacy. In contrast, we prevent data and models sharing from curious parties. (2) Most existing methods change the authenticity of graph data by adding or removing nodes and edges. Our method will completely preserve the authenticity of data as we add edges only and the edges are encoded with indistinguishable $E(0)$s. (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 have to work on the entire graph to determine the graph perturbation scheme [36], which is impractical for big data hosting in the cloud.

3.4.2 Sparse Encoding for Graph Matrices

We briefly describe the sparse encoding method that preserves the eigen-structure and allows the injections of encrypted zero entries. The following discussion will be specific to sparse Laplacian graph matrices (i.e., the L matrix defined earlier) for spectral clustering; other types of sparse matrices may need different encoding methods.

We use the following transformation that preserves the eigen-structure. Let $H = I - L = D^{-1}W$. Let *the top-k eigenvectors* of H be the eigenvectors corresponding to

the largest k eigenvalues. Clearly, we have the following Proposition.

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

It is easier for both the Lanczos and the Nyström methods to obtain the top-k eigenvectors than the bottom-k ones.

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

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

With integer conversion and sparse encoding, the entries are encoded as $(i, j, E(\lfloor \gamma / Dii \rfloor))$, where γ is a large integer to preserve the desired precision if the edges exist; otherwise, with some probability p_i (to be described) it outputs $(i, j, E(0))$, $i \neq j$.

3.4.3 Bin-Based Differentially Private Graph Perturbation Algorithm

We will first describe the method to protecting node degrees and then discuss why it also protects edge existence. Under the privacy assumption, the adversaries depend on counting the submitted entries to estimate node degrees or the existence of edges. The basic idea is to treat adversaries' estimation on node degrees and edges as queries on the encrypted matrix.

In the standard differential privacy definition, the goal is to disguise any specific person among the entire set of persons that are related to the database. Thus, the key factor, the sensitivity of function, is applied to the whole dataset, which, however, results in very large sensitivity for functions related to node degree on graph datasets. As a result, data contributors have to add many fake items to achieve the desired differential privacy, which seriously impairs sparsity. Specifically, let the query function $F()$ about node degree be quite general, say finding the node degree ranked at k. Let A and A' be the *neighboring graphs* which differ by only one node. Thus, the sensitivity $\Delta = \max\{F(A) - F(A')\}$ is the difference between the largest and the smallest node degree. For a graph of N nodes, this sensitivity can be up to N.

To achieve a better balance between privacy and sparsity, we use a bin-based method to achieve weaker contributor indistinguishability, which is reduced from the whole graph to a subset of nodes in a bin. Specifically, we sort the nodes by their node degrees and then partition the degree distribution by bins. The contributors in the same bin select the number of fake edges with the bin-specific parameter, where the function sensitivity can be much smaller. The node degree distribution can be estimated with the node degrees of randomly sampled nodes. This can be achieved by the data owner asking some randomly selected data contributors to submit encrypted node degrees before them submitting the graph data. The data owner can then build a histogram to approximate the node degree distribution. Apparently, this additional cost is quite low.

Specifically, we 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. Now let A and A' be the neighboring graphs which differ from each other by only one node *in the bin*. We can derive the sensitivity $\Delta_i = \max\{F(A) - F(A')\} = U_i - L_i$, which should be much smaller than N.

According to the noise design of differential privacy, we derive that the parameter b of Laplace distribution $Lap(0, b)$ to be $(U_i - L_i)/\epsilon$. However, this noise can be negative, which asks the contributor to 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, which reduces the overall sparsity but satisfactorily preserves both privacy and authenticity. For a specific b, we can always identify the bound p for $Pr(x < p) <= 0.01$ ($p \approx 3.912$ for $b = 1$ and p linearly scales with b: $p \approx 3.912b$). That means, if we add an offset $|p|$ 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

$$k_{i,j} = |p_i| + \delta_{i,j}, \tag{6}$$

where $|p_i|$ 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 on node-degree based functions.

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 3 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 other types of query functions.

Algorithm 3 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: $p \leftarrow b * 3.912$;// for $p \approx 3.912$ for $b = 1$ the p linearly scales with b: $p \approx 3.912b$;
5: draw a value $\delta_{i,j}$ from the distribution Laplace $(0, b)$;
6: $k_{i,j} \leftarrow |p| + \delta_{i,j}$;
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 items with index (i, j) for $j \geq i$ if it is an undirected graph, otherwise submit all $d_{i,j} + k_{ij}$ items.

3.4.4 Construction of AHE-Based Nyström Method for Sparse Matrix

Note that by using the Lanczos method, data owner does not gain cost reduction from sparse representation, as the $\{\bar{b}_i\}$ and $\{A\bar{b}_i\}$ vectors are always dense. We thus turn to the Nyström method to see whether it can benefit from the sparse representation.

Recall the key steps of the Nyström method in Section 2. Under the cloud-centric framework, Cloud will do the sampling step and the final computation of CV, and Owner will download $E(W)$ and decompose W. Typically, the size of W should be much smaller than the whole matrix but still incurs a significant cost. In practice, m is often set to $0.1N$, thus asymptotically still a parameter related to N. For this reason, the Nyström method does not really fit the goal of $O(N)$ complexity for data owner processing since W has a size $0.01N^2$. However, W might be much smaller with sparse representations. Thus, we can assume that $E(W)$ can be processed with a reasonable cost.

Challenges. Due to the large size C, $N \times m$, it is expected to compute $D_{N\times k} = CV$ in the cloud and return D. Since $k \ll m$, e.g., $k = 10$, this will save the communication and computation cost significantly. An cost-effective solution seems to upload the matrix V in plaintext so that $E(CV)$ can be computed with pseudo homomorphic operations. The challenge is to protect V, as V contains the eigenvectors of W, which can be used to approximately reconstruct the link structure of the corresponding nodes and thus does not meet our security goal.

In the following, we describe the Nyström algorithm that meets our goals. The key idea of our privacy-preserving Nyström algorithm is a matrix masking method and the corresponding protocol.

$\mathbf{Prepare_{Ny-AHE}(m)}$: Owner asks Cloud to randomly select m columns as $E(C)$ and from $E(C)$ selects the m rows for $E(W)$. Owner then decrypts $E(W)$ and eigen-decomposes W to get V.

$(\mathbf{P}, \mathbf{Q}) \leftarrow \mathbf{Query_{Ny-AHE}(K, q, EG_{sparse})}$: The query $q = (\bar{V}, \Delta)$ is generated as follows. First, Owner generates a uniformly random matrix $\Delta_{m\times k} \in \mathbb{Z}_p^{m\times k}$ and an invertible random matrix $R_{k\times k} \in \mathbb{Z}_p^{m\times k}$. Then, V is masked by

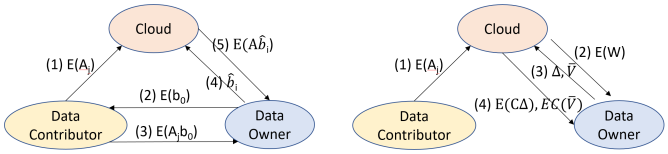
$$\bar{V} = (V + \Delta)R \bmod p, \tag{7}$$

where p is a large non-secret integer, e.g., with 128 bits to preserve both privacy and precision. Both $\bar{V}$ and Δ are submitted to Cloud, who will compute both $P = E(C\bar{V})$ and $Q = E(C\Delta)$ and send them back to Owner. Owner then recovers CV by $CV = C\bar{V}R^{-1} - C\Delta \bmod q$. Algorithm 5 in Appendix shows the detailed steps.

This algorithm has a few important features. (1) The expensive matrix-matrix multiplications $E(C\bar{V})$ and $E(C\Delta)$ of $O(Nmk)$ complexity are conducted in the cloud, which can be easily parallelized with a framework like MapReduce. (2) The computation by data owner involves much smaller matrices: the sparse $m \times m$ W, and dense yet much smaller $m \times k$ V and Δ. (3) The upload cost is small, involving only the plaintext $\bar{V}$ and Δ.

3.5 Security Analysis for AHE-based Constructions

Our security analysis focuses on finding a simulator $\mathcal{S}$ to generate random queries (this is a bit confusing, calling the random vectors queries) that an adversary cannot (computationally) tell from real queries. The proofs will be sketchy. Figure 2 summarizes the interactions in these constructions for easier understanding.



(a) AHE-based Lanczos (Lan-AHE)

(b) AHE-based Nyström (Ny-AHE)

Fig. 2: Interactions among cloud, data owner, and data contributors for the AHE-based algorithms.

Security Analysis for Lan-AHE. In the Lan-AHE algorithm, Clouds view includes the seed vectors $\{s_j\}$ and the perturbed vectors $\{\hat{b}_i\}$. As $\{s_j\}$ is a set of random vectors that do not leak information, we want to show that each $\hat{b}_i$ cannot be distinguished from any random vectors and thus any query sequence q is no different from a randomly generated one. Therefore, the desired simulator $\mathcal{S}$ can just use any random vectors to simulate $\{\hat{b}_i\}$.

Proposition 2. *$\bar{b}_i$, $i = 0..t$, cannot be computationally distinguished from uniformly random vectors by the curious cloud who knows $\{s_j, j = 1..h\}$.*

Proof. We will prove the $\bar{b}_0$ case, and other cases are similar. Let $a_i = (\alpha_{i1}, \ldots, \alpha_{ik})^T$ be the random parameter vector for the round i, and $S = (s_1, ..., s_h)$ be the matrix consisting of $s_j, j = 1..h$, as the column vectors. We represent the Equation 4 with matrix operations for $i = 0$: $\bar{b}_0 = Sa_0 + b_0 \bmod q$, with adversary-known S and $\bar{b}_0$, and unknown a_0 and b_0. If S, a_0, and b_0 are drawn uniformly at random, the problem of distinguishing $< S, \bar{b}_0 >$ from uniformly random samples over $\mathbb{Z}_p^{N\times m} \times \mathbb{Z}_p^N$ is exactly the decision version of the *Learning with Errors* (LWE) problem [28]: it says that $\bar{b}_0$ cannot be computationally distinguished from any uniformly random vectors if b_0 is a random vector and a_0 is secret [28]. Therefore, b_0 is securely protected. The same conclusion can be extended to the cases $i \geq 1$ with more unknowns included. □

The setting of h determines the security level of the protocol. According to Regev [28], finding approximate solutions for the LWE problem costs $O(2^h)$. Thus, we consider $h = 80$ for providing roughly 80-bit security in our experiments.

Security Analysis for Ny-AHE. As Owner's query exposes the masked matrix $\bar{V}$ and the random matrix Δ of the masking $\bar{V} = (V + \Delta)R \bmod p$ to the cloud, we need to show that the masking algorithm effectively preserves the desired security of V. We can safely assume $V \neq 0$ for practical cases. We address this problem from two aspects: (1) since the matrix V contains the clustering structure of W, we show that the masking will surely hide the clustering structure; and (2) we show that it's computationally intractable to distinguish $\bar{V}$ from a randomly generated one of the same size. Therefore, we can conclude our desired simulator $\mathcal{S}$ in the security model can simply produce

randomly generated matrices of same size as $\bar{V}$ as the view of the adversary $\mathcal{A}$.

First, let's understand how the noise addition disguise the clustering structure for the rows of V, in $(V + \Delta)R$. One of the key use of V is the clustering result of the V rows indicating the clusters in the graph of W, which is the basic idea of spectral clustering [32]. Consider each column vector of V as sample values from a random variable. Then, the signal (e.g., the distribution and the clustering structure of V) is covered by the noise if the noise's strength (the mean and variance) is large enough. Typically if the signal-to-noise ratio is $\ll 1$, the signal cannot be recovered. As mentioned in Section 2.4, if we preserve 10 fractional digits for normalized values in $[-1, 1]$, the values in V are represented with about 30 bits. In contrast, the values in Δ are uniformly sampled from $[0, q]$, which has a mean $q/2$ and variance $q^2/12$. q can be selected large enough, e.g., 128 bits, to cover the information in V. In this case, the signal-to-noise ratio based on variances is around the scale $(2^{30}/2^{128})^2 \ll 1$. Thus, the distribution of $V + \Delta$ is dominated by Δ and almost uniformly random, which is not changed by a random transformation $(V + \Delta)R$.

Second, we study the complexity of the attack that distinguishes a uniformly random matrix from a normal $\bar{V}$. The attack is to decide whether there is a valid pair (V, R) that generates the given $\bar{V}$. There are two choices: either enumerating R candidates or V candidates. For each possible R, notated by $\hat{R}$, the estimate of V is $\hat{V} = \bar{V}\hat{R}^{-1} - \Delta$. The attacker then checks the orthogonality of the column vectors in $\hat{V}$ to further screen the candidates. On the other hand, given a valid orthogonal column matrix $\hat{V}$, the test is done as follows. Let $X = \hat{V} + \Delta$. To check whether there is a R to fit $\bar{V} = XR$, one can first apply linear regression to find $\hat{R}$, i.e., $\hat{R} = (X^T X)^{-1} X^T \bar{V}$. If $X\hat{R} == \bar{V}$, then the test passes. The complexity of these attacks is determined by the number of valid $\hat{R}$ and $\hat{V}$. The following proposition shows that this attack is computationally intractable.

Proposition 3. *For values encoded in the h-bit finite field, there are $O(2^{hk})$ candidate R or $O(2^{hm})$ candidate V.*

Proof. According to the theory of general linear group of degree k in a finite field $\mathbb{Z}_p$, where p is h-bit, the number of $k \times k$ invertible matrices is $\Pi_{i=0}^{k-1}(p^k - p^i)$ [11]. It follows there are $O(2^{hk})$ such matrices as the valid candidate $\hat{R}$ to be checked. Similarly, according to the theory of orthogonal matrix group, there are $O(p^m)$ orthogonal matrices in $\mathbb{Z}_p^{m \times m}$ [11]. Thus, for h-bit p, there are $O(2^{nm})$ orthogonal matrices. As V contains k of m orthogonal vectors, there are also $O(2^{nm})$ valid $\hat{V}$. □

Clearly, for a sufficiently large h i.e., $h = 128$, the attacks are computationally intractable.

3.6 SHE-based Constructions

We also consider the SHE-based schemes, as they have been discussed as practical options for outsourced computation [17], [24]. The purpose is to understand how they can be used to construct the solutions and whether they have advantages over the AHE-based solutions.

SHE-Based Lanczos Method. The core operation $E(b_i) = E(Ab_{i-1})$ can be implemented directly when both A and b_{i-1} are encrypted with a SHE scheme with only one-level of multiplication as we have shown in Section 2. Due to the limited one-level multiplication, the data owner needs to help recover the result of $E(b_i) = E(Ab_{i-1})$ and re-encrypt it for the next round. Algorithm 6 in Appendix gives the detail of the SHE-Based Lanczos Algorithm.

Compared to the AHE-based algorithm, this algorithm simplifies the interactions. (1) It does not need data distributors to participate in the computation. (2) It does not have the **Prepare** stage and the query, $E(b_i)$, passed to Cloud is encrypted by SHE. However, decrypting, local processing, encrypting, and uploading $E(b_i)$ are now becoming the major costs for the data owner. The actual costs will be evaluated in experiments.

SHE-Based Nyström Method. The Nyström method involves homomorphic matrix-matrix multiplication CV, which consists of a set of vector dot-products. Similarly, SHE schemes can be applied directly to this operation. The SHE-Based Nyström method is a slight revision of the original Nyström method since V can be encrypted now. The data owner has the responsibility to download and decrypt *sparse* $E(W)$, locally decompose W, encrypt and upload the *dense* V matrix, and finally download and decrypt the *dense* $E(CV)$. We show the details in Algorithm 7.

Algorithm 7 in Appendix gives the steps of the Nyström method for SHE schemes. It differs from the AHE-based algorithm in several aspects. (1) The perturbation and recovery steps are gone due to the encrypted V and thus the client side computation is simplified. (2) The upload cost will increase due to the increased size of encrypted V. (3) The download cost depends on the specific SHE method. The number of download items is reduced but each item's ciphertext size may increase.

Security Analysis. Note that for both SHE-based algorithms, the queries $\{q_i\}$ in the operation **Query** are all encrypted by SHE. For an IND-CPA [21] SHE like RLWE, the simulator can choose random queries to encode and thus the protocols are adaptively semantically secure.

3.7 Cost Analysis

Table 1 compares the asymptotic costs for all the algorithms, where k is the number of top eigenvectors/values, t is the number of Lanczos iterations, and m is the number of samples in the Nyström method. These parameters have the relationships: $k \ll m < N$ and $k < t \ll N$. h is the number of seed vectors in LWE-based masking. The communication costs consider only the encrypted traffics as other traffics are much smaller. The contributors' cost $O(N)$ only occurs in the Lanczos AHE algorithm and not included. Note that the initial matrix setup costs are the same for all the methods and thus not included. The dominating computational costs for data owner are encryption and decryption, compared to other linear-cost operations on plaintext. The acceptable m for the Nyström method can be smaller for a dense matrix that have clearly separated clusters. However, in reality it has to be considerably large to preserve the data utility when we do not know the underlying cluster distribution. As Kumar et al. [22] suggested, m is often set to $0.05N \sim 0.1N$ to get good data utility, which makes the client-side costs non-linear to N for dense matrices.

Thus, Ny-* algorithms are not appropriate for dense matrices. Finally, the costs for the same algorithm (AHE or SHE) implemented with different encryption methods are asymptotically same, and thus we have to look at the real costs to see the effects of different encryption methods.

TABLE 1: Cost distribution between cloud and data owner

Algorithm	Cloud	Data Owner	Comm. cost
Lan-AHE	$O(tN^2)$	$O((t+h)N)$	$O((t+h)N)$
Lan-SHE	$O(tN^2)$	$O(2tN)$	$O(2tN)$
Ny-AHE	$O(Nkm))$	$O(m^2+mk^2+Nk)$	$O(2Nk+2km+m^2)$
Ny-SHE	$O(Nkm))$	$O(m^2+mk^2+Nk)$	$O(Nk+km+m^2)$

3.8 Cloud-Side Parallel Computation

Data encrypted with the mentioned encryption schemes are significantly larger than the unencrypted values. For example, with a 1024-bit key for Paillier encryption, a 64-bit double-type value becomes a 2048-bit ciphertext, a 32-time increase. For RLWE, it's even larger. The encrypted matrices literally turn a common-size problem to a "big data" problem, which requires us to exploit the parallel processing power with the cloud.

In the following, we show the parallel processing algorithm for homomorphic matrix-vector multiplication with AHE-encrypted data. It is straightforward to extend the algorithm for data encrypted with SHE schemes and for matrix-matrix multiplication. For clear presentation, we describe the algorithm with the MapReduce programming model [13]. The MapReduce program consists of the Map and Reduce functions. The Map function takes the masked vector sent by the data owner as the parameter. It applies the vector dot-products to the encrypted rows and emits the results indexed by the row number. The Map outputs are partitioned and sent to Reducers which automatically sort the items by their row numbers and write the result to disk. Readers can check details of MapReduce [13] for a better understanding of the algorithm.

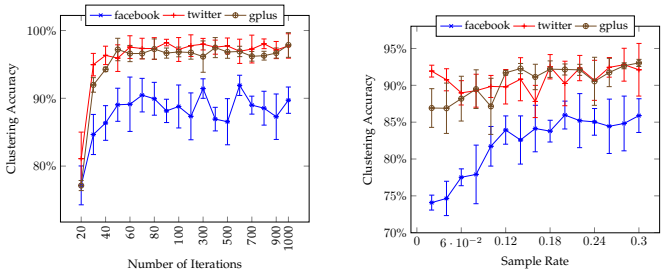
4 Experiments

We have shown that all the developed algorithms provide privacy guarantee under the assumption of the framework. The experiments will evaluate various costs associated with these methods to find out whether any of these algorithms are more efficient. Specifically, our evaluation has three aspects: (i) comparing the basic setup costs for the cloud and data contributors with different encryption methods; (ii) comparing the costs occurring in executing the AHE and SHE based privacy-preserving Lanczos algorithms for the data owner; (iii) the cost-benefit of sparse submission, and the comparison between the AHE and SHE based privacy-preserving Nyström algorithms.

4.1 Setup

Resources. Our setup simulates the framework we described in Section 3. The data owner's system has 128 GB of RAM and four quad-core AMD processors. The cloud infrastructure consists of an in-house Hadoop cluster with a 16-node setup (1 master node and 15 work nodes: each has two quad-core 2.6GHz AMD CPUs and 16GB memory).

Datasets. Three graph datasets in the SNAP database (snap.stanford.edu) are used in our evaluation. They were originally used to study social circles in the three popular social networks - Facebook, Twitter, and GPlus. We make the edges undirected for easier processing in the evaluation.



(a) Number of iterations vs. clustering accuracy for Lanczos

(b) Sample rate vs. clustering accuracy for Nystrom

Fig. 3: Clustering accuracy and parameter settings for Lanczos and Nyström Algorithms

Evaluation Methods. The costs of proposed algorithms are inherently linked to the following parameters: the number of iterations, t, for the Lanczos method, and the sampling size, m, for the Nyström method, respectively, which both in turn are related the quality of results. To untangle this intricate relationship, we use spectral clustering [15] as the application of eigendecomposition to determine the appropriate setting of these parameters. Specifically, we will fix the quality criterion to derive the corresponding parameter setting, and then evaluate the cost of each algorithm under this setting. First, we set the number of clusters to $k = 10$ and derive the ideal clustering results by running the spectral clustering algorithm on plaintext data with the *exact eigendecomposition* algorithm using functions from the Armadillo C++ library (arma.sourceforge.net). Then, the Lanczos method and the Nyström method use different settings of t and m, respectively, to get approximate clustering results. The clustering accuracy is computed by matching the approximate result to the ideal result. The settings are selected as the clustering accuracy becomes stable.

Figures 3a and 3b show how the parameter settings affect the accuracy of the approximate spectral clustering algorithm. Table 2 shows the minimum parameter settings, with which the clustering accuracy becomes stable. They will be used in the cost evaluation.

TABLE 2: The number of iterations (t) for Lanczos and sampling size (m) for Nystrom to reach stable clustering accuracy.

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

4.2 Implementation

We implemented the Paillier encryption for the AHE-based algorithms. The core algorithms are implemented with C++ using GMP big integer library and Armadillo linear algebra library. We also implement the cloud-side MapReduce program with Java and Java native library that accesses the C++ encryption libraries. We use the 80-bit security level to setup

the encryption parameters, and preserve 10 fractional-digit precision for floating-integer conversion (Section 2.4). The results generated by the AHE-based algorithms are verified with those from the normal algorithms on plaintext data. A demo system can be downloaded[1].

We use the HELib library (github.com/shaih/HElib) for the RLWE scheme. 32-bit plaintext encoding is used, which is also the maximum number of bits allowed by HELib. HELib uses the ciphertext packing technique [31], which can be used for encoding dense matrices efficiently. For 80-bit security and 32-bit plaintext, one ciphertext can encode a vector of 630 encrypted values and thus greatly improve the efficiency. However, sparse matrix cannot use ciphertext packing and thus the cost for encoding each value will be 630 times larger. We have also tested the PBC library (crypto.stanford.edu/pbc/) for the pairing-based scheme. However, because its decryption involves solving the expensive discrete logarithm problem, e.g., a 20-bit value encrypted will take about 1 second to decrypt, the aggregated high cost for big matrices will become impractical for the data owner. Thus, the pairing scheme is not included in evaluation.

Table 3 summarizes the costs of basic operations. The Ciphertext-size (C-size), Enc., and Dec. columns represent the costs for encoding, encrypting, and decrypting one value, respectively. RLWE-P represents RLWE using ciphertext packing and the numbers are the average per-element costs based on 630 elements that are encoded in one ciphertext. HELib uses the text format to store the ciphertext. We also zip the ciphertext to minimize the costs. Since the size may vary slightly due to the text-based encoding, the RLWE costs are based on the average of 10 runs. The cost of homomorphic dot-product (dot-p) is based on vectors of 630 elements for an easier comparison crossing different encryption schemes. The dot-p cost can be roughly scaled up for estimating matrix-vector multiplication and matrix-matrix multiplication in different sizes. Note that the RLWE-P costs for encryption and decryption are really low, while the ciphertext is much larger than Paillier.

TABLE 3: Basic costs for different encryption methods in 80-bit security. C-size: Ciphertext size, B:bytes, and ms: milliseconds.

Method	C-size	Enc(ms)	Dec(ms)	dot-p(ms)
Paillier	256B	1.7	1.6	36.0
RLWE	489.3KB	25.3	120.0	5.5E5
RLWE-P	795.3B	0.04	0.2	875.0

4.3 Results on Dense Matrices

The results are organized in three parts: (1) the basic initialization costs including the cloud storage and the data contributors' costs on encoding and submitting the vectors, and (2) the related costs for the cloud and the data owner running the AHE- and SHE-based Lanczos methods. Since the Nyström methods are mainly designed for sparse data, they will be discussed later.

4.3.1 Setup Costs

The setup costs include the contributors' cost and the cloud storage cost. In our framework, we assume each distributed data contributor submits a row (or a few rows) of the matrix $E(A)$. Examples may include a social network user who submits their interactions with others; or a customer that submits ratings/preferences on products. They will download the public key from the data owner, encrypt their share of rows with the selected encryption scheme, and transmit to the cloud. Two costs involved here are the encryption cost and the transmission cost that is represented by the amount of encrypted data. As shown in Table 4, data contributor's encryption costs are the lowest for the RLWE-P method, thanks to the packing technique. However, RLWE-P's communication cost is several times higher than Paillier's.

TABLE 4: Contributor's costs for dense submission

Scheme	Encrypt A_i (seconds)			Upload E(A_i) (MB)		
	FB	Twitter	GPlus	FB	Twitter	GPlus
Paillier	6.7	129.6	173.6	1.0	18.6	24.9
RLWE-P	0.2	3.1	4.1	3.4	58.3	77.9

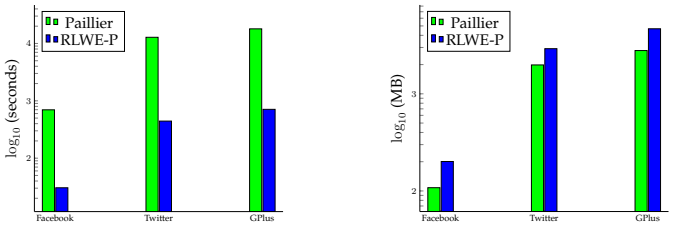
Table 5 lists the basic cloud-side storage costs for the datasets with different encryption methods. Clearly, the dense form of matrix is really expensive. For data of this scale, only cloud infrastructures can handle the storage.

TABLE 5: Cloud Storage Costs for Dense Submissions

Dataset	Facebook	Twitter	Gplus
Paillier	3.7GB	1.4TB	2.5TB
RLWE-P	12.9GB	4.3TB	7.8TB

4.3.2 Privacy Preserving Lanczos Algorithms

We compare the costs of SHE- and AHE-based Lanczos algorithms to see which one has the cost advantage. For the AHE-based algorithm, there is an additional setup cost for data contributors to compute and submit $E(A_i b_0)$, which is the same as the cost of initial data submission as shown in Table 4 and thus skipped in the report. In the following, we show the accumulated costs of all rounds to have a clearer comparison.



(a) Accumulated encryption and decryption costs

(b) Accumulated communication costs.

Fig. 4: Data owner's costs for privacy-preserving Lanczos methods.

Data owner's costs. Figure 4(a) shows the accumulated encryption (only RLWE has) and decryption costs for all iterations of the Lanczos method. For the AHE-based method, this also includes the setup cost for the masking matrix ($h = 80$). Figure 4(b) shows the total communication costs. The Paillier-based method takes much less communication costs, but its computational time is significantly higher.

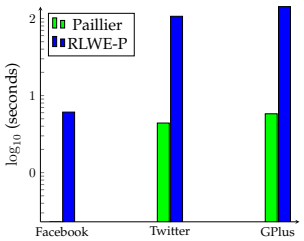
Table 6 shows detailed data owner's costs for the most expensive Gplus dataset. Note that in the AHE-based algorithm, the data owner has no encryption cost. It is clear that the high decryption time is the major shortcoming of

1. sites.google.com/site/privategraphdemo/

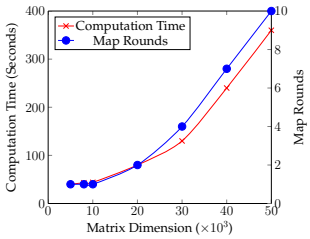
Lan-AHE, among which about 80% is used for setting up the masking matrix. This can be partially addressed by multi-core computers. Overall, the RLWE-P based method spends about two times of the Paillier-based method, while the computational time is much lower. **Cloud-side computation.** The cloud-side computation can be easily parallelized. The computation of $\mathrm{E}(Ab_i)$ can be decomposed to dot-products between a matrix row and b_i, which can be directly mapped to a MapReduce program. With sufficient resources, the computation cost is proportional to the cost of per dot-product. Figure 5 (a) shows the cost of per dot-product for the datasets. Pseudo homomorphic multiplication with Paillier has much lower costs than RLWE-P's. Figure 5 (b) shows the nice scalability of the MapReduce implementation for the Paillier-based matrix-vector multiplication, where most work is done in the Map phase and thus the overall cost is proportional to the number of Map rounds, which implies excellent scalability.

TABLE 6: Data owner's accumulated costs on the Gplus dataset. h: hours; GB: Gigabytes

Schemes	Enc.(h)	Dec.(h)	Upload.(GB)	Dwnld.(GB)
Paillier(Lan-AHE)	-	5.0	0.05	2.7
RLWE-P(Lan-SHE)	0.04	0.16	2.3	2.3



(a) Cost of vector dot-product.



(b) MapReduce scalability.

Fig. 5: Cloud-side processing.

4.4 Sparse Submission and Nyström Algorithms

In this section, we focus on the cost savings of the differentially private sparse matrices and the Nyström algorithms working on the sparse matrices. 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 total number of submitted elements depends on the personalized privacy parameter ϵ, as described in Section 3.4. 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 7. The numbers in the column "$|E|$ pert." are the average of 10 runs. Apparently, the size of increased edges are quite manageable.

We have shown that for the same number of elements, the pairing scheme has about the same ciphertext size as the Paillier's and the RLWE has about four times of the Paillier's. In the following we show only the Paillier cost difference between dense and sparse representations if the vector/matrix is the same for different encryption methods.

Data Contributors' Costs. Table 8 shows the average contributors' costs for sparse submission with different encryption methods. The actual costs for each data contributor should vary according to their original node degree. The ciphertext packing of RLWE cannot be used for sparse encoding anymore. Comparing it with the dense submission costs in Table 4, we can see that sparse encoding for the Paillier-based method can dramatically reduce the contributor's costs, while the RLWE's costs are about the same with the RLWE-P's costs for dense submission.

TABLE 7: The perturbation parameters and results. "orig. $|E|$": the number of original edges. "pert. $|E|$": the number of edges after perturbation. "%inc.": percentage of increase.

Dataset	nbins	nodes/bin	orig. $\lvert E\rvert$	pert. $\lvert E\rvert$	% inc.
Facebook	100	40	84243	99965	18.66
Twitter	1000	76	1242390	1527286	22.93
GPlus	2000	52	12113501	13228599	9.21

TABLE 8: Contributor's Average Cost for sparse submission.

Method	Encrypt A_i (seconds)			Upload $\mathrm{E}(A_i)$ (MB)		
	FB	Twitter	GPlus	FB	Twitter	GPlus
Paillier	0.04	0.03	0.22	0.006	0.005	0.032
RLWE	0.64	0.51	3.28	12.1	9.6	61.9

Cloud-side Costs The cloud side storage cost is the sum of all data contributors' submitted data. Table 9 summarizes these costs. The costs are about 100-1000 times less than the dense-matrix ones for Paillier, while RLWE without ciphertext packing has slightly less costs than the dense matrix with packing.

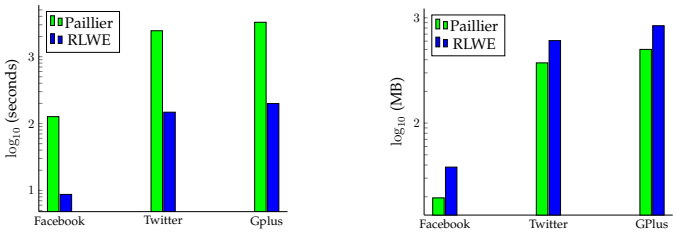
TABLE 9: The cloud storage costs with sparse submission. (MB: megabytes, GB: gigabytes, TB: terabytes)

Format	Facebook	Twitter	GPlus
Paillier	24.4MB	372.9MB	3.2GB
RLWE	47.8GB	729.8GB	6.3TB

Data Owner's Costs. Note that for the Lanczos method, sparse representation does not affect the data owner's costs, as in both sparse and dense matrix representations dense vectors have to be used in data owner's computation. Thus, our comparison will be on the Nyström method.

According to the protocol, the data owner's communication costs in the AHE-based Nyström method include downloading $E(W)$, $E(C\bar{V})$, and $E(C\Delta)$, and uploading $\bar{V}$ and Δ (in 128 bits per value) in plaintext. The computation cost is dominated by decrypting $E(W)$, $E(C\bar{V})$, and $E(C\Delta)$. In contrast, the communication costs in the RLWE-based method consist of downloading $E(W)$ and $E(CV)$ and uploading $E(V)$, where $E(W)$ are encoded without packing due to the sparse nature, but $E(V)$ and (CV) can use packing. Similarly, the computation cost is dominated by encrypting V (packed) and decrypting $E(W)$ (non-packed) and $E(CV)$ (packed).

Figure 6 summarizes the comparison. Due to the randomness of the sparse submission results, the numbers are the averages based on the statistics given by Table 7. Interestingly, it shows a similar pattern to Figure 4 for the Lanczos method, i.e., the Paillier-based method has lower communication costs but higher computational costs for the data owner. Table 10 shows the detailed comparison on the data owner's costs for Lanczos and Nyström on the largest matrix Gplus. The computational costs of the Nyström method is about 1/4 to 1/5 of the Lanczos method's, while the savings on communication are even larger: reduced



(a) Enc. and Dec. costs. (b) Communication costs.

Fig. 6: Data owner's costs for the privacy-preserving Nyström methods.

to about 1/6 for the Paillier-based method and 1/11 for the RLW-based method. This is consistent with the earlier complexity analysis (Table 1). For example, in terms of Lan-SHE and Ny-SHE, with $t = 30$ and $k = 10$, Lan-SHE's cost is about 5-6 times of Ny-SHE's. Between the Nyström algorithms, the Paillier-based method has a lower cost in communication (501MB vs. 840MB), while the RLWE-based one has much less computational time (about 1/18 of the Paillier-based), which seems more appealing for the data owner. However, the RLWE-based method still needs much larger cloud storage as shown in Table 9, which might be improved by a better RLWE storage encoding scheme.

TABLE 10: Data owner's costs by using Nyström on sparse data and Lanczos on dense/sparse Gplus data, encrypted by Paillier and RLWE, respectively. h: hours

Method	Ny-Sparse		Lanczos-Sparse/Dense	
	Comm.	Compute	Comm.	Compute
Paillier	**501MB**	**0.9h**	2.8GB	5h
RLWE	**841MB**	**0.05h**	10GB	0.2h

5 Related Work

There are a few recent studies on the application of garbled circuits (GC) [18], [35] and RLWE [7] for data mining algorithms. GC and AHE have been used by Nikolaenko et al. [26] for matrix factorization. However, it comes with high communication overhead and execution time. As noted in the paper, the execution time for one iteration of the solution on 4096x4096 matrix is almost 5000 seconds, and the communication cost is about 40GB per iteration, making it almost impractical for datasets of larger sizes as considered in our work. ML Confidential [17] uses FHE to learn Linear Means Classifier and Fisher's Linear Discriminant Classier from the encrypted data. However, the result shows that FHE is very expensive for the tasks even for very small training datasets.

Several matrix computation approaches have been proposed with methods other than encryption to ensure security. Atallah et al. [3] present secure outsourcing solutions that are specific to large-scale systems of linear equations and matrix multiplication applications with random noise masking. Their solutions fall short as they leak private information, depend on multiple non-colluding servers, or require a large communication overhead. Wang et al. [33] use an iterative approach for solving linear equations via client-cloud collaboration and matrix perturbation. However, there are several problems making it difficult to apply in practice. First, it requires that the entire unencrypted matrix be present at the client side in the initial setup. Secondly, the client side must perform a problem transformation step with a computation cost of $O(N^2)$. These weaknesses render this approach as impractical for big matrices and do not fully utilize the cloud infrastructures.

Bost et al. [6] consider applying already learned classifiers in an encrypted form (i.e., encrypted classifier parameters) to encrypted data to get classification results. However, applying classifiers is more about evaluating a function, which is a small-scale problem. Thus, many expensive operations can be possibly applied, which are impractical for mining large datasets.

Privacy-preserving graph data publishing [36] is slightly related to our work. However, it has a totally different problem setting. Graph data publishing wants to share the graph data but needs to address the privacy attacks from the curious data miners. However, the attacks with background knowledge cannot be thoroughly discovered and understood. Thus, differential privacy for graph analysis becomes popular in recent years [20], [34]. We do not aim to publish graph structures. However, since most graph matrices have special structures (i.e., non-zero entries represent the edges), simple sparse encoding exposes too much information. Our method is equivalent to adding fake edges to satisfy differential privacy [36]. However, this edge addition does not change data integrity: the added entries are encrypted 0s, which do not affect the matrix computation.

6 Conclusions

We present a cloud-centric framework for privacy-preserving spectral analysis of large matrices, which provides strong privacy guarantee protecting from honest-but-curious cloud providers. It allows data contributors to submit encrypted graph data to the cloud, and the analysis is done via secure protocols between the data owner and the cloud. The framework succeeds in outsourcing the expensive $O(N^2)$ computations to the cloud in a secure manner, and limiting in-house computations to $O(N)$ for the resource-restricted data owner and data contributors.

We design two privacy-preserving algorithms for spectral analysis: privacy-preserving Lanczos and Nyström algorithms, and study their constructions with somewhat homomorphic encryption (SHE) methods (e.g., the RLWE encryption method) and additive homomorphic encryption (AHE) methods (e.g., the Paillier encryption). The AHE methods need to protect the plaintext operands from adversaries, for which we designed masking methods that provide desired privacy guarantee and allow the data owner to recover in $O(N)$ complexity. The privacy-preserving Nyström method benefits from sparse big matrices, for which we have designed the privacy-preserving sparse data submission algorithm for data contributors to achieve the balance between data sparsity and privacy. The Nyström method on sparse data brings significantly cost reductions to data owner. Among different construction methods, the RLWE-based methods have less computational costs due to the ciphertext packing technique, while the Paillier-based methods save significantly in cloud storage and data owners' communication costs.

As a part of the future work, we will focus on improving the RLWE-based method and investigate the technical challenges with evolving graphs. We will try to optimize the algorithms in HELib to reduce the storage and communication costs. We will also study incremental updating methods for evolving graphs to minimize the overall costs while preserving the same level of privacy guarantee.

7 Acknowledgment

This work is supported by the National Science Foundation under Grant 1245847. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] C. C. Aggarwal and H. Wang. *Managing and Mining Graph Data*. Springer, 2010.
[2] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele. Secure computation on floating point numbers. In *Annual Network and Distributed System Security Symposium*, 2013.
[3] M. J. Atallah and K. B. Frikken. Securely outsourcing linear algebra computations. In *ACM Symposium on Information, Computer and Communications Security*, pages 48–59, 2010.
[4] P. Berkhin. A survey on pagerank computing. *Internet Mathematics*, 2:73–120, 2005.
[5] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *International Conference on Theory of Cryptography (TCC)*, pages 325–341. Springer-Verlag, 2005.
[6] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *Annual Network and Distributed System Security Symposium (NDSS)*, 2015.
[7] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science Conference (ITSC)*, pages 309–325, 2012.
[8] D. Catalano and D. Fiore. Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1518–1529, 2015.
[9] A. Chen. Gcreep: Google engineer stalked teens, spied on chats. *Gawker, http://gawker.com/5637234/*, 2010.
[10] J. K. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*. Cambridge University Press, 1985.
[11] M. L. Curtis. *Matrix Groups*. Universitext, 1984.
[12] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM CCS*, pages 79–88, 2006.
[13] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.
[14] C. Dwork. Differential privacy. In *International Colloquium on Automata, Languages andProgramming*, pages 1–12. Springer, 2006.
[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] T. Graepel, K. Lauter, and M. Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pages 1–21, 2013.
[18] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Conference on Security*, pages 35–35, 2011.
[19] I. T. Jolliffe. *Principal Component Analysis*. Springer, 1986.
[20] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. *Theory of Cryptography*, 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] X. Meng, S. Kamara, K. Nissim, and G. Kollios. Grecs: Graph encryption for approximate shortest distance queries. In *ACM CCS*, pages 504–517, New York, NY, USA, 2015. ACM.
[24] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of cloud computing security workshop*, pages 113–124, New York, NY, USA, 2011. ACM.
[25] M. E. J. Newman. Spectral methods for community detection and graph partitioning. *Phys. Rev. E*, 88:042822, Oct 2013.
[26] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh. Privacy-preserving matrix factorization. In *ACM SIGSAC conference on Computer and communications security*, pages 801–812. ACM, 2013.
[27] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238. Springer-Verlag, 1999.
[28] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Annual ACM symposium on Theory of computing*, pages 84–93, 2005.
[29] 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*, pages 199–212, New York, NY, USA, 2009.
[30] B. Scholkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.
[31] N. Smart and F. Vercauteren. Fully homomorphic simd operations. Cryptology ePrint Archive, Report 2011/133, 2011.
[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 *Advances in Knowledge Discovery and Data Mining*. Springer Berlin Heidelberg, 2013.
[35] A. C. Yao. How to generate and exhange secrets. In *IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.
[36] 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.

Sagar Sharma is currently a PhD student in the Department of Computer Science and Engineering at Wright State University. His research interests include privacy-preserving outsourced data mining, big data, cloud computing, and Internet of Things.

James Powers is currently a PhD student in the Department of Computer Science and Engineering at Wright State University. His research interests include big data and privacy&security.

Keke Chen is an associate professor in the Department of Computer Science and Engineering and directs the Data Intensive Analysis and Computing (DIAC) Lab of the Ohio Center of Excellence in Knowledge-Enabled Computing (the Kno.e.sis Center) at Wright State University. He earned his Ph.D. degree in Computer Science from Georgia Institute of Technology in 2006. His current research areas include secure data services and mining of outsourced data, privacy issues of social computing, cloud computing, Internet of things, healthcare informatics, and big data. During 2006-2008, he was a senior research scientist at Yahoo! Labs, working on web search ranking, cross-domain ranking, and web-scale data mining. He owns three patents for his work in Yahoo! Labs.