1-2020

# PGAS: Privacy-preserving graph encryption for accurate constrained shortest distance queries

Can ZHANG
*Beijing Institute of Technology*

Liehuang ZHU
*Beijing Institute of Technology*

Kashif SHARIF
*Beijing Institute of Technology*

Chuan ZHANG
*Beijing Institute of Technology*

Ximeng LIU
*Singapore Management University*, xmliu@smu.edu.sg

## Citation

ZHANG, Can; ZHU, Liehuang; SHARIF, Kashif; ZHANG, Chuan; and LIU, Ximeng. PGAS: Privacy-preserving graph encryption for accurate constrained shortest distance queries. (2020). *Information Sciences*. 506, 325-345.
**Available at:** https://ink.library.smu.edu.sg/sis_research/5908

# PGAS: Privacy-preserving graph encryption for accurate constrained shortest distance queries

Can Zhang[a], Liehuang Zhu[a, *], Chang Xu[a ,*], Kashif Sharif[a], Chuan Zhang[a], Ximeng Liu [b,c,d]

a School of Computer Science & Technology, Beijing Institute of Technology, Beijing, China

b College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China

c Fujian Provincial Key Laboratory of Information Security of Network Systems, Fuzhou, China

d School of Information Systems, Singapore Management University, Singapore

Published in Information Sciences, January 2020, 506, Pages 325-345. DOI: 10.1016/j.ins.2019.07.082

## Abstract

The constrained shortest distance (CSD) query is used to determine the shortest distance between two vertices of a graph while ensuring that the total cost remains lower than a given threshold. The virtually unlimited storage and processing capabilities of cloud computing have enabled the graph owners to outsource their graph data to cloud servers. However, it may introduce privacy challenges that are difficult to address. In recent years, some relevant schemes that support the shortest distance query on the encrypted graph have been proposed. Unfortunately, some of them have unacceptable query accuracy, and some of them leak sensitive information that jeopardizes the graph privacy. In this work, we propose **P**rivacy-preserving **G**raph encryption for **A**ccurate constrained **S**hortest distance queries, called **PGAS**. This solution is capable of providing accurate CSD queries and ensures the privacy of the graph data. Besides, we also propose a secure integer comparison protocol and a secure minimum value protocol that realize two kinds of operations on encrypted integers. We provide theoretical security analysis to prove that PGAS achieves *CQA-2 Security* with less privacy leakage. In addition, the performance analysis and experimental evaluation based on real-world dataset show that PGAS achieves 100% accuracy with acceptable efficiency.

Keywords: Cloud computing, Constrained shortest distance query, Graph encryption, Outsourced computing

## 1. Introduction

Graphs and graph data have been used in many fields of sciences and engineering for a long time. With the advancements in information technology, it has not only found its use in computer sciences but has enabled digitization of graph data for other domains. These domains include large scale Internet topologies, online social networks, biometric networks, communication systems, road networks, and so on. Similarly, with its widespread usage, a handful of tools have also been proposed in order to analyze and process massive graphs (e.g., GraphLab [23], TurboGraph++ [16] and GraphBase).

The shortest distance query has been considered as one of the most fundamental operations of graphs and has a wide range of applications. In online social networks, assume that Alice wants to meet a stranger Bob, the shortest distance query can return the minimum number of intermediate nodes between Alice and Bob. Compared to the shortest distance

* Corresponding author. E-mail addresses: liehuangz@bit.edu.cn (L. Zhu), xuchang@bit.edu.cn (C. Xu)

query, constrained shortest distance (CSD) query is a specialized kind that considers both the shortest distance as well as cost conditions. For example, in road networks, CSD queries can be used if a user intends to find the shortest distance from the source $s$ to the destination $t$ while maintaining the total time cost below the threshold $\theta$.

The graph owners can benefit from cloud storage systems by outsourcing the massive graph data to third-party servers. This will reduce the maintenance and management costs for such organizations. However, this increases the risk of potential leakage of data, which may compromise the user's privacy. To tackle this privacy challenge, graph owners can encrypt their graph data before outsourcing it to the cloud server, however, simply encrypting the graph data results in loss of querying abilities of graphs. Taking the CSD query as an example, some methods have been proposed to solve the approximate or exact (accurate) CSD query problems on plain graphs [11,32,33,35]. Unfortunately, these schemes cannot be directly used on encrypted graphs.

In order to alleviate the privacy concerns, some methods have been presented in the literature to protect access privacy [38,44], query privacy [39], identity privacy [13], data privacy [42,43], and location privacy [14] in cloud computing environments. Chase et al. [5] first proposed the concept of graph encryption to protect graph data privacy and query privacy. Using such encryption methods, graph owners can outsource the encrypted graph data to a semi-honest cloud server without losing the querying abilities. A series of graph encryption schemes that support the shortest distance query have also been proposed in [24,30]. These schemes make use of cryptographic primitives (e.g., Homomorphic Encryption, Pseudo-Random Function, and Order Revealing Encryption) to encrypt the graph itself or the corresponding pre-generated index. Unfortunately, some of the existing schemes do not support constraint filtering during the shortest distance query processes, which means that they do not support the CSD query. Moreover, some of them only return an approximate query result, and some of them even have unacceptable leakage that jeopardizes the graph privacy.

In light of these shortcomings in existing schemes, we propose a privacy-preserving graph encryption scheme that supports accurate constrained shortest distance (ACSD) queries. Our scheme makes use of the Paillier Cryptosystem with Threshold Decryption to encrypt the distance and cost values in a graph's Two-Hop Cover Label. Our scheme also separates the storage and computation of outsourced data which achieves advanced privacy protection. We also propose a privacy-preserving security integer comparison protocol and a secure minimum value protocol. Hence, the scheme can return ACSD query results with acceptable leakage.

To the best of our knowledge, this is the first graph encryption scheme that supports ACSD queries (i.e., it returns accurate CSD query results) with privacy protection.

Following are the major contributions of this work:

- We propose the complete system architecture of the graph encryption scheme that supports ACSD queries. Our architecture separates the storage and computation of outsourced data, which realizes advanced privacy protection.
- We present PGAS, the first graph encryption scheme that supports ACSD queries on encrypted graph data. PGAS uses the Constraint Filtering algorithm to filter the cost values. Besides, PGAS can return accurate query results to the user.
- We propose two novel protocols: The *Secure Integer Comparison* (*SIC*) protocol compares two encrypted integers, and the *Secure Minimum value* (*SMin*) protocol finds the minimum value of some encrypted integers. Both protocols directly operate the ciphertexts, so they do not reveal any information about plaintexts. In addition, these protocols can be used not only in our proposed scheme but also in other relevant application scenarios.
- We also present a strict security analysis of PGAS to prove that it can achieve *CQA2-Security*. We also make a throughout the theoretical analysis and comprehensive experimental evaluation based on real-world datasets to show that our proposed scheme has acceptable efficiency and achieves 100% accuracy.

The rest of this paper is organized as follows. In Section 2, we describe related works. In Section 3, we make a brief introduction of the Two-Hop Cover Label and the Paillier Cryptosystem with Threshold Decryption and define the graph encryption scheme that supports ACSD queries. The formalized scenario and security model are defined in Section 4. Detailed descriptions of PGAS are presented in Section 5, and in Section 6 we present two kinds of secure outsourced integer computation protocols *SIC* and *SMin*. Security analysis is given in Section 7. Performance analysis and experimental evaluation are presented in Section 8. Finally, Section 9 concludes this paper.

## 2. Related works

### 2.1. Constrained shortest distance query

CSD query is a special kind of shortest distance query on graphs. It has received much attention to its cost condition capabilities. Hansen [11] proposed a scheme to find the exact constrained shortest path based on enhanced Dijkstra's algorithm. Tsaggouris et al. [33] provided an optimized method to find the approximate constrained shortest path with improved efficiency. Both of these schemes are not practical for processing large graphs due to the high computation costs. In order to improve the efficiency, some schemes use a pre-generated index to speed up the query process. Storandt [32] proposed a method to find the exact constrained shortest path with an index, but it still requires significant query processing.

In recent years, a series of efficient schemes have been proposed to solve constrained shortest path problems. Wang et al. [35] presented a practical solution for index-based approximate constrained shortest path querying for large road networks. Bode et al. [2] provided a labeling algorithm to solve the shortest path problem under resource constraints with (k,2)-loop elimination. Yang et al. [40] proposed an approximate heuristic algorithm to find the shortest path over large-scale graphs, and experiments on real-world datasets show that the proposed algorithm outperforms the existing methods.

However, all of the above schemes are based on unencrypted graphs. Hence, if users want to outsource their unencrypted graph data to the cloud, sensitive information may be compromised. More specifically, the topology structure, the identity of all graph vertices, the distance and cost values between any two vertices, will be revealed to adversaries, which results in the risk of graph privacy leakage.

### *2.2. Graph privacy protection*

In recent years, some privacy-preserving methods have been presented in the literature to alleviate privacy concerns. Song et al. [31] first proposed Searchable Encryption which is widely used in Internet of Things [45], cloud computing [18,25,41] and data sharing scenarios [19,22]. The Searchable Encryption enables searching on encrypted data stored in untrusted servers.

Graph encryption is a generalization of Searchable Encryption. Chase et al. [5] first introduced a simple graph encryption scheme which supports adjacency and neighboring queries. Cao et al. [4] utilized the principle of "filtering-and-verification" and developed privacy-preserving queries over encrypted graph data.

Mouratidis et al. [26] used Private Information Retrieval to obtain the shortest paths without information leakage. Differential Privacy is also utilized to protect graph data [28]. However, in our scenario, the graph data is outsourced to the cloud, and the graph itself contains sensitive information. Hence, the above scheme is not applicable.

Oblivious RAM and Secure Multi-Party Computation are also utilized to tackle the privacy-preserving shortest path problems in [15,37]. However, these methods are not suitable for large-scale graphs.

Wang et al. [36] presented GraphProtector, a visual interface that helps users protect their identity privacy. GraphProtector supports hybrid privacy protection where different privacy-preserving schemes can be activated simultaneously. Sharma et al. [29] designed two data masking schemes, and utilized Additive Homomorphic Encryption (AHE) and Somewhat Homomorphic Encryption (SWHE) to realize privacy-preserving Lanczos algorithms and Nystrom algorithms. Experimental results show that SWHE-based approaches are computational efficient, while AHE-based approaches are communicational efficient. Unfortunately, both of the two schemes mentioned above do not support CSD queries.

Meng et al. [24] proposed three graph encryption schemes that support approximate shortest distance queries. In their proposed scheme *GraphEnc*$_3$, the authors make use of SWHE to encrypt a graph-based structure called *Distance Oracle*. *GraphEnc*$_3$ is semantically secure against a semi-honest cloud server, and it achieves $\mathcal{O}(1)$ communication cost between the user and the cloud server. However, it does not support the CSD query, and the query result is an approximate shortest distance.

Shen et al. [30] proposed a graph encryption scheme Connor which enables CSD queries. This scheme uses SWHE to encrypt the distances and uses Order-Revealing Encryption (ORE) [17] to encrypt the costs. The authors prove that there proposed scheme achieves *CQA-2 Security*. However, this scheme leaks the order information about costs, and the query result is also an approximate shortest distance. Hence, the provided privacy protection is deficient, and the query accuracy is unacceptable in practical scenarios.

To the best of our knowledge, none of the schemes discussed above support both ACSD queries and advanced privacy protection. This forms the basic motivation of our work.

### *2.3. Secure outsourced computation*

With the proliferation of cloud services, more and more users try to outsource their data to third-party cloud servers. However, the privacy of outsourced data needs to be protected.

Homomorphic Encryption (HE) has mostly been utilized to solve this problem. Some cryptosystems only support one kind of homomorphic operation, such as additive homomorphic cryptosystem (e.g., Paillier) and multiplicative homomorphic cryptosystem (e.g., RSA). Some cryptosystems allow limited homomorphic operations, known as SWHE. For instance, the BGN cryptosystem proposed by Boneh et al. [3] is a typical example that can support a limited number of additive homomorphic operations and only one multiplicative homomorphic operation.

Gentry [8] used ideal lattices to construct the first Fully Homomorphic Encryption (FHE) scheme, followed by a series of FHE schemes [9,34]. Unfortunately, it is not practical to use FHE schemes to solve real-world problems because of their high complexity.

Liu et al. [21] proposed a toolkit for efficient and privacy-preserving outsourced calculation under multiple encrypted keys by utilizing a distributed two-trapdoor public-key cryptosystem. Liu et al. [20] proposed POCR, which can perform several kinds of calculations on rational numbers without compromising the privacy of outsourced data. These schemes use the Paillier-based cryptosystem which can be perceived as an additive homomorphic encryption cryptosystem.

From the analysis presented, we conclude that none of these schemes can be directly used to solve the ACSD query problem on encrypted graphs.

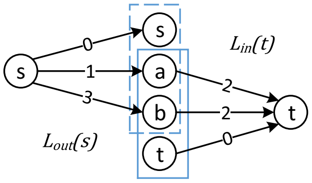


**Fig. 1.** An example of 2HCL.

## 3. Preliminaries

In this section, we present the basic information related to Two-Hop Cover Label, graph encryption schemes which support ACSD queries, and paillier cryptosystem with threshold decryption.

### 3.1. Two-Hop cover label

Two-Hop Cover Label (2HCL) [1] is a kind of label $L$ which can be pre-generated as an index to accelerate the process of finding shortest distance in a graph $G = (V, E)$. Note that we refer $G$ as a directed graph in this paper unless otherwise specified.

For each vertex $v \in V$, $L(v) = \{L_{out}(v), L_{in}(v)\}$ represents a subset of label $L$ associated with $v$. Each entry $(u,\ d_{uv}) \in L_{in}(v)$ corresponds to the shortest distance $d_{uv}$ from $u$ to $v$, and $(w,\ d_{vw}) \in L_{out}(v)$ corresponds to the shortest distance $d_{vw}$ from $v$ to $w$. When querying the shortest distance from $s$ to $t$, the following equation can be used to calculate the shortest distance from $s$ to $t$:

$$Query(s, t, L) = \min\{d_{sv} + d_{vt}\} \tag{1}$$

where $(v,\ d_{sv}) \in L_{out}(s)$, $(v,\ d_{vt}) \in L_{in}(t)$. If $L_{out}(s)$ and $L_{in}(t)$ do not share the same vertex, we define $Query(s, t, L) = \infty$. 2HCL can be formally defined as:

**Definition 1** (Two-Hop Cover Label)**.** Let $L$ be a label of a directed graph $G$, and $d(s,\ t)$ represents the shortest distance between $s$ and $t$. If for any $s,\ t \in V$, $d(s, t) = Query(s, t, L)$ holds, it can be concluded that $L$ is a Two-Hop Cover Label of $G$.

Fig. 1 gives an example of a 2HCL $L$, where $L_{out}(s) = \{(s, 0), (a, 1), (b, 3)\}$ and $L_{in}(t) = \{(a, 2), (b, 2), (t, 0)\}$. It can be observed that $a$ and $b$ are vertices that appear both in $L_{out}(s)$ and $L_{in}(t)$. According to Eq. (1), we can calculate $Query(s, t, L) = \min\{1 + 2, 3 + 2\} = 3$. Because $L$ is a 2HCL, $d(s, t) = Query(s, t, L) = 3$, the shortest distance between $s$ and $t$ is 3.

Note that the term *label* is used to represent 2HCL in the following sections.

### 3.2. Graph encryption supporting ACSD queries

ACSD query is a special kind of CSD query which returns an accurate shortest distance instead of an approximate value. First we introduce the concept of CSD query. Let $G = (V, E)$ be a graph, where each edge $e \in E$ has a distance $d_e$ and the cost $c_e$ which satisfy $d_e \geq 0$ and $c_e \geq 0$. Assume that a path $P = \{e_1,\ \cdots,\ e_n\}$, with the total distance and cost of this path to be $d_P = \sum_{i=1}^{n} d_{e_1}$ and $c_P = \sum_{i=1}^{n} c_{e_1}$, respectively.

As shown in [12], the CSD query problem is NP-hard, hence, literature aims at solving an approximate CSD query which can be defined as $\alpha$*-CSD Query*. Given two vertices $s,\ t \in V$ as the source and distance, and a cost threshold $\theta$, assume that the exact shortest distance is $d_{st}$, an $\alpha$-CSD query is used to find the approximate shortest distance $d_P$ between $s$ and $t$ which satisfies $d_{st} \leq d_P \leq \alpha \cdot d_{st}$, while the total cost of the corresponding path must not exceed the threshold $\theta$. In fact, if we set $\alpha = 1$, then the $\alpha$-CSD query can be ACSD query. Hence, we formalize the definition of ACSD queries as follows.

**Definition 2** (ACSD Query)**.** Given a source $s$, a destination $t$, and a cost threshold $\theta$, an ACSD query returns the distance $d_P$ and the cost $c_P$ which satisfy $d_P = d_{st}$ and $c_P = c_{st}$, where $d_{st}$ represents the shortest distance, for which the total cost $c_{st}$ does not exceed $\theta$.

Fig. 2 shows an example of ACSD query over a 2HCL, for which the entries include both distance and cost values. There are two entries associated with $b$ in $L_{out}(s)$ due to the influence of costs. Assuming that we set the cost threshold $\theta = 5$, we can find that only two paths $P_1 = (e_2, e_5)$ and $P_2 = (e_3, e_7)$, for which the total cost does not exceed $\theta$. The shorter distance among paths $P_1$ and $P_2$ is that of 5 (given by $P_2$), hence the constrained shortest distance between $s$ and $t$ is 5.

Note that in this work, the key research focus is not on generating of 2HCL that supports ACSD queries. Hence, we assume that the 2HCL method in [35] can support ACSD queries on plain graphs. The assumption is valid because if we set $\alpha = 1$, the CSD query result will be an ACSD result.

Below we provide the formal definition of graph encryption scheme supporting ACSD queries. The reason to use the term *graph encryption scheme* is for simplicity.

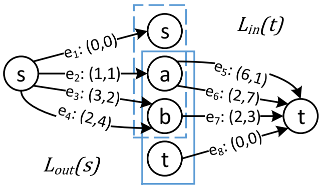


**Fig. 2.** An example of ACSD query over a 2HCL.

**Definition 3** (Graph Encryption)**.** A graph encryption scheme $\Lambda = (\mathit{KeyGen}, \mathit{GraphEnc}, \mathit{TokenGen}, \mathit{DistQuery})$ consists of four polynomial-time algorithms that work as given below:

- $(K, pk, sk) \leftarrow \mathit{KeyGen}(\lambda)$ is a probabilistic algorithm that takes the security parameter $\lambda$ as input, and it outputs a secret key $K$, and a key-pair $(pk, sk)$.
- $EL \leftarrow \mathit{GraphEnc}(G, K, pk)$ is a probabilistic algorithm that takes a graph $G = (V, E)$, a secret key $K$ and a public key $pk$ as input, and it outputs an encrypted label $EL$.
- $T_{st} \leftarrow \mathit{TokenGen}(q, K, pk)$ is a probabilistic algorithm that takes an ACSD query $q = (s, t, \theta)$, a secret key $K$ and a public key $pk$ as input, and it outputs a query token $T_{st}$.
- $D_{st} \leftarrow \mathit{DistQuery}(EL, pk, T_{st})$ is a deterministic algorithm that takes an encrypted label $EL$, a public key $pk$ and a query token $T_{st}$ as input, and it outputs an encrypted shortest distance $D_{st}$ as the query result. The user can decrypt $D_{st}$ by using $sk$ and get the plain shortest distance $d_{st}$ as the user's ACSD query result.

Note that if a graph encryption scheme does not support CSD queries (e.g., GRECS [24]), it does not need the cost threshold $\theta$ in the *TokenGen* algorithm. If a graph encryption scheme which supports only approximate CSD queries (e.g., Connor [30]), the output $D_{st}$ of *DistQuery* algorithm is only an encrypted approximate shortest distance. In section 8, we use both theoretical and experimental analysis to prove that our proposed scheme PGAS can achieve 100% accuracy compared with GRECS and Connor.

### *3.3. Paillier cryptosystem with threshold decryption*

When encrypting and computing outsourced data under the naive Paillier-based cryptosystem, the cloud server must get access to the user's private key to decrypt data, which means that the cloud server can collect all the plaintexts and the corresponding encrypted result.

To avoid such security risks, some cryptosystems have been proposed to achieve ($t$, $n$) threshold decryption by splitting the private key into different parts. The Paillier-based Cryptosystem with Threshold Decryption (PCTD) is a typical kind of threshold decryption [20], which includes following six algorithms:

*KeyPairGen($\lambda$)*: This probabilistic algorithm generates a key pair $(pk, sk)$ which will be used to encrypt the outsourced integers. Let $\lambda$ be the security parameter, first the algorithm finds four distinct prime numbers $p$, $q$, $p'$, $q'$ which satisfy $p = 2p' + 1$, $q = 2q' + 1$ and $|p| = |q| = \lambda$. $p$ and $q$ are two strong prime numbers [6]. Then it computes $N = pq$, $\alpha = lcm(p - 1, q - 1)$, and chooses a generator g of order $(p-1)(q-1)/2$. In order to generate such a g easily, it randomly chooses $a \in \mathbb{Z}^*_{N^2}$ and compute $g = -a^{2N}$ [6]. The tuple $(N, \alpha)$ can be represented as the key pair $(pk, sk)$ of PCTD.

*Enc(m,pk)*: This probabilistic algorithm uses $pk$ (noted as $N$) to encrypt an integer $m$ and outputs an encrypted result $c$. Given an integer $m \in \mathbb{Z}_N$, the algorithm first randomly chooses $r \in \mathbb{Z}_N$ and encrypts $m$ by computing

$$c = g^m \cdot r^N \bmod N^2 = (1 + mN) \cdot r^N \bmod N^2. \tag{2}$$

*Dec(c,sk)*: This deterministic algorithm uses $sk$ (noted as $\alpha$) to decrypt the ciphertext $c$ and obtain the decrypted integer $m$. The algorithm computes

$$m = c^\alpha = r^\alpha(1 + mN\alpha) \bmod N^2 = 1 + mN\alpha. \tag{3}$$

Since $\gcd(\alpha, N) = 1$, $m$ can be decrypted by computing

$$m = L(c^\alpha \bmod N^2)\alpha^{-1} \bmod N^2, \tag{4}$$

where $L(x) = \frac{x-1}{N}$.

*KeySplit(sk)*: This probabilistic algorithm splits the private key $sk$ and generates a partial private key set $\{sk_1, \cdots, sk_n\}$. The algorithm first randomly chooses $\delta$ which satisfies both $\delta \equiv 0 \bmod \alpha$, and $\delta \equiv 1 \bmod N^2$. Since $\gcd(\alpha, N^2) = 1$, according to Chinese remainder theorem, $\delta = \alpha \cdot (\alpha^{-1} \bmod N^2) \bmod \alpha N^2$. Then it defines a polynomial $q(x) = \delta + \sum_{i=1}^{k-1} r_i x_i$, where $r_1, \cdots, r_{k-1}$ are $k - 1$ numbers chosen randomly from $\mathbb{Z}^*_{\alpha N^2}$. Let $\alpha_1, \cdots, \alpha_n \in \mathbb{Z}^*_{\alpha N^2}$ be $n$ distinct nonzero elements which are chosen randomly and known to all the parties who participate in the computation. Then it computes $sk_i = q(a_i)$ as the partial key.

*PDec(c,sk$_i$)*: This deterministic algorithm uses the partial private key $sk_i$ to decrypt $c$ and outputs the partially decrypted result $c_i$ which can be computed as

$$c_i = c^{sk_i} \bmod N^2. \tag{5}$$

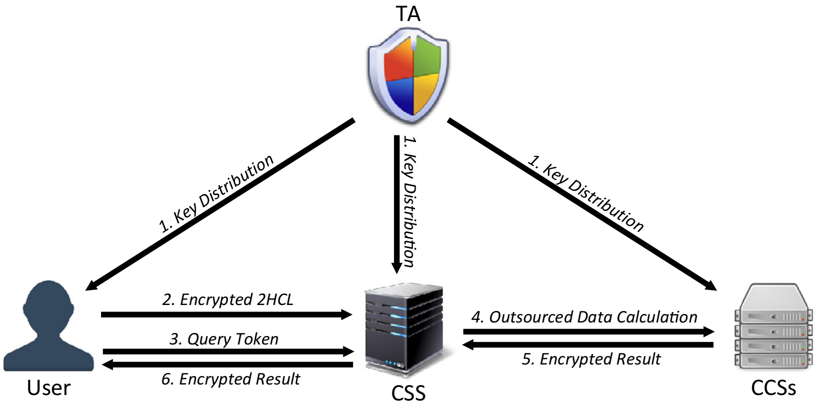


**Fig. 3.** System model.

*TDec*($c_1, \cdots, c_d$): This deterministic algorithm combines the partial decrypted results to get the complete decrypted result. When partially decrypted results $\{c_1, \cdots, c_d\}$ are received (where $k \le d \le n$), this algorithm randomly chooses $k$ distinct numbers between 1 and $d$ to generate a set $S = \{s_1, \cdots, s_k\}$. Then it first computes

$$T = \prod_{i=1}^{k} (c_{s_i})^{\Delta(s_i,0)} \bmod N^2, \tag{6}$$

where $\Delta(s_i, x) = \prod_{j \in S, j \neq s_i} \frac{x-\alpha_j}{\alpha_{s_i}-\alpha_j}$, and then it computes the decrypted result $m$ by calculating $m = L(T)$.

Moreover, given any $m \in \mathbb{Z}_n$, it satisfies

$$[m]^{N-1} = (1 + (N-1)m \cdot N) \cdot r^{N-1 \cdot N} \bmod N^2 = [-m], \tag{7}$$

where $[m]$ represents the encrypted result of $m$, so we can compute $[m]^{N-1}$ to get $[-m]$.

Suppose that $m_1$, $m_2$ are two plaintexts and $c_1$, $c_2$ are the corresponding ciphertexts which are encrypted by the same public key *pk*. If a cryptosystem is additive homomorphic, it satisfies the following equation:

$$Add(c_1, c_2) = Enc(m_1 + m_2), \tag{8}$$

where *Add* represents the operation of homomorphic addition. Note that PCTD is additive homomorphic which is based on the additive homomorphism of Paillier cryptosystem.

## 4. Problem formalization

This section presents the system, threat, and security model, along with the design goals of the proposed scheme.

### 4.1. System model

This section presents the system model which formalizes the process of ACSD queries in our proposed scheme PGAS. As can be seen from Fig. 3, the system comprises of four entities: Trust Authority (TA), User, Cloud Storage Server (CSS) and Cloud Computation Servers (CCSs).

The *TA* is tasked with key generation and distribution. By using the *KeyGen* algorithm, it first generates the *user*'s secret key $K$ and keypair (*pk, sk*), and provides them to the *user*. Then it sends *pk* to the *CSS*. Finally, it generates a partial private key set $PSK = \{sk_1, \cdots, sk_n\}$ and sends $sk_i$ to each $CCS_i$.

The *user* in this model is the owner of graph data. It first executes the *GraphEnc* algorithm to encrypt a graph $G$ by using $K$ and *pk*. Then it outsources the encrypted graph to the *CSS*. When it wants to obtain ACSD query results, it creates a token-based ACSD query request by executing the *TokenGen* algorithm. Following this, it sends the token to the *CSS* to query the accurate constrained shortest distance between any two vertices in encrypted graph data. When receiving the encrypted ACSD query result $D_{st}$ sent by the *CSS*, it decrypts $D_{st}$ and gets the query result $d_{st}$.

We assume that the *CSS* has enough storage capacity to store outsourced graph data owned by the *user*. When the *CSS* receives an ACSD query request created by the *user*, it executes the *DistQuery* algorithm. First, it parses the query token and finds corresponding indices. Then, it performs a series of protocols with *CCSs* to find the minimum distance under the cost threshold. At last, it receives an encrypted minimum distance $D_{st}$ from *CCSs* and forwards it to the *user*.

The *CCSs* consists of several cloud computation servers and they can provide online computation services. In our system, the *CCSs* mainly performs two kinds of encrypted integer computation: comparing the relationship of two encrypted integers, and finding the minimum value of some encrypted integers.

### 4.2. Threat model

In this work, the threat model assumes that the *TA* is *trustworthy*. This means the *TA* cannot leak any keys that have been generated, and the *TA* can resist any kind of attacks. We also assume that the *TA* uses secure channels to distribute these keys. Hence, the adversary cannot eavesdrop on distributed keys passively or modify them actively.

We assume that both the *CCSs* and *CSS* are *honest-but-curious*. This means that they will strictly execute the algorithms and follow the protocols, however, they will try to get some sensitive information that they should not know (e.g., the topology structure, the vertex identity, distance and cost values between two vertices).

We also assume that both the *CCSs* and the *CSS* are *vulnerable*, which means the adversary can compromise the *CSS* or part of *CCSs* in order to steal stored data and intermediate data when executing our algorithms and protocols.

Note that in our threat model, collusion between the *CSS* and *CCSs* is not allowed, in other words, they cannot collaborate to get sensitive information of the *user*'s outsourced graph data. Moreover, the adversaries cannot compromise both the *CSS* and the *CCSs* at the same time. These limitations of collusion and the adversary's ability are reasonable because in real life, cloud storage service and computation service are provided by two different providers and they cannot collude in order to get the *user*'s sensitive information. The adversary cannot control both cloud storage service providers and cloud computation service providers to perform effective attacks in real life either.

### 4.3. Security definition

Graph encryption can be considered as a special kind of Structured Encryption (SE), which generalizes previous work on Symmetric Searchable Encryption (SSE). In that case, the security definitions of SE and SSE are also suitable for graph encryption. We modify the security definition proposed in [5] and [7], which is called *CQA2-Security* (chosen-query attack security) and formalize it for our proposed graph encryption scheme PGAS.

**Definition 4** (CQA2-Security)**.** Let scheme $\Lambda = (KeyGen, GraphEnc, TokenGen, DistQuery)$ be a graph encryption scheme, and $\mathcal{L} = (\mathcal{L}_{Enc}, \mathcal{L}_{Query})$ be the (stateful) leakage function in this scheme. Given a security parameter $\lambda$, a challenger $\mathcal{C}$, an (semi-honest) adversary $\mathcal{A}$, and a simulator $\mathcal{S}$, we can conduct the following two probabilistic experiments:

(1). $Real_{\mathcal{C},\mathcal{A}}(\lambda)$:

- $\mathcal{A}$ chooses a graph $G$ and forwards it to $\mathcal{C}$.
- $\mathcal{C}$ first runs $KeyGen(\lambda)$ to generate a secret key $K$ and a keypair ($pk$, $sk$). Then it runs $GraphEnc(K, pk, G)$ to generate encrypted label $EL$ and returns it to $\mathcal{A}$.
- $\mathcal{A}$ adaptively generates a polynomial number of ACSD queries $Q = \{(s_1, t_1, \theta_1), \cdots, (s_n, t_n, \theta_n)\}$ and sends $Q$ to $\mathcal{C}$.
- For each ACSD query $q_i = (s_i, t_i, \theta_i) \in Q$, $\mathcal{C}$ runs $TokenGen(q_i, K, pk)$ to generate the corresponding query token $T_i$. Then $\mathcal{C}$ sends $\mathcal{T} = \{T_1, \cdots, T_n\}$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$ as the final output of this experiment.

(2). $Ideal_{\mathcal{C},\mathcal{A},\mathcal{S}}(\lambda)$:

- $\mathcal{A}$ chooses a graph $G$ and forwards it to $\mathcal{C}$.
- $\mathcal{C}$ forwards the output of leakage function $\mathcal{L}_{Enc}(G)$ to $\mathcal{S}$. After that, $\mathcal{S}$ simulates an encrypted label $EL^*$ and forwards it to $\mathcal{A}$.
- $\mathcal{A}$ adaptively generates a polynomial number of ACSD queries $Q = \{(s_1, t_1, \theta_1), \cdots, (s_n, t_n, \theta_n)\}$ and sends $Q$ to $\mathcal{C}$.
- For each ACSD query $q_i = (s_i, t_i, \theta_i) \in Q$, $\mathcal{C}$ sends the output of leakage function $\mathcal{L}_{Query}(q)$ to $\mathcal{S}$, where $q$ represents a series of queries that have been sent, $\mathcal{S}$ simulates the corresponding query token $T_i^*$. Then $\mathcal{C}$ sends $\mathcal{T}^* = \{T_1^*, \cdots, T_n^*\}$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$ as the final output of this experiment.

If for all Probabilistic Polynomial Time (PPT) adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ and a negligible function $negl(\lambda)$, which satisfies the in-equation

$$|\Pr[Real_{\mathcal{C},\mathcal{A}}(\lambda) = 1] - \Pr[Ideal_{\mathcal{C},\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq negl(\lambda), \tag{9}$$

then, in such a case, it is established that $\Lambda$ is $(\mathcal{L}_{Enc}, \mathcal{L}_{Query})$-secure against the CQA2 attack, i.e., this scheme can resist adaptive chosen-query attacks.

### 4.4. Design goals

Based on the earlier discussed models, here we define the following design goals of our graph encryption scheme PGAS.

#### 4.4.1. Privacy protection

The system should protect both graph privacy and query privacy. In other words, the *CSS* and the *CCSs* should not be able to determine any kind of sensitive information (e.g., topology structure, vertex identity, distance and cost values between two vertices) of original graph by analyzing encrypted outsourced data, the *user*'s historical queries, or the intermediate data during the querying processes.

**Table 1**
List of notations.

| Notation | Description |
|---|---|
| $G = (V, E)$ | Input graph, with a collection of $\underline{V}$ertex and $\underline{E}$dges |
| $\lvert V\rvert$, $\lvert E\rvert$ | Cardinality of vertices and edges in graph $G$ |
| $n$ | The number of *CCSs* |
| $\lambda$ | Security parameter |
| $K$ | Secret key |
| $(pk,\ sk)$ | Public key and private key |
| $sk_i$ | $CCS_i$'s partial private key |
| $L,\ EL$ | Plain and encrypted graph label |
| $L_{in}(u)$, $L_{out}(u)$ | Plain in- and out-label set associated with vertex $u$ |
| $g,\ h$ | Secure hash function |
| $I_{in,u}$, $K_{in,u}$ | Master encryption keys of in-label associated with set $L_{in}(u)$ |
| $I_{out,u}$, $K_{out,u}$ | Master encryption keys of out-label associated with set $L_{out}(u)$ |
| $H_v$ | Vertex $v$'s hash value |
| $EL_{out}[I_{out,\ uv}]$ | $EL_{out}$'s encrypted label element under the index $I_{out,\ uv}$ |
| $T_{st}$ | Query token of $q = (s, t, \theta)$, where $s,\ t \in V$ |
| $d_{st}$, $c_{st}$ | Plain distance and cost value from $s$ to $t$ |
| $D_{st}$, $C_{st}$ | Encrypted distance and cost value from $s$ to $t$ |
| $\theta$, $\Theta$ | Plain and encrypted cost threshold of ACSD queries |
| $k$ | Output length of PCTD's ciphertexts |

### *4.4.2. Acceptable efficiency*

The system should be efficient during both encryption and query processes. This means:

- The encryption time and storage space should be acceptable.
- The *user* should receive the query result sent by the *CSS* in time.
- The communication cost (i.e., the number of bits exchanged by the *user* and the *CSS*) should be as minimal as possible.

If the efficiency is unacceptable, communication and storage resources will be wasted, and it will become a significant bottleneck of this scheme especially in the case of multiple and concurrent ACSD queries.

### *4.4.3. High accuracy*

The system should return the ACSD query result instead of an approximate result to the *user*. Hence, the system should achieve 100% accuracy for each ACSD queries.

## 5. PGAS scheme

In this section, we present the pseudo-code and process description of all algorithms used in the Privacy-Preserving Graph Encryption for Accurate Constrained Shortest Distance Query scheme.

### *5.1. Notations*

The notations used in this work are shown in Table 1. The algorithms and protocols also use the same symbols, unless otherwise explicitly mentioned.

In order to achieve advanced privacy protection, we use two hash functions $g$ and $h$ which are modeled as random oracles, as is illustrated in Eq. (10):

$$\begin{aligned} &h : \{0,1\}^{\lambda} \times \{0,1\}^{*} \rightarrow \{0,1\}^{\lambda} \\ &g : \{0,1\}^{\lambda} \times \{0,1\}^{*} \rightarrow \{0,1\}^{\lambda+2k} \end{aligned} \tag{10}$$

where $\lambda$ is the security parameter and $k$ is the output length of PCTD's ciphertexts.

### *5.2. Construction of PGAS*

We construct our graph encryption scheme $PGAS = (KeyGen,\ GraphEnc,\ TokenGen,\ DistQuery)$ in order to achieve advanced privacy protection and 100% accuracy.

The *KeyGen* shown in Algorithm 1 works as follows. Given a security parameter $\lambda$, the *TA* first randomly generates a $\lambda$-bit string as the *user*'s secret key $K$. Then the *TA* generates a keypair $(pk,\ sk)$ for the *user* and a partial private key set $PSK = \{sk_1,\ \cdots,\ sk_n\}$ for $n$ *CCSs*. It then sends $(K,\ pk,\ sk)$ to the *user*, sends $pk$ to the *CSS* and sends $sk_i$ to corresponding $CCS_i$.

The *GraphEnc* in Algorithm 2 works as follows. The *user* first generates a plain label $L = \{L_{out}, L_{in}\}$ of graph $G$. Then for each vertex $u \in V$, it encrypts the plain label $L(u)$ as given in lines 3–23, and finally gets an encrypted label $EL = \{EL_{out}, EL_{in}\}$ which is the output of this algorithm.

**Algorithm 1:** KeyGen.

```
Input: A security parameter $\lambda$.
Output: A secret key $K$, a keypair $(pk, sk)$, and a partial private key set $PSK$.
begin
    generate $K \xleftarrow{\$} \{0, 1\}^{\lambda}$.
    generate $(pk, sk) = KeyPairGen(\lambda)$.
    generate $PSK = KeySplit(sk)$.
    return $K$, $(pk, sk)$ and $PSK$.
end
```

**Algorithm 2:** GraphEnc.

```
   Input: A graph $G = (V, E)$, a secret key $K$, and a public key $pk$.
   Output: An encrypted label $EL$.
1  begin
2      generate 2-Hop Cover Label $L = \{L_{in}, L_{out}\}$ of graph $G$.
3      foreach $u \in V$ do
4          calculate $I_{out,u} = h(K, u||0)$.
5          calculate $K_{out,u} = h(K, u||1)$.
6          set counter $ctr = 0$.
7          foreach $(v, d_{uv}, c_{uv}) \in L_{out}(u)$ do
8              calculate $H_v = h(K, v)$.
9              calculate $I_{out,uv} = h(I_{out,u}, ctr)$.
10             calculate $K_{out,uv} = g(K_{out,u}, ctr)$.
11             compute $D_{uv} = Enc(pk, d_{uv})$.
12             compute $C_{uv} = Enc(pk, c_{uv})$.
13             set $\Phi_{uv} = H_v||D_{uv}||C_{uv}$.
14             set $EL_{out}[I_{out,uv}] = K_{out,uv} \oplus \Phi_{uv}$.
15             set $ctr = ctr + 1$.
16         end
17         calculate $I_{in,u} = h(K, u||2)$.
18         calculate $K_{in,u} = h(K, u||3)$.
19         set counter $ctr = 0$.
20         foreach $(v, d_{vu}, c_{vu}) \in L_{in}(u)$ do
21             repeat above procedure but replace subscript out with in.
22         end
23     end
24     return $EL = \{EL_{in}, EL_{out}\}$.
25 end
```

Note that both the plain and encrypted label consist of its in-label and out-label. The process of out-label $L_{out}$'s encryption is shown in lines 4–16. For each vertex $u \in V$, the *user* first calculates $I_{out,u}$ and $K_{out,u}$ which will be used to generate the keys for index obfuscation and re-encryption. Then for each entry ($v$, $d_{uv}$, $c_{uv}$) $\in L_{out}(u)$, the *user* calculates a hash value $H_v$ as the pseudo-identity of $v$ to hide its real identity, and uses a counter *ctr* to generate $I_{out,\ uv}$ and $K_{out,\ uv}$. It is obvious that $I_{out,\ uv}$ and $K_{out,\ uv}$ are unique because both tuples ($I_{out,u}$, *ctr*) and ($K_{out,u}$, *ctr*) are unique.

After generating these keys, the *user* encrypts the distance $d_{uv}$ and cost $c_{uv}$ from $u$ to $v$ and gets the encrypted result $D_{uv}$ and $C_{uv}$. Then it performs re-encryption of ($H_v||D_{uv}||C_{uv}$) to generate the pseudo-identity, the encrypted distance, and the encrypted cost indistinguishable by executing an XOR operation with a unique key $K_{out,\ uv}$. The encrypted result is stored in $EL_{out}$ with the obfuscated index $I_{out.uv}$ that makes the location of each entry's storage indistinguishable, which is an added benefit. Because $I_{out,\ uv}$ is also unique, hence there is no conflict when constructing $EL_{out}$. Based on index obfuscation and re-encryption, the *CSS* cannot infer the storage location or plaintext of each encrypted entry ($v$, $d_{uv}$, $c_{uv}$) in $EL_{out}$ without knowing the secret key $K$ and the private key *sk*.

Similar to the user process for $L_{out}$, next it encrypts the in-label $L_{in}$. First it calculates $I_{in,u}$ and $K_{in,u}$ which will be used on encrypting $L_{in}(u)$. The following procedure is similar to the process given in lines 7–16 but replaces subscript *out* with *in*. Finally, an encrypted label $EL = \{EL_{out}, EL_{in}\}$ for graph $G$ will be constructed, and the *user* can outsource it to the *CSS* without risking privacy leakage.

The *TokenGen* in Algorithm 3 assumes the following scenario: the *user* wants to get the ACSD query result between $s$ and $t$ under the cost threshold $\theta$. First the *user* calculates $s$'s master keys $I_{out,s}$, $K_{out,s}$ of out-label and $t$'s master keys $I_{in,t}$, $K_{in,t}$ of in-label. Then it uses the public key $pk$ to encrypt the cost threshold $\theta$ in order to obtain encrypted result $\Theta$. Finally, it constructs a query token $T_{st}$ which consists of four master keys and the encrypted threshold. We use $T_{st}$ as the query token instead of using ($s$, $t$, $\theta$) because $T_{st}$ can both reduce the *CSS*'s computation costs and also obfuscate vertices' identity and cost threshold, so that *CSS* cannot get the vertices' identity of original graph $G$ and cost threshold by analyzing the *user*'s query token.

**Algorithm 3:** TokenGen.

```
Input: An ACSD query $q = (s, t, \theta)$, a secret key $K$ and a public key $pk$.
Output: A query token $T_{st}$.
begin
    calculate $I_{out,s} = h(K, s||0)$, $K_{out,s} = h(K, s||1)$.
    calculate $I_{in,t} = h(K, t||2)$, $K_{in,t} = h(K, t||3)$.
    compute $\Theta = Enc(pk, \theta)$.
    set $T_{st} = (I_{out,s}, K_{out,s}, I_{in,t}, K_{in,t}, \Theta)$.
    return $T_{st}$.
end
```

The *DistQuery* in Algorithm 4 works as follows. Upon receiving the *user*'s query token $T_{st}$, the *CSS* first parses $T_{st}$ as ($I_{out,s}$, $K_{out,s}$, $I_{in,t}$, $K_{in,t}$, $\Theta$) to get the master keys of both $s$'s out-label and $t$'s in-label, and encrypted cost threshold. Then the *CSS* uses the parsed token to search the entries of $EL_{out}$ associated with $s$ and $EL_{in}$ associated with $t$ as described in line 4–23.

**Algorithm 4:** DistQuery.

```
   Input: An encrypted label $EL$, a public key $pk$, and a query token $T_{st}$.
   Output: An encrypted ACSD query result $D_{st}$.
 1 begin
 2     parse $T_{st}$ as $(I_{out,s}, K_{out,s}, I_{in,t}, K_{in,t}, \Theta)$.
 3     init $L_s$, $L_t$, $Dist$.
 4     set counter $ctr = 0$.
 5     calculate $I_{out,su} = h(I_{out,s}, ctr)$.
 6     while $EL_{out}[I_{out,su}] \neq \perp$ do
 7         calculate $K_{out,su} = h(K_{out,s}, ctr)$.
 8         set $\Phi_{su} = EL_{out}[I_{out,su}] \oplus K_{out,su}$.
 9         parse $\Phi_{su}$ as $H_u||D_{su}||C_{su}$.
10         set $L_s[H_u] = (D_{su}, C_{su})$.
11         set $ctr = ctr + 1$.
12         calculate $I_{out,su} = h(I_{out,s}, ctr)$.
13     end
14     set counter $ctr = 0$.
15     calculate $I_{in,vt} = h(I_{in,t}, ctr)$.
16     while $EL_{in}[I_{in,vt}] \neq \perp$ do
17         calculate $K_{in,vt} = h(K_{in,t}, ctr)$.
18         set $\Phi_{vt} = EL_{in}[I_{in,vt}] \oplus K_{in,vt}$.
19         parse $\Phi_{vt}$ as $H_{in,vt}||D_{vt}||C_{vt}$.
20         set $L_t[H_v] = (D_{vt}, C_{vt})$.
21         set $ctr = ctr + 1$.
22         calculate $I_{in,vt} = h(I_{in,t}, ctr)$.
23     end
24     compute $Dist = CFA(L_s, L_t, \Theta)$.
25     find $\min\{Dist\}$ by performing $SMin$ protocol.
26     set $D_{st} = \min\{Dist\}$.
27     return $D_{st}$.
28 end
```

Let us describe a concrete procedure of searching $EL_{out}$ (in line 4–13) as an example. The searching on $EL_{in}$ is the same as what the *CSS* does on $EL_{out}$. First the *CSS* initializes a counter $ctr$ and calculates $I_{out,su}$ by calculating $h(I_{out,s}, ctr)$. Then it searches the entries in $EL_{out}$ whose index is $I_{out,su}$. If the search result is not *null*, the *CSS* calculates $K_{out,su}$ and executes an

XOR operation to restore the re-encrypted result. Once the result is restored, the *CSS* inserts the parsed entry ($D_{su}$, $C_{su}$) to $L_s$ with the pre-generated index $H_u$. Note that when parsing these encrypted entries, the *CSS* only knows the pseudo-identity of each vertex (e.g., $H_u$) instead of its real identity. After inserting the parsed result of this entry, the *CSS* increases the counter, calculates the index of next entry and repeats the procedure above until the search result is empty, which means that there are no unparsed entry associated with $u$ in the encrypted label $EL_{out}$.

When parsing of $EL_{out}$ and $EL_{in}$ is finished, the *CSS* will get the parsed result $L_s$ and $L_t$. Then it executes Constraint Filter Algorithm (CFA) shown in Algorithm 5 to get the filtered distance set *Dist*. Finally, both the *CSS* and the *CCSs* execute the *Secure Minimal Value* (*SMin*) protocol to find the encrypted minimum distance value $D_{st}$ as the output. When receiving $D_{st}$, the *user* uses the private key *sk* to decrypt $D_{st}$ and finally get the plain ACSD query result $d_{st}$.

**Algorithm 5:** Constraint filter algorithm (CFA).

```
Input: Encrypted parsed label $L_s$, $L_t$, and an encrypted cost threshold $\Theta$.
Output: An encrypted distance set $Dist$.
begin
    init $tmp$, $Dist$.
    foreach $L_s[H_i] \in L_s$, $L_t[H_j] \in L_t$ do
        if $H_i == H_j$ then
            compute $C_i = Add(C_{si} + C_{jt})$.
            compute $D_i = Add(D_{si} + D_{jt})$.
            set $tmp = tmp \cup \{(C_i, D_i)\}$.
        end
    end
    foreach $(C_i, D_i) \in tmp$ do
        compare $C_i$ and $\Theta$ by performing $SIC$ protocol.
        if $C_i \leq \Theta$ then
            set $Dist = Dist \cup \{D_i\}$.
        end
    end
    return $Dist$.
end
```

In order to get the filtered results which satisfy the cost constraint, we propose an algorithm to filter the parsed label. The *CFA* in Algorithm 5 works as follows. First the *CSS* traverses each entry in parsed labels $L_s$ and $L_t$. If there exists two indices $H_i$ in $L_s$ and $H_j$ in $L_t$ which satisfy $H_i == H_j$ (i.e., two vertices are the same), the *CSS* computes $D_i = Add(D_{si}, D_{jt})$ and $C_i = Add(C_{si}, C_{jt})$. Because we use additive homomorphic cryptosystem PCTD to encrypt distance and cost values, the *CSS* can execute the addition operation without *CCSs*' help. After homomorphic addition, the tuple ($C_i$, $D_i$) is joined to *tmp*.

For each ($C_i$, $D_i$) $\in$ *tmp*, both the *CSS* and *CCSs* execute the *Secure Integer Comparison* (*SIC*) protocol to compare the encrypted cost $C_i$ and its threshold $\Theta$. If $C_i$ is greater than $\Theta$, which means that this result does not satisfy the cost constraint, the tuple ($D_i$, $C_i$) will be filtered. Otherwise, it will be joined to *Dist*. Finally, the algorithm outputs *Dist* as the filtered result set.

## 6. Secure minimum value protocol for outsourced integers

We propose a secure integer comparison protocol *SIC* to compare two encrypted integers and a secure minimum value protocol *SMin* to find the minimum value among some encrypted integers. In PGAS, these two protocols are used to filter the cost constraint and find the shortest distance, and they can also be used in other relevant application scenarios. We introduce these two protocols in this section.

### *6.1. Background*

Assume that a set of encrypted integers is outsourced to a cloud server. For any subset $S = \{s_1, \cdots, s_m\}$, the user wants to query the minimum value $s_{\min}$. Then the server follows some process and returns the encrypted query result to the user. During such a process of finding the minimum value, the server cannot get the plaintexts of these integers. A simple and straightforward way to tackle this problem (which is usually the case in real scenarios) is to download the whole subset *S*. However, if *S* is a large set, the communication cost between the user and the server is not practical.

Finding the minimum value has to make comparisons between integers, hence a secure integer comparison method should be used to compare two encrypted integers and reveals nothing except their relationship (i.e., greater-than, less-than or equality), which can be seen as acceptable disclosures. A naive method is to use Order-Preserving Encryption (OPE) or Order-Revealing Encryption (ORE) to encrypt these integers and outsource them to the server. By utilizing these kinds of encryption algorithms, the server can directly compare the encrypted results just as what it does on plaintexts. Nevertheless, both OPE and ORE leak the information about plaintexts which causes some security issues. Naveed et al. [27] described several inference attacks which can recover nearly all the plaintexts encrypted by deterministic OPE schemes. Grubbs et al. [10] introduced leakage-abuse attacks which can recover 99% of first names of customer records from OPE/ORE-encrypted databases.

Hence, it is valuable to solve the problems of encrypted integer comparison and finding the minimum value from several encrypted integers without leaking any information about plaintexts.

### 6.2. Main idea

To find the minimum value of an encrypted integer set $S$, we use the naive algorithm as follows: First, the algorithm initializes a temporary variable $tmp = s_1$. Then, for each $s_i \in S \backslash \{s_1\}$, it sets $tmp = \min\{tmp, s_i\}$. Finally, it sets the minimum value $s_{\min} = tmp$.

The algorithm above needs to compare two encrypted integers. In Section 5, we utilize the additive homomorphism of PCTD to get the encrypted sum of two encrypted cost values. However, PCTD's encryption is probabilistic, hence we cannot judge the relationship without decrypting the two ciphertexts. To the best of our knowledge, none of these existing cryptosystems support both homomorphism and OPE/ORE properties. In addition, based on the security and privacy concerns mentioned above, both OPE and ORE cannot guarantee the security of outsourced data. Thus, we mainly focus on separating the storage and computation of outsourced data to achieve advanced privacy protection instead of using ORE or OPE schemes directly.

To separate data storage and computation, we use a cloud storage server and $n$ cloud computation servers as discussed in the system model. The *CSS* stores the encrypted integers that are outsourced. Both the *CSS* and the *CCSs* participate in comparing two encrypted integers and finding the minimum value of several encrypted integers. During the whole process, if the *CSS* and the *CCSs* do not collude, they only know the relationship of two integers. Unlike OPE/ORE, separating data storage and computation does not leak the information about plaintexts, which ensures advanced privacy protection.

Let $[x]$ be the encrypted integer of value $x$. In order to obtain the relationship between two encrypted integers $[x]$ and $[y]$ (the corresponding plaintexts are $x$ and $y$), we can calculate $[x-y]$ and judge whether $x-y$ is positive or not. $[x-y]$ can be computed as $[x] \cdot [-y]$ because of PCTD's additive homomorphism. If $x-y > 0$, it means $x > y$ and vice versa. Our proposed *SIC* protocol is based on this idea, and we enhance its privacy protection by choosing a random bit $b$ to decide whether the *CSS* computes $x-y$ or $y-x$, and add some random numbers to perturb the computation results. Hence, the *CSS* can get the comparison result without knowing the value of $x$, $y$ and $x-y$, and the *CCSs* cannot know which two integers are compared. This ensures that the result is not leaked and real values are protected.

In the following sub-sections, detailed information about our proposed *SIC* and *SMin* protocol and prove the correctness of them. Here, we use $k$ to represent the decryption threshold and $n$ to represent the number of *CCSs*. We also use $|m|$ to represent $m$'s bit length.

### 6.3. Secure integer comparison protocol

The *SIC* shown in Protocol 1 works as follows. Given two encrypted integers $[x]$ and $[y]$, the *SIC* protocol will compute an indicator $u^*$ which indicates the relationship between $x$ and $y$ (i.e., $x \geq y$ or $x < y$). If $u^* = 0$, it indicates $x \geq y$, otherwise, it indicates $x < y$.

**Protocol 1:** SIC protocol.

**Step 1**: The *CSS* first computes $[x'] = [x]^2 \cdot [1]$ and $[y'] = [y]^2$. Then it randomly chooses a bit $b \in \{0, 1\}$ and two random numbers $r_1, r_2$ which satisfy $|r_1| < |N|/4$ and $|r_2| < |N|/8$. If $b = 1$, the *CSS* computes $[l] = ([x'] \cdot [y']^{N-1})^{r_1} \cdot [r_2]$, otherwise, it computes $[l] = ([x']^{N-1} \cdot [y'])^{r_1} \cdot [r_2]$.

**Step 2**: For each $CCS_i$, it executes *PDec* to compute partially decrypted result $c_i$, and sends $c_i$ to $CCS_\gamma$ which is randomly designated by all *CCSs*.

**Step 3**: The $CCS_\gamma$ executes *TDec* to decrypt received $\{c_1, \cdots, c_n\}$ and gets decrypted result $l$. If $l$ satisfies $|l| > |N|/2$, then the $CCS_\gamma$ sets $u' = 1$, otherwise, it sets $u' = 0$. Then the $CCS_\gamma$ sends $u'$ to the *CSS*.

**Step 4**: The *CSS* computes $u^*$ as the indicator. If $b = 1$, it sets $u^* = u'$, otherwise, it sets $u^* = 1 - u'$. Finally, the *CSS* returns $u^*$ as the result.

Note that $CCS_\gamma$ can use one of symmetric encryption algorithms (e.g., AES) to encrypt $u'$ by $K_\gamma$ agreed by both $CCS_\gamma$ and the *CSS* before sending it to achieve confidentiality. Then, the *CSS* decrypts the encrypted $u'$ to get the result. We omit the detailed description of this for simplicity.

In **Step 1**, the ciphertext $[x']$ and $[y']$ can be represented as $[2x+1]$ and $[2y]$ respectively, and $[l]$ can also be represented as $[r_1(x'-y')+r_2]$ (when $b=1$) or $[r_1(y'-x')+r_2]$ (when $b=0$). We use $[x']$ and $[y']$ instead of using $[x]$ and $[y]$ in order to hide the equivalence relationship between $x$ and $y$. We also use two random numbers $r_1$, $r_2$ to perturb $[x'-y']$. If $x=y$, the $CCS_{\gamma}$ can get $l = 0$ by decrypting $[l]$ which means that $x = y$. If $x$ and $y$ are both integers, it is easy to prove that $2x + 1 = x' \neq y' = 2y$. Based on this, the following two equations are established:

$$x' < y' \Leftrightarrow 2x + 1 < 2y \Leftrightarrow x < y$$
$$x' > y' \Leftrightarrow 2x + 1 > 2y \Leftrightarrow x \geq y \tag{11}$$

Choosing a bit $b$ is an added method to hide the real relationship between $x$ and $y$ because the $CCS_{\gamma}$ cannot infer which is larger by decrypting $[l]$ without knowing the bit $b$.

In **Step 3**, if $|l| > |N|/2$, it means that $x' < y'$ when $b = 1$, and $x' > y'$ when $b = 0$, then the $CCS_{\gamma}$ sets $u' = 1$. In **Step 4**, the *CSS* computes $u^*$. If $b = 1$, $u^* = 1$, which indicates that $x < y$, and if $b = 0$, $u^* = 0$, which indicates that $x \geq y$.

On the other hand, if $|l| \leq |N|/2$ in **Step 3**, it means that $x' > y'$ when $b = 1$, and $x' < y'$ when $b = 0$, then the $CCS_{\gamma}$ sets $u' = 0$. In **Step 4**, the *CSS* computes $u^*$. If $b = 1$, $u^* = 0$, which indicates that $x \geq y$, and if $b = 0$, $u^* = 1$, which indicates that $x < y$.

The result shows that the *SIC* protocol is correct.

### 6.4. Secure minimal value protocol

The *SMin* shown in Protocol 2 works as follows. Given a set $S = \{s_1, \cdots, s_m\}$ which consists of $m$ encrypted integers, the *SMin* protocol will find the minimal value of them. Our proposed *SMin* protocol makes use of the aforementioned *SIC* protocol to compare two encrypted integers.

**Protocol 2:** SMin protocol.

**Step 1**: The *CSS* initializes $s_{\min} = s_1$.
**Step 2**: For each $s_i$ in $S \setminus \{s_1\}$, both the *CSS* and the *CCSs* perform $SIC(s_i, s_{\min})$ and the *CSS* gets the result $u^*$. If $u^* = 1$ (i.e., $s_i < s_{\min}$), then *CSS* sets $s_{\min} = s_i$.
**Step 3**: The *CSS* returns $s_{\min}$ as the result.

The *SMin* protocol is based on the naive minimum value algorithm which finds the minimum value by comparing against all of the integers in a candidate set. If the comparison result is correct, it can find the accurate minimum value successfully. Hence, the correctness of the *SMin* protocol is based on that of the *SIC* protocol. In the previous discussion, we have established that *SIC* protocol is correct, hence the *SMin* protocol is also correct.

## 7. Security analysis

In order to establish the security and privacy of PGAS, we present a detailed analysis in this section. First, we present formal definitions of leakage function, then we prove that PGAS is secure under the *CQA2-Security* model. We also show that our scheme can achieve advanced privacy protection compared with [30].

### 7.1. Leakage function

Here, we give precise definitions of two leakage functions: encryption leakage function $\mathcal{L}_{Enc}$ and query leakage function $\mathcal{L}_{Query}$. We also present leakage analysis to prove that PGAS leaks less information compared with Connor [30] and achieves advanced privacy protection.

#### 7.1.1. Encryption leakage $\mathcal{L}_{Enc}$

The encryption leakage function $\mathcal{L}_{Enc}$ of PGAS reveals the information that can be inferred by encrypted label *EL* stored on the *CSS*. This information includes the number of vertices in graph $G$ (noted as $n$), the number of entries in $G$'s label $L = \{L_{out}, L_{in}\}$ (noted as $|L_{out}|$ and $|L_{in}|$, respectively). Thus the leakage function $\mathcal{L}_{Enc} = (n, |L_{out}|, |L_{in}|)$.

#### 7.1.2. Query leakage $\mathcal{L}_{Query}$

This function consists of two types of leakages: the query pattern leakage reveals whether a query has appeared before, and the label pattern leakage reveals some information of label $L(v)$ which associates with a queried vertex $v$ and the common vertices between different labels.

**Definition 5** (Query Pattern Leakage)**.** Let $Q = \{q_1, \cdots, q_m\}$ be a non-empty sequence of queries, and for each $q_i \in Q$, the corresponding tuple is noted as $(s_i, t_i, \theta_i)$. For any two queries $q_i, q_j \in Q$, define $Sim(q_i, q_j) = (s_i = s_j, t_i = t_j, \theta_i = \theta_j)$, i.e., whether each of the nodes $q_i = (s_i, t_i, \theta_i)$ matches each of the nodes of $q_j = (s_j, t_j, \theta_j)$. The query pattern leakage function $\mathcal{L}_{QP}(Q) = S$ where $S$ is an $m \times m$ symmetric matrix, in which each entry $a_{ij}$ satisfies $a_{ij} = Sim(q_i, q_j)$. Note that we use pseudo-identities to hide the real identities of all vertices in $G$, thus $\mathcal{L}_{QP}(Q)$ cannot leak the real identity of the queried vertices.

**Definition 6** (Label Pattern Leakage)**.** Let $EL$ be an encrypted label of graph $G$, $q = (s, t, \theta)$ be a query on $EL$. The label pattern leakage function $\mathcal{L}_{LP}(EL, q) = (\Pi, \Omega)$. $\Pi$ is a pair $(X, Y)$ where $X = \{h(w) : (w, d, c) \in L_{out}(s)\}$ and $Y = \{h(w) : (w, d, c) \in L_{in}(t)\}$ are multi-sets and $h$: $\{0, 1\}^\lambda \times \{0, 1\}^* \to \{0, 1\}^\lambda$ is a secure pseudo-random function. $\Omega$ is a list of encrypted labels associated with queried vertices, i.e., let $\{v_1^*, \cdots, v_k^*\}$ be a set of vertices which has appeared in historical queries, $\Omega$ can be defined as $\Omega = \{EL(v_1^*), \cdots, EL(v_k^*)\}$, where $EL(v_i^*) = \{EL_{in}(v_i^*), EL_{out}(v_i^*)\}$ is the encrypted label associated with $v_i^*$. Note that $w$ represents a pseudo-identity so adversaries cannot get real identity of the corresponding vertex.

Based on the above definitions, the query leakage function can be defined as $\mathcal{L}_{Query} = (\mathcal{L}_{QP}(Q), \mathcal{L}_{LP}(EL, q))$.

*7.1.3. Leakage analysis*

Compared with Connor [30], our proposed scheme PGAS leaks less information. Connor's leakage also consists of *encryption leakage* (in Connor it is called *setup leakage*) and *query leakage*. Apart from this, Connor's *setup leakage* includes the maximum distance $D$, which appears in all the labels of graph $G$, i.e., $D = \max_{u \in V} \max_{\{(v, d_{uv}, c_{uv}) \in L_{out}, (v, d_{uv}, c_{uv}) \in L_{in}\}} d_{uv}$.

Note that $D$ is a plain distance instead of an encrypted distance. In PGAS, $D$ cannot be leaked, as we directly encrypt distance $d$ instead of encrypting $2^{N-d}$ (as in Connor), where $N = 2 * D + 1$.

Because Connor uses ORE to encrypt the cost values, the order information is also revealed. Thus, Connor's query leakage consists of *query pattern leakage, label pattern leakage* (*sketch pattern leakage*) and *cost pattern leakage*. Connor's *cost pattern leakage* leaks the order relationship: 1) for every two costs, and 2) between costs and the cost constraint during the query procedure. Note that for a non-empty sequence of queries $Q = \{q_1, \cdots, q_m\}$, if for each query $q_i \in Q$, there exists a unique cost threshold $\theta_i$, all the interval information of $\{\theta_1, \cdots, \theta_m\}$ will be revealed during the constant filtering process. In that case, adversaries can infer the range of cost values and thresholds, and if $m$ gets larger, the range will be smaller.

In PGAS, the *query leakage* does not include *cost pattern leakage* because we use PCTD to encrypt cost values which does not leak any order information.

According to the above leakage analysis, we can conclude that compared to Connor, PGAS leaks less information about the original plain graph $G$, which means that PGAS achieves better privacy protection.

*7.2. Security of PGAS*

In this section, we show that our proposed scheme PGAS achieves *CQA2-Security* by proving the following theorem:

**Theorem 1.** *If the cryptographic primitives g, h and PCTD are secure, then the proposed graph encryption scheme* $\Lambda$*=(KeyGen, GraphEnc, TokenGen, DistQuery) is* ($\mathcal{L}_{Enc}$, $\mathcal{L}_{Query}$)*-Secure against the adaptive chosen-query attack.*

**Proof.** The basis of this proof is to construct a simulator $\mathcal{S}$. Given the leakage functions $\mathcal{L}_{Enc}$ and $\mathcal{L}_{Query}$, $\mathcal{S}$ simulates a dummy encrypted label $EL^* = \{EL_{out}^*, EL_{in}^*\}$ and a dummy list of token $\mathcal{T}^*$. For all Probabilistic Polynomial Time (PPT) adversaries $\mathcal{A}$, if they cannot distinguish between the two experiments *Real* and *Ideal*, we can say that the graph encryption scheme PGAS is ($\mathcal{L}_{Enc}$, $\mathcal{L}_{Query}$)-Secure against the adaptive chosen-query attack.

*Simulating $EL^*$*: Given $\mathcal{L}_{Enc}$, $\mathcal{S}$ generates a dummy $EL_{in}^*$ based on graph $G$'s out-label $L_{out}$. First $\mathcal{S}$ generates a dummy secret key $K^* \leftarrow \{0, 1\}^\lambda$. Assume the vertex set $V = \{v_1, \cdots, v_n\}$, for each $v_i$, $\mathcal{S}$ randomly chooses $w_i$ which satisfies $\sum_1^n w_i = |L_{out}|$. It then uniformly samples unique $\eta_i \leftarrow \{0, 1\}^\lambda$ and $\rho_i \leftarrow \{0, 1\}^\lambda$, where $\lambda$ is a security parameter. For all $0 \le j \le w_i$, $\mathcal{S}$ performs the following steps to simulate each dummy encrypted entries in $EL_{out}^*(v_i)$:

1) $\mathcal{S}$ calculates $\eta_{ij} = h(\eta_i, j)$ and $\rho_{ij} = g(\rho_i, j)$, where $h$ and $g$ are two hash functions given in Eq. (10). 2) Assuming that the $j^{th}$ vertex which appeared in $L_{out}(v_i)$ is $u$, $\mathcal{S}$ calculates $H_u^* = h(K^*, u)$. 3) $\mathcal{S}$ randomly generates two non-negative integers $d_{ij}^*$ and $c_{ij}^*$ as dummy distance and cost values, then uses PCTD to encrypt them, and get the corresponding ciphertexts $D_{ij}^*$ and $C_{ij}^*$. 4) $\mathcal{S}$ sets $\Phi_{ij}^* = H_u^* || D_{ij}^* || C_{ij}^*$ and then sets $EL_{out}^*[\eta_{ij}] = \rho_{ij} \oplus \Phi_{ij}^*$. Similarly, $\mathcal{S}$ generates dummy $EL_{in}^*$ and finally gets the dummy encrypted label $EL^* = \{EL_{out}^*, EL_{in}^*\}$.

*Simulating $\mathcal{T}^*$*: Assume the query sequence made by adversary $\mathcal{A}$ is $Q = \{q_1, \cdots, q_n\}$, then the corresponding query token lists can be noted as $\mathcal{T} = \{T_1, \cdots, T_n\}$. For each $q_i = (s_i, t_i, \theta_i) \in Q$, given $\mathcal{L}_{Query}$, $\mathcal{S}$ simulates the dummy query token $T_i^*$ as follows: $\mathcal{S}$ first find whether $s_i$ or $t_i$ has appeared in the previous queries. For vertex $s_i$, if $s_i$ has appeared, $\mathcal{S}$ sets $I_{out,s_i}$ and $K_{out,s_i}$ as the previous value. Otherwise, $\mathcal{S}$ sets $I_{out,s_i} = \eta_i$ and $K_{out,s_i} = \rho_i$ where $\eta_i$ and $\rho_i$ have never been used before, and stores the relationship between $s_i$, $\eta_i$ and $\rho_i$. For vertex $t_i$, the operations is similar to what $\mathcal{S}$ does for $s_i$. Following this, when generating dummy encrypted cost threshold $\Theta_i^*$ for $q_i$, $\mathcal{S}$ uses PCTD to encrypt a positive integer $\theta_i^*$ which is chosen randomly. Finally, $T_i^*$ is simulated as an element of $\mathcal{T}^*$, in the end, $\mathcal{T}$ can be simulated by $\mathcal{T}^*$ in polynomial time because the query sequence is polynomial.

If the cryptographic primitives $g$, $h$ and PCTD are secure, the dummy encrypted label $EL^*$ and the query token list $\mathcal{T}^*$ are indistinguishable from the real ones. Therefore, the distinction between *Real* and *Ideal* experiments is not possible for all PPT adverseries $\mathcal{A}$. Thus, we have

$$|\Pr[Real_{\mathcal{C},\mathcal{A}}(\lambda) = 1] - \Pr[Ideal_{\mathcal{C},\mathcal{A},\mathcal{S}}(\lambda) = 1]| \le negl(\lambda) \tag{12}$$

where $negl(\lambda)$ is a negligible function. □

**Table 2**
Properties of different graph encryption schemes.

| | GRECS | Connor | PGAS |
|---|---|---|---|
| Graph Type | undirected | directed | directed |
| CSD Queries | unsupported | supported | supported |
| Accuracy | approximate | approximate | accurate |

**Table 3**
Time, space and communication complexity.

| Complexity Type | Algorithm | GRECS | Connor | PGAS |
|---|---|---|---|---|
| Time Complexity | *GraphEnc* | $\mathcal{O}(m)$ | $\mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| | *TokenGen* | $\mathcal{O}(1)$ | $\mathcal{O}(2^{d_\theta})$ | $\mathcal{O}(1)$ |
| | *DistQuery* | $\mathcal{O}(n)$ | $\mathcal{O}(nd_\theta)$ | $\mathcal{O}(n)$ |
| Space Complexity | *GraphEnc* | $\mathcal{O}(m)$ | $\mathcal{O}(m)$ | $\mathcal{O}(m)$ |
| | *TokenGen* | $\mathcal{O}(1)$ | $\mathcal{O}(2^{d_\theta})$ | $\mathcal{O}(1)$ |
| Communication Complexity | *TokenGen* | $\mathcal{O}(1)$ | $\mathcal{O}(2^{d_\theta})$ | $\mathcal{O}(1)$ |
| | *DistQuery* | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ |

## 8. Performance analysis and experimental evaluation

The complete evaluation of PGAS has been done in two parts. We first make a theoretical analysis, followed by quantitative evaluation. We make the comparisons between PGAS and two previously proposed graph encryption scheme: GraphEnc$_3$ in GRECS [24] (we use the term GRECS in the following parts for simplicity) and Connor [30]. Note that we use 2HCL-based GRECS instead of original GRECS in order to eliminate the influence of different labels in the same graphs. A brief comparison of these schemes given in Table 2.

As is shown in Table 2, GRECS can only encrypt undirected graphs and does not support CSD queries. Both Connor and PGAS support encryption and CSD queries on direct graphs. In addition, PGAS can return accurate query results to the *user* instead of approximate query results as in GRECS and Connor.

The following analysis and experimental evaluations show that our scheme achieves 100% accuracy with acceptable efficiency. We also make numerous experiments on real-world datasets to prove the practical efficiency and accuracy of PGAS.

### 8.1. Complexity analysis

The important factors which affect the efficiency of a graph encryption scheme are the time and space complexity of algorithm *GraphEnc, TokenGen* and *DistQuery*. Hence, we first present their concrete analysis, followed by analyzing the communication complexity between the *user* and the *CSS*.

In all the three schemes, the *GraphEnc* algorithm encrypts all the entries in a graph *G*'s label. If $m$ represents the number of entries in label $L$, it is easy to calculate both time and space complexity of all three schemes to be $\mathcal{O}(m)$.

In the *TokenGen* algorithm, GRECS uses two pseudo-identities generated by Pseudo-Random Function (PRF) as a query token, and PGAS's query token consists of four hash values and an encrypted cost threshold, so both their time and space complexity is $\mathcal{O}(1)$. However, in Connor, the query token consists of four hash values and a ciphertext-based binary tree whose depth is $d_\theta$. Hence, the number of nodes in this tree is $2^{d_\theta}-1$ and the time and space complexity is $\mathcal{O}(2^{d_\theta})$.

In the *DistQuery* algorithm, there are four steps (three in GRECS) to return the encrypted query result. The first two steps are searching & parsing the encrypted label and get the parsed result sets (i.e., $L_s$ and $L_t$ associated with $s$ and $t$). The third step is filtering the results which satisfy the cost constraint. This step is not included in GRECS because it cannot support CSD queries. The last step is calculating approximate shortest distance or finding the accurate shortest distance. Let $n$ represents the maximum size of label $L(v)$ for each $v \in V$, where $L(v) = L_{in}(v) \cup L_{out}(v)$ in GRECS and $L(v) = \{L_{in}(v), L_{out}(v)\}$ in Connor and PGAS. In each step, for both GRECS and PGAS, the time complexity is $\mathcal{O}(n)$. In Connor, the third step needs to execute the *Tree-based Ciphertexts Comparison Algorithm* (*TCCA*) at most $n$ times, and each execution needs to compare $d_\theta$ ciphertexts, so the time complexity of Connor is $\mathcal{O}(nd_\theta)$. The time complexity of other steps in Connor is all $\mathcal{O}(n)$. Therefore, the total time complexity of GRECS, Connor and PGAS are $\mathcal{O}(n)$, $\mathcal{O}(nd_\theta)$ and $\mathcal{O}(n)$, respectively.

Next we discuss the communication complexity. In a query process, the *user* sends a query token to the *CSS*, and the *CSS* returns the encrypted result. So the total communication complexity is the combination of *TokenGen*'s space complexity and the returned result's space complexity. Because all the three schemes return single encrypted distance, as a result, hence the space complexity for them is $\mathcal{O}(1)$. Therefore, the communication complexity of GRECS, Connor and PGAS are $\mathcal{O}(1)$, $\mathcal{O}(2^{d_\theta})$ and $\mathcal{O}(1)$, respectively.

Table 3 lists the time, space and communication complexity of all three schemes. It is evident that PGAS achieves the same complexity of GRECS, and the overall complexity is lower than that of Connor.

### 8.2. Accuracy analysis

In 2HCL-based shortest distance query (e.g., [1]), the calculation of shortest distance $d_{st}$ uses the following equation:

$$d(s,t) = \min\{d_{sv} + d_{vt} | (v, d_{sv}) \in L(s), (v, d_{vt}) \in L(t)\} \tag{13}$$

This is used to make a shortest distance query in a undirected graph. In directed graph, $L(s)$ is replaced by $L_{out}(s)$ and $L(t)$ is replaced by $L_{in}(t)$.

#### 8.2.1. Accuracy of GRECS

In GRECS, it is impossible to find the minimum value of encrypted distances because it uses the BGN cryptosystem [3] which supports limited additive and only one multiplicative homomorphic operations. Unfortunately it does not support comparison operation between two ciphertexts. So authors used an alternative method to calculate the approximate shortest distance by performing some add operations and one multiply operation [24].

We assume the following scenario: a user wants to query the shortest distance between $s$ and $t$. Let $I$ be the set of nodes common between $L(s)$ and $L(t)$, and let $d(s,t) = \min\{d_{sv} + d_{vt} | (v, d_{sv}) \in L(s), (v, d_{vt}) \in L(t)\}$. The larger scale of a graph means larger $I$. The calculated result $d_{st}$ satisfies the following in-equation:

$$d(s,t) - \log|I| \le d_{st} \le \alpha \cdot d(s,t) \tag{14}$$

where $\alpha(\alpha \ge 1)$ is the approximation factor. In-Eq. (14) gives the upper and lower bounds of the calculated approximate distance. The complete proof of the error bound is in [24].

In this in-equation, we can easily find three conclusions that indicate the defect of GRECS: 1) If $|I| > 1$, the relative error between approximate value $d_{st}$ and accurate value $d(s, t)$ must exist. 2) If $|I|$ is too large or $d(s, t)$ is too small, the relative error can be very large. 3) If $|I| \gg d(s, t)$, the lower bound $d(s,t) - \log|I|$ could be negative which means that the approximate result $d_{st}$ might be negative.

According to the analysis mentioned above, the accuracy of GRECS is low especially in large-scale graphs. Our experiments have proven this conclusion which will be described later.

#### 8.2.2. Accuracy of connor

Connor uses the same method to calculate approximate shortest distances, so the upper and lower bounds of absolute error are the same as that in GRECS. Connor uses *TCCA* to filter the results, the cost of which does not exceed the cost threshold. However, *TCCA* returns *uncertainty* which means that it cannot determine the relationship between the cost and the cost threshold. Let $d_\theta$ be the depth of ciphertext cost tree, the probability of *uncertainty* is $(\frac{1}{2})^{d_\theta}$ [30]. In Connor, if *TCCA* returns *uncertainty*, the pair $(D_{sv}, D_{vt})$ is also added as one of the filtered results even if it does not satisfy the cost constraint actually, which decreases the accuracy of CSD query results.

#### 8.2.3. Accuracy of PGAS

If the 2HCL $L$ of a graph $G$ supports plain ACSD queries, our proposed scheme PGAS can achieve 100% accuracy. PGAS uses the *Constraint Filter Algorithm (CFA)* to filter the result. In *CFA*, the filtering method is to compare the cost and its threshold which are both encrypted integers. The accurate comparison result can be obtained by performing the *SIC* protocol which makes the constraint filter accurate. According to Eq. (13), the ACSD query result can be obtained by performing the *SMin* protocol which finds the encrypted minimum value from filtered distance set *Dist*. In general, the 100% accuracy of the query results of PGAS is guaranteed by the correctness of *SIC* and *SMin* protocol.

### 8.3. Experimental evaluation

The quantitative evaluation uses real-world datasets to measure the efficiency and accuracy of GRECS, Connor, and PGAS. These evaluation results prove the analysis presented earlier.

#### 8.3.1. Dataset

We use public real-world graph datasets available on Stanford SNAP website.[1] We choose four different kinds of datasets: *as-skitter*, a large Internet topology graph; *ego-facebook*, a social circles from Facebook; *slashdot*, a Slashdot social network from February 2009; and *email-enron*, an email communication network from Enron. Table 4 shows different properties of these datasets.

Note that some graphs contain thousands of or even millions of vertices and edges. Due to the limited computational resources, we randomly choose subgraphs of these graphs. For each graph, we choose nine subgraphs whose number of vertices range from 100 to 900. Since these graphs are unweighted, we randomly add the cost for each edge which follows a uniform distribution between 1 and 10. Because GRECS does not support the directed graph, we convert some directed graphs to undirected graphs by adding edge $(v, u)$ if there exists an edge $(u, v)$ in the original graphs. The converted graphs are only used in the evaluation of GRECS, and in Connor and PGAS, we use the original directed graph.

[1] http://snap.stanford.edu/data/.

**Table 4**
Characteristics of graph datasets used for evaluation.

| Dataset | Vertices | Edges | Storage |
|---|---|---|---|
| as-skitter | 1,696,415 | 11,095,298 | 142.20MB |
| ego-facebook | 4039 | 88,234 | 0.81MB |
| slashdot | 82,168 | 438,643 | 4.93MB |
| email-enron | 36,692 | 367,662 | 1.84MB |

**Table 5**
Encryption and token generation time.

| | GRECS | Connor | PGAS |
|---|---|---|---|
| Encryption time for each entry | 0.62 ms | 4.49 ms | 1.42 ms |
| Token generation time | 0.04 ms | 63.74 ms | 0.72 ms |

### *8.3.2. Experimental setup*

We implement the 2HCL generation algorithm in [35] and [1] to generate the label of each graph. We also implement GRECS, Connor, and PGAS to compare their efficiency and accuracy. All programs are written in Java and compiled with JDK 1.8 x64 environment.

We use the JPBC library[2] to implement the BGN cryptosystem [3] used in GRECS and Connor. We also use the ORE scheme[3] proposed by Lewi et al. [17] to encrypt the costs and generate ciphertext cost tree in Connor. In PGAS, we use *BigInteger* class in Java to implement the PCTD cryptosystem and both *SIC* and *SMin* protocol. We deploy two *CCSs* to undertake the outsourced computation tasks, set the security parameter $\lambda = 256$, the depth of ciphertext cost tree $d_\theta = 4$, and use HMAC-SHA256 to instantiate all hash functions.

For each scheme, we randomly choose 500 query requests per graph and calculate the average of each result. We also compare the query results between the plain query and the query based on the encrypted graph, which indicates the accuracy for each scheme.

All experiments are run on a system with 8 GB RAM and Intel i7-4790 CPU running 64-bit Windows 10 operating system.

### *8.3.3. Performance of graphenc*

We evaluate the *GraphEnc* algorithm for all the three graph encryption schemes: GRECS, Connor, and PGAS. Because the time complexity of *GraphEnc* is $\mathcal{O}(m)$, the total encryption time for a graph is determined by the encryption time for each entry in graph *G*'s label *L*.

As is shown in Table 5, the encryption time per entry in GRECS is less than 1*ms*, and the encryption time of PGAS is about 1.4*ms*. Because PGAS encrypts both distance and cost values, and GRECS only encrypts the distance, the encryption time in PGAS is about two times that of GRECS. In Connor, the encryption time is significantly higher than that of other schemes because it uses ORE to encrypt the cost, where encryption time is higher than that of BGN and PCTD cryptosystem.

Note that the *GraphEnc* algorithm in PGAS is highly-parallelizable, hence the encryption time can be further improved by using a cluster. Assume that the encryption time is $t$ hours if we use one computer to encrypt a graph *G*, in a clustered environment that consists of $n$ computers, the encryption time will be decreased to $t/n$ hours.

Based on the analysis above, we can deduce that the efficiency of the *GraphEnc* algorithm is acceptable in practical scenarios.

### *8.3.4. Performance of TokenGen*

We evaluate the *TokenGen* algorithm of three aforementioned schemes. Table 5 shows the average token generation time for these three schemes. In Connor, the query token includes a ciphertext cost tree whose depth is $d_\theta$, so $2^{d_\theta}$ vertices are encrypted to build such a tree, which brings significant computational costs when generating tokens. Hence, Connor's token generation time is significantly higher than that of other schemes.

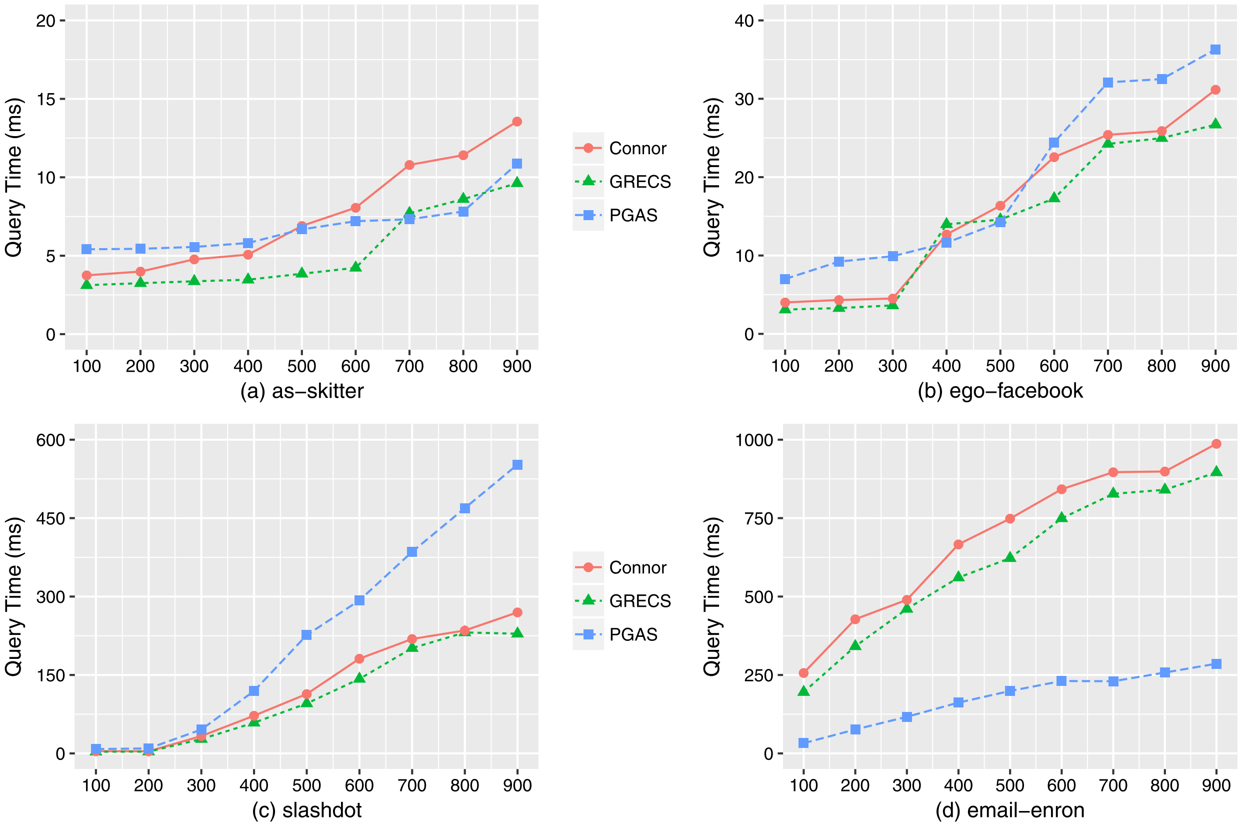
As for GRECS and PGAS, the latter uses asymmetric encryption, so its token generation time is higher than that of the former which only calculates two HMACs. However, PGAS supports ACSD queries that GRECS cannot provide, and its token generation time is not high. Therefore, we can say that the efficiency of the *TokenGen* algorithm is acceptable.

### *8.3.5. Performance of DistQuery*

We evaluate the *DistQuery* algorithm of three aforementioned schemes. As can be seen in Fig. 4, larger scale graphs lead to longer query times. The result proves the complexity analysis given earlier.

In *as-skitter, ego-facebook* and *slashdot*, the average query time of GRECS and Connor is similar to or smaller than that of PGAS. This is reasonable because PGAS performs the *SIC* and *SMin* protocol between the *CSS* and *CCSs* to find the accurate

[2] http://gas.dia.unisa.it/projects/jpbc/.
[3] https://crypto.stanford.edu/ore/.



(a) as–skitter

(b) ego–facebook

(c) slashdot

(d) email–enron

**Fig. 4.** Query time.

result, whereas, in GRECS and Connor, only the cloud server participates in the process of computing the approximate result. In PGAS, we slightly sacrifice the query efficiency in order to guarantee the security and accuracy, this scarification is tolerable because we achieve 100% accuracy and the efficiency loss is not significant in practical scenarios.

However, in *email-enron*, PGAS's average query time is significantly lower than the other two schemes. GRECS and Connor use BGN cryptosystem to encrypt $2^{N-d_{st}}$, where $d_{st}$ is an entry which appears in $s$'s label and $N = 2 * d_{\max} + 1$ is related to the maximum distances $d_{\max}$ which appears in the label $L$. In BGN's decryption operation, it needs to calculate the discrete logarithm to get the decrypted result. In this scenario, if we use brute-force to calculate the discrete logarithm, the time complexity can reach $\mathcal{O}(2^{2N})$. Even if we use the Pollard method, the time complexity of decryption can only decrease to $\mathcal{O}(2^{N})$. In general, the time cost of BGN's decryption is so large that it has been the bottleneck of the *DistQuery* algorithm in GRECS and Connor.

In PGAS, it uses PCTD to encrypt and decrypt distances in $L$ instead of an exponent of these distances, so the message space is relatively small. In addition, PCTD does not need to calculate a discrete logarithm when decrypting the ciphertexts. Therefore, in some of the graphs (e.g., *email-enron*), the efficiency of GRECS and Connor is significantly lower than that of PGAS. Note that, the high efficiency of PGAS in *email-enron* mainly based on the bottleneck of BGN when decrypting large plaintexts. If $N$ is small or a more efficient homomorphic encryption scheme is developed, it is uncertain that PGAS can still be more efficient than GRECS and Connor.

In this evaluation, we can say that the query efficiency of PGAS is acceptable. However, in some kinds of graphs, the efficiency is higher than the other two schemes which proves that our scheme can be more effective when querying on such graphs.

### *8.3.6. Query accuracy*

It can be observed from Fig. 5, that the query accuracy of GRECS and Connor is similar because they use the same method to calculate the approximate shortest distance. We can also conclude that as the scale of the graph grows, the accuracy of them becomes lower. If the graph scale is large, this accuracy is intolerable because most of these query results become incorrect. Even if the scale of graphs is the same, the query accuracy can be significantly different, i.e., when querying on *as-skitter*, the accuracy can be 95%–100%, however, in *email-enron*, the accuracy is below 25% even if the vertex number of its subgraph is 900. Hence, we can conclude that the topology structure can also influence the query accuracy in GRECS and Connor.

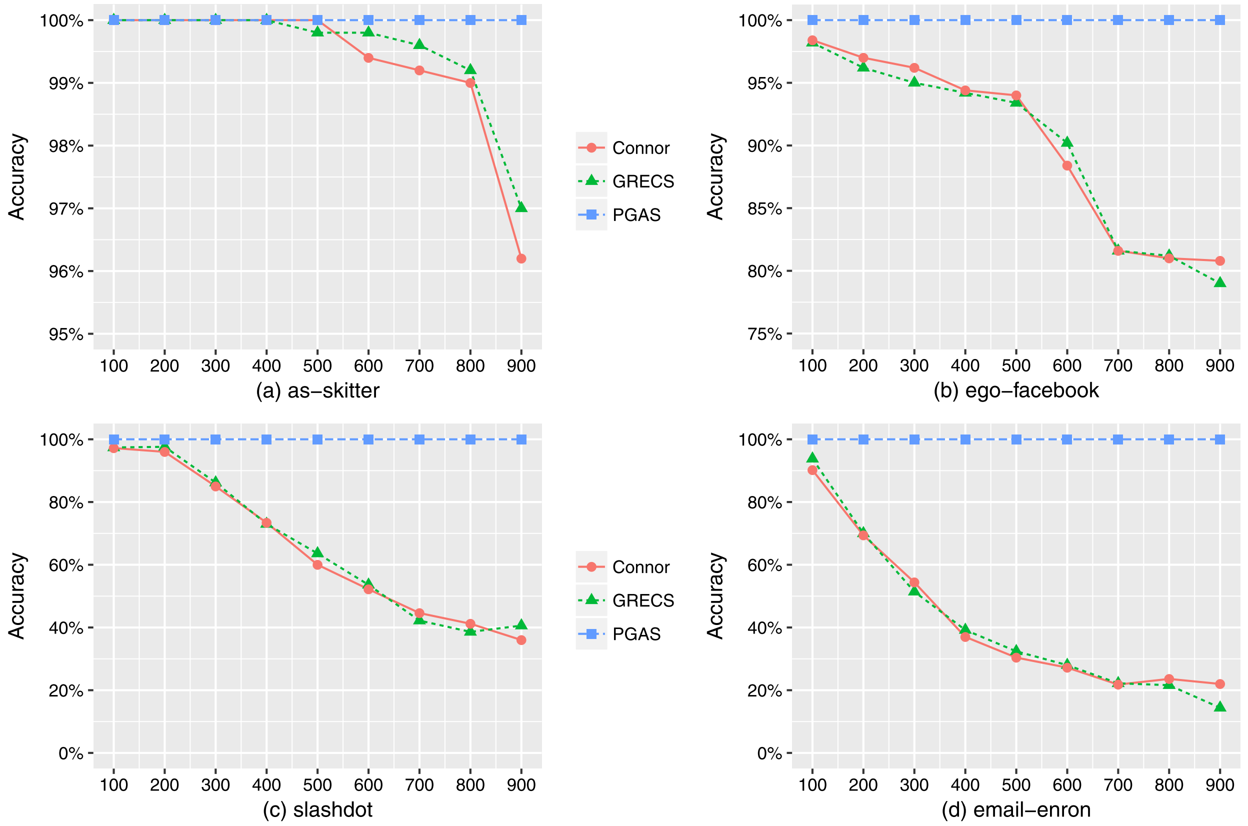


**Fig. 5.** Query accuracy.

In PGAS, the query accuracy is always 100%, which means that PGAS can return accurate query results, independent of type & size of the graph. In general, PGAS has significant advantages on query accuracy when comparing with GRECS and Connor.

## 9. Conclusion

In this work, we proposed PGAS, an advanced privacy-preserving graph encryption scheme that supports ACSD queries. Based on a novel system architecture that separates the data storage and computation, both the *CSS* and *CCSs* cannot get any sensitive information. PGAS also leverages 2HCL and secure cryptographic primitives to hide the structural information of graph data, and guarantee acceptable query efficiency simultaneously. In addition, we proposed a secure integer comparison protocol *SIC* to compare two encrypted integers and a secure minimum value protocol *SMin* to find the minimal value of several encrypted integers. Security analysis shows that PGAS achieves *CQA-2 Security* with less leakage compared with the existing schemes, which provides advanced privacy protection. The evaluation results show that PGAS has acceptable efficiency and achieves 100% accuracy.

Compared with existing schemes, to achieve 100% accuracy, PGAS needs to execute more cyphertext-based comparison operations between the CSS and the CCSs. Besides, dynamic index update and multiple clouds are not considered to enhance the scalability of PGAS. Hence, future work may focus on constructing more efficient graph encryption schemes that support dynamic index update and multiple data sources. For instance, multiple clouds can be introduced to alleviate both computational and storage burden on the centralized cloud server, while storage of a single large-scale graph on multiple clouds without sacrificing the querying accuracy can be considered as an interesting research direction.

## Declaration of Competing Interest

None.

## Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant Nos. 61402037, 61272512, 61702105, U1804263), and grant No. 2019CX10014 from the Graduate Technological Innovation Project of Beijing Institute of Technology.

## References

[1] T. Akiba, Y. Iwata, Y. Yoshida, Fast exact shortest-path distance queries on large networks by pruned landmark labeling, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22–27, 2013, 2013, pp. 349–360.

[2] C. Bode, S. Irnich, The shortest-path problem with resource constraints with (k, 2)-loop elimination and its application to the capacitated arc-routing problem, Eur. J. Oper. Res. 238 (2) (2014) 415–426.

[3] D. Boneh, E. Goh, K. Nissim, Evaluating 2-DNF formulas on ciphertexts, in: Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10–12, 2005, Proceedings, 2005, pp. 325–341.

[4] N. Cao, Z. Yang, C. Wang, K. Ren, W. Lou, Privacy-preserving query over encrypted graph-structured data in cloud computing, in: 2011 International Conference on Distributed Computing Systems, ICDCS 2011, Minneapolis, Minnesota, USA, June 20–24, 2011, 2011, pp. 393–402.

[5] M. Chase, S. Kamara, Structured encryption and controlled disclosure, in: Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5–9, 2010. Proceedings, 2010, pp. 577–594.

[6] R. Cramer, V. Shoup, Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption, in: Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28–May 2, 2002, Proceedings, 2002, pp. 45–64.

[7] R. Curtmola, J.A. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions, J. Comput. Secur. 19 (5) (2011) 895–934.

[8] C. Gentry, Fully homomorphic encryption using ideal lattices, in: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31–June 2, 2009, 2009, pp. 169–178.

[9] C. Gentry, A. Sahai, B. Waters, Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based, in: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2013. Proceedings, Part I, 2013, pp. 75–92.

[10] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, T. Ristenpart, Leakage-abuse attacks against order-revealing encryption, in: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22–26, 2017, 2017, pp. 655–672.

[11] P. Hansen, Bicriterion path problems, in: G. Fandel, T. Gal (Eds.), Multiple Criteria Decision Making Theory and Application, Springer Berlin Heidelberg, Berlin, Heidelberg, 1980, pp. 109–127.

[12] R. Hassin, Approximation schemes for the restricted shortest path problem, Math. Oper. Res. 17 (1) (1992) 36–42.

[13] X. Huang, R. Yu, J. Kang, N. Wang, S. Maharjan, Y. Zhang, Software defined networking with pseudonym systems for secure vehicular clouds, IEEE Access 4 (2016) 3522–3534.

[14] J. Kang, R. Yu, X. Huang, S. Maharjan, Y. Zhang, On-demand pseudonym systems in geo-distributed mobile cloud computing, in: 3rd IEEE International Conference on Cyber Security and Cloud Computing, CSCloud 2016, Beijing, China, June 25–27, 2016, 2016, pp. 136–141.

[15] M. Keller, P. Scholl, Efficient, oblivious data structures for MPC, in: Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014, Proceedings, Part II, 2014, pp. 506–525.

[16] S. Ko, W. Han, Turbograph++: A scalable and fast graph analytics system, in: Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10–15, 2018, 2018, pp. 395–410.

[17] K. Lewi, D.J. Wu, Order-revealing encryption: new constructions, applications, and lower bounds, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24–28, 2016, 2016, pp. 1167–1178.

[18] J. Li, D. Lin, A.C. Squicciarini, J. Li, C. Jia, Towards privacy-preserving storage and retrieval in multiple clouds, IEEE Trans. Cloud Comput. 5 (3) (2017) 499–509.

[19] T. Li, Z. Liu, C. Jia, Z. Fu, J. Li, Key-aggregate searchable encryption under multi-owner setting for group data sharing in the cloud, IJWGS 14 (1) (2018) 21–43.

[20] X. Liu, K.R. Choo, R.H. Deng, R. Lu, J. Weng, Efficient and privacy-preserving outsourced calculation of rational numbers, IEEE Trans. Dependable Sec. Comput. 15 (1) (2018) 27–39.

[21] X. Liu, R.H. Deng, K.R. Choo, J. Weng, An efficient privacy-preserving outsourced calculation toolkit with multiple keys, IEEE Trans. Inf. Forensics Secur. 11 (11) (2016) 2401–2414.

[22] Z. Liu, T. Li, P. Li, C. Jia, J. Li, Verifiable searchable encryption with aggregate keys for data sharing system, Future Gener. Comput. Syst. 78 (2018) 778–788.

[23] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, J.M. Hellerstein, Graphlab: a new framework for parallel machine learning, in: UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8–11, 2010, 2010, pp. 340–349.

[24] X. Meng, S. Kamara, K. Nissim, G. Kollios, GRECS: graph encryption for approximate shortest distance queries, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12–16, 2015, 2015, pp. 504–517.

[25] Y. Miao, J. Weng, X. Liu, K.R. Choo, Z. Liu, H. Li, Enabling verifiable multiple keywords search over encrypted cloud data, Inf. Sci. 465 (2018) 21–37, doi:10.1016/j.ins.2018.06.066.

[26] K. Mouratidis, M.L. Yiu, Shortest path computation with no information leakage, PVLDB 5 (8) (2012) 692–703.

[27] M. Naveed, S. Kamara, C.V. Wright, Inference attacks on property-preserving encrypted databases, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12–16, 2015, 2015, pp. 644–655.

[28] Z. Qin, T. Yu, Y. Yang, I. Khalil, X. Xiao, K. Ren, Generating synthetic decentralized social graphs with local differential privacy, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30–November 03, 2017, 2017, pp. 425–438.

[29] S. Sharma, J. Powers, K. Chen, Privategraph: privacy-preserving spectral analysis of encrypted graphs in the cloud, IEEE Trans. Knowl. Data Eng. 31 (5) (2019) 981–995, doi:10.1109/TKDE.2018.2847662.

[30] M. Shen, B. Ma, L. Zhu, R. Mijumbi, X. Du, J. Hu, Cloud-based approximate constrained shortest distance queries over encrypted graphs with privacy protection, IEEE Trans. Inf. Forensics Secur. 13 (4) (2018) 940–953.

[31] D.X. Song, D.A. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: 2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14–17, 2000, 2000, pp. 44–55.

[32] S. Storandt, Route planning for bicycles - exact constrained shortest paths made practical via contraction hierarchy, in: Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012, 2012.

[33] G. Tsaggouris, C.D. Zaroliagis, Multiobjective optimization: improved FPTAS for shortest paths and non-linear objectives with applications, Theory Comput. Syst. 45 (1) (2009) 162–186.

[34] M. van Dijk, C. Gentry, S. Halevi, V. Vaikuntanathan, Fully homomorphic encryption over the integers, in: Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings, 2010, pp. 24–43.

[35] S. Wang, X. Xiao, Y. Yang, W. Lin, Effective indexing for approximate constrained shortest path queries on large road networks, PVLDB 10 (2) (2016) 61–72.

[36] X. Wang, W. Chen, J. Chou, C. Bryan, H. Guan, W. Chen, R. Pan, K. Ma, Graphprotector: a visual interface for employing and assessing multiple privacy preserving graph algorithms, IEEE Trans. Vis. Comput. Graph. 25 (1) (2019) 193–203, doi:10.1109/TVCG.2018.2865021.

[37] X.S. Wang, K. Nayak, C. Liu, T.H. Chan, E. Shi, E. Stefanov, Y. Huang, Oblivious data structures, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3–7, 2014, 2014, pp. 215–226.

[38] K. Xue, J. Hong, Y. Ma, D.S.L. Wei, P. Hong, N. Yu, Fog-aided verifiable privacy preserving access control for latency-sensitive data sharing in vehicular cloud computing, IEEE Netw. 32 (3) (2018) 7–13.

[39] K. Xue, S. Li, J. Hong, Y. Xue, N. Yu, P. Hong, Two-cloud secure database for numeric-related SQL range queries with privacy preserving, IEEE Trans. Inf. Forensics Secur. 12 (7) (2017) 1596–1608.

[40] Y. Yang, Z. Li, X. Wang, Q. Hu, Finding the shortest path with vertex constraint over large graphs, Complexity 2019 (2019) 8728245:1–8728245:13, doi:10.1155/2019/8728245.

[41] Y. Yang, X. Liu, R.H. Deng, Expressive query over outsourced encrypted data, Inf. Sci. 442–443 (2018) 33–53, doi:10.1016/j.ins.2018.02.017.

[42] C. Zhang, L. Zhu, C. Xu, PTBI: an efficient privacy-preserving biometric identification based on perturbed term in the cloud, Inf. Sci. 409 (2017) 56–67, doi:10.1016/j.ins.2017.05.006.

[43] C. Zhang, L. Zhu, C. Xu, K. Sharif, X. Liu, PPTDS: a privacy-preserving truth discovery scheme in crowd sensing systems, Inf. Sci. 484 (2019) 183–196, doi:10.1016/j.ins.2019.01.068.

[44] Y. Zhang, X. Chen, J. Li, D.S. Wong, H. Li, I. You, Ensuring attribute privacy protection and fast decryption for outsourced data security in mobile cloud computing, Inf. Sci. 379 (2017) 42–61, doi:10.1016/j.ins.2016.04.015.

[45] R. Zhou, X. Zhang, X. Du, X. Wang, G. Yang, M. Guizani, File-centric multi-key aggregate keyword searchable encryption for industrial internet of things, IEEE Trans. Ind. Inf. 14 (8) (2018) 3648–3658.