# Cloud-Based Approximate Constrained Shortest Distance Queries Over Encrypted Graphs

With Privacy Protection

Meng Shen , *Member, IEEE*, Baoli Ma, Liehuang Zhu, *Member, IEEE*, Rashid Mijumbi, *Member, IEEE*, Xiaojiang Du, *Senior Member, IEEE*, and Jiankun Hu , *Senior Member, IEEE*

***Abstract*— Constrained shortest distance (CSD) querying is one of the fundamental graph query primitives, which finds the shortest distance from an origin to a destination in a graph with a constraint that the total cost does not exceed a given threshold. CSD querying has a wide range of applications, such as routing in telecommunications and transportation. With an increasing prevalence of cloud computing paradigm, graph owners desire to outsource their graphs to cloud servers. In order to protect sensitive information, these graphs are usually encrypted before being outsourced to the cloud. This, however, imposes a great challenge to CSD querying over encrypted graphs. Since perform- ing constraint filtering is an intractable task, existing work mainly focuses on unconstrained shortest distance queries. CSD querying over encrypted graphs remains an open research problem. In this paper, we propose Connor, a novel graph encryption scheme that enables approximate CSD querying. Connor is built based on an efficient, tree-based ciphertext comparison protocol, and makes use of symmetric-key primitives and the somewhat homomorphic encryption, making it computationally efficient. Using Connor, a graph owner can first encrypt privacy-sensitive graphs and then outsource them to the cloud server, achieving the necessary privacy without losing the ability of querying. Extensive experiments with real-world data sets demonstrate the effectiveness and efficiency of the proposed graph encryption scheme.**

***Index Terms*— Cloud computing, privacy, graph encryption, constrained shortest distance querying.**

Manuscript received April 4, 2017; revised August 19, 2017,

October 6, 2017, and October 29, 2017; accepted November 4, 2017. Date of publication November 16, 2017; date of current version January 3, 2018. This work was supported in part by the National Science Foundation of China under Grant 61602039, in part by the Beijing Natural Science Foundation under Grant 4164098, and in part by the China National Key Research and Development Program under Grant 2016YFB0800301. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Marina Blanton. *(Corresponding author: Liehuang Zhu.)*

M. Shen, B. Ma, and L. Zhu are with the Beijing Engineering Research Center of High Volume Language Information Processing and Cloud Computing Applications, School of Computer Science, Beijing Insti- tute of Technology, Beijing 100081, China (e-mail: shenmeng@bit.edu.cn; baolimasmile@bit.edu.cn; liehuangz@bit.edu.cn).

R. Mijumbi is with Bell Labs CTO, Nokia, D15Y6NT Dublin, Ireland (e-mail: rashid.mijumbi@nokia.com).

X. Du is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122 USA (e-mail: dxj@ieee.org).

1. Hu is with the School of Engineering and IT, University of New South Wales, Canberra, ACT 2610, Australia (e-mail: J.Hu@adfa.edu.au).

Color versions of one or more of the figures in this paper are available online at [http://ieeexplore.ieee.org.](http://ieeexplore.ieee.org/)

Digital Object Identifier 10.1109/TIFS.2017.2774451

* 1. INTRODUCTION

ECENT years have witnessed the prosperity of appli- cations based on graph-structured data [1], [2], such

**R**

as online social networks, road networks, web graphs [3], biological networks, and communication networks [4], [5]. Consequently, many systems for managing, querying, and analyzing massive graphs have been proposed in both acad- emia (e.g., GraphLab [6], Pregel [7] and TurboGraph [8]) and industry (e.g., Titan, DEX and GraphBase). With the prevalence of cloud computing, graph owners (e.g., enterprises and startups for graph-based services) desire to outsource their graph databases to a cloud server, which raises a great concern regarding privacy. An intuitive way to enhance data privacy is encrypting graphs before outsourcing them to the cloud. This, however, usually comes at the price of inefficiency, because it is quite difficult to perform operations over encrypted graphs. Shortest distance querying is one of the most fundamental graph operations, which finds the shortest distance, according to a specific criterion, for a given pair of source and destination in a graph. In practice, however, users may consider multiple criteria when performing shortest distance queries [2]. Taking the road network as an example, a user may want to know the shortest distance, in terms of travelling time, between two cities within a budget for total toll payment. This problem can be represented by a constrained shortest distance (CSD) query, which finds the shortest distance based on one criterion with

one or more constraints on other criteria.

In this paper, we focus on single-constraint CSD queries. This is because most practical problems can be represented as a single-constraint CSD query. For instance, such a query on a communication network could return the minimum cost from a starting node to a terminus node, with a threshold on routing delay. In addition, multi-constraint CSD queries can usually be decomposed into a group of sub-queries, each of which can be abstracted as a single-constraint CSD query. Formally, a CSD query1 is such that: given an origin *s*, a destination *t*, and a cost constraint *θ* , finding the shortest distance between *s* and *t* whose total cost *c* does not exceed *θ* .

Existing studies in this area can be roughly classified into two categories. The *first* category mainly focuses on the

1For simplicity, we refer to single-constraint CSD queries as CSD queries hereafter.

1556-6013 © 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.

See <http://www.ieee.org/publications_standards/publications/rights/index.html> for more information.

CSD query problem over unencrypted graphs [2], [9]–[12]. However, these methods cannot be easily applied in the encrypted graph environment, because many operations on plain graphs required in these methods (e.g., addition, multi- plication, and comparison) cannot be carried out successfully without a special design for encrypted graphs. The *second* category aims at enabling the shortest distance (or shortest path) queries over encrypted graphs [1], [13]. They usually adopt distance oracles such that the approximate distance between any two vertices can be efficiently computed, e.g., in a sublinear way. The main limitation of these approaches is that they are incapable of performing constraint filtering over the cloud-based encrypted graphs. Therefore, they cannot be directly applied to answering CSD queries.

Motivated by the limitations of existing schemes, our goal in this paper is to design a practical graph encryption scheme that enables CSD queries over encrypted graphs. As the CSD problem over plain graphs has been proved to be NP-hard [10], existing studies (e.g., [2]) usually resort to approximate solutions, which guarantee that the resulting dis- tance is no longer than *α* times of the shortest distance (where *α* is an approximation ratio predefined by graph owners), subject to the cost constraint *θ* . The encryption of graphs would make the CSD problem even more complicated. Hence, we also focus on devising an approximate solution.

Specifically, this paper presents Connor, a novel graph encryption scheme targeting the approximate CSD querying over encrypted graphs. Connor is built on a secure 2-hop cover labeling index (2HCLI), which is a type of distance oracle such that the approximate distance between any two vertices in a graph can be efficiently computed [1], [2]. The vertices of the graph in the secure 2HCLI are encrypted by particular pseudo-random functions (PRFs). In order to protect real values of graph attributes while allowing for cost filtering, we encrypt *costs* and *distances* (between pairs of vertices) by the order-revealing encryption (ORE) [14], [15] and the somewhat homomorphic encryption (SWHE) [16], respectively. Based on the ORE, we design a simple but efficient tree-based ciphertexts comparison protocol, which can accelerate the constraint filtering process on the cloud side.

The main contributions of this paper are as follows.

1. We propose a novel graph encryption scheme, Connor, which enables the approximate CSD querying. It can answer an *α*-CSD query in milliseconds and thereby achieves computational efficiency.
2. We design a tree-based ciphertexts comparison protocol, which helps us to determine the relationship of the sum of two integers and another integer over their cipher- texts with controlled disclosure. This protocol can also serve as a building block in other relevant application scenarios.
3. We present a thorough security analysis of Connor and demonstrate that it achieves the latest security definition named CQA2-security [17]. We also implement a pro- totype and conduct extensive experiments on real-world datasets. The evaluation results show the effectiveness and efficiency of the proposed scheme.

To the best of our knowledge, this is the first work that enables the approximate CSD querying over encrypted graphs. The rest of this paper is organized as follows. We summarize the related work in Section II and describe the background of the approximate CSD querying in Section III. We formally define the privacy-preserving approximate CSD querying prob- lem in Section IV. After that, the construction of Connor is presented in Section V, with a detailed description of the tree-based ciphertexts comparison protocol in Section VI. We exhibit the complexity and security analyses in Section VII, evaluate the proposed scheme through exten- sive experiments in Section VIII, and conclude this paper

in Section IX.

* 1. RELATED WORK

In an era of cloud computing, security and privacy become great concerns of cloud service users [18]–[23]. Here we briefly summarize the related work from two aspects, i.e., CSD querying over plain graphs and graph privacy protection.

1. *Plain CSD Queries*

The constrained shortest distance/path querying over plain graphs has attracted many research attentions. Hansen [9] pro- posed an augmented Dijkstra’s algorithm for exact constrained shortest path queries without an index. This method, however, resulted in a significant computational burden. In order to improve the querying efficiency, another solution [11] focused on approximate constrained shortest path queries, which were also index-free.

The state-of-the-art solution to the exact constrained shortest path querying with an index was proposed by Storandt [12], which accelerated query procedure with an indexing technique called contraction hierarchies. This approach still results in impractically high query processing cost. Wang et al. [2] proposed a solution to the approximate constrained shortest path querying over large-scale road networks. This method took full advantage of overlay graph techniques to construct an overlay graph based on the original graph, whose size was much smaller than that of the original one. Consequently, they built a constrained labeling index structure over the overlay graph, which greatly reduced the query cost. Unfortunately, all these solutions are merely suitable to perform queries over unencrypted graphs.

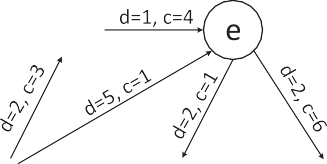
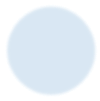
1. *Graph Privacy Protection*

Increasing concerns about graph privacy have been raised with the wide adoption of the cloud computing paradigm over the past decade. Chase and Kamara [17] first intro- duced the notion of graph encryption, where they proposed several constructions for graph operations, such as adjacency queries and neighboring queries. Cao et al. [24] defined and solved the problem of privacy-preserving query over encrypted graph data in cloud computing by utilizing the principle of “filtering-and-verification”. They built the feature-based index of a graph in advance and then chose the efficient inner product to carry out the filtering procedure. Some approa- ches [13], [25], [26] utilized the differential privacy tech- nique to query graphs privately, which might suffer from

weak security. These studies, however, introduced pro- hibitively great storage costs and were not practical for large- scale graphs. Meng et al. [1] proposed three computationally efficient constructions that supported the approximate shortest distance querying with distance oracles, which were provably secure against a semi-honest cloud server.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Secure multi-party computation (SMC) techniques have been widely applied to address the privacy-preserving shortest path problem [27]–[30], as well as other secure computation problems [31]. Aly et al. [28] focused on the shortest path problem over traditional combinatorial graph in a general multi-party computation setting, and proposed two proto- cols for securely computing shortest paths in the graphs. Blanton et al. [27] designed data-oblivious algorithms to securely solve the single-source single-destination shortest path problem, which achieved the optimal or near-optimal performance on dense graphs. Keller and Scholl [29] designed several oblivious data structures (e.g., priority queues) for SMC and utilized them to compute shortest paths on general graphs. Gupta et al. [30] proposed an SMC-based approach for finding policy-compliant paths that have the least routing cost or satisfy bandwidth demands among different network domains. However, existing general-purpose SMC solutions for the shortest path problem may result in heavy communi- cation overhead.



Although there are respectable studies on graph querying over encrypted graphs, the privacy-preserving CSD query remains unsolved. In this paper, we propose a novel and efficient graph encryption scheme for CSD queries.

TABLE I

LIST OF NOTATIONS

* 1. BACKGROUND

This section presents the formal definition of the CSD query problem and introduces the 2HCLI structure for graph queries.

1. *Approximate CSD Query*

Let *G (V, E)* be a directed graph2 with a vertex set *V* and an edge set *E* . Each edge *e E* is associated with a *distance d(e)* 0 and a *cost c(e)* 0. We regard the cost *c(e)* as the constraint. We denote the set of edges that connect two vertices as a *path*. For a path *P e e e* , its distance

≥ ≥

∈

=

= *(* 1*,* 2*, ... , k)*

*d(P)* is defined as *d(P)* = Σ*k d(ei)*, which indicates the

*i*=1

Fig. 1. An example illustrating the *α*-CSD query over a graph.

*Definition 1 (α-CSD QUERY):* Given an origin *s*, a desti- nation *t*, a cost constraint *θ* and an approximation ratio *α*, an *α*-CSD query returns the distance *d(P)* of a path *P*, such that *c(P) θ* and *d(P) α dopt*, where *dopt* is the optimal answer to the exact CSD query with the origin *s*, destination *t* and cost constraint *θ* .

≤ ≤ ·

Fig. 1 shows a simple graph with five vertices, where the

distance and cost of each edge are marked alongside it. Given

distance from its origin to its destination. Similarly, we define

the cost of *P* as *c(P)* = Σ*k c(ei)*. The notations throughout

in T

the paper are summarized *i*=1 able I.

= 6, where the

an origin *a*, a destination *c*, a cost constraint *θ* = 4, the exact

CSD query returns the optimal distance *d*

*opt*

Given a graph *G*, an origin vertex *s V* , a destination vertex *t V* , and a cost constraint *θ* , a CSD query is to find the the shortest distance *d* between *s* and *t* with the total cost no more than *θ* . Since the CSD query problem has been proved to be NP-hard [10], we keep in line with existing solutions [2] and focus on proposing an approximate CSD solution in this paper.

∈

∈

Inspired by a common definition of the approximate shortest path query over plain graphs [2], we define the approximate CSD query (i.e., *α*-CSD query) as follows.

2We refer to *G* as a directed graph in this paper, unless otherwise specified.

*opt*

corresponding path is *(a, b, c)*. For an approximation ratio *α* 1*.*5, a valid answer to the *α*-CSD query with the same parameters (e.g, the origin *a*, the destination *c*, and *θ* 4) is 8, with the corresponding path *Pα (a, e, b, c)*. That is because *d(Pα)* 8 *< α dopt* 9 and *c(Pα)* 3 *< θ* .

= · = =

=

=

=

Based on the above definition, given two paths *P*1 and *P*2 with the same origin and destination, we say that *P*1 *α*-*dominates P*2 iff *c(P*1*) c(P*2*)* and *d(P*1*) α d(P*2*)*. With this principle, we can reduce the construction complexity of graph index significantly, because a great deal of redundant entries in the index can be filtered out. We will make a further illustration in the following subsection.

≤ ≤ ·

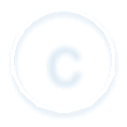
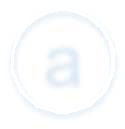
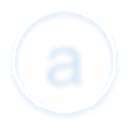
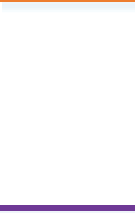
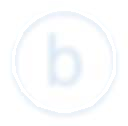
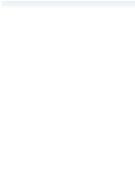
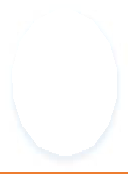
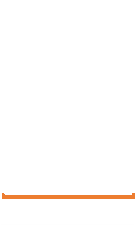
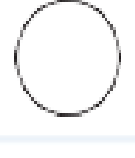
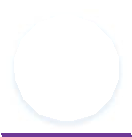
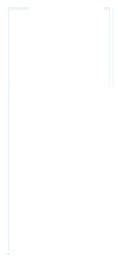
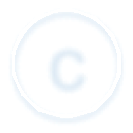
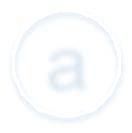
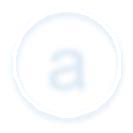
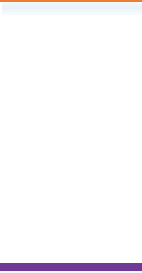
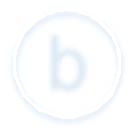
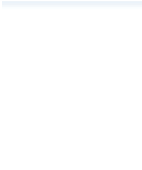
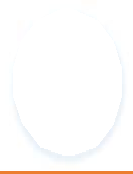
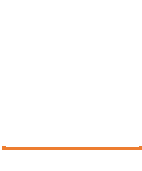
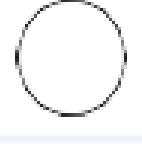
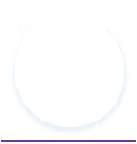
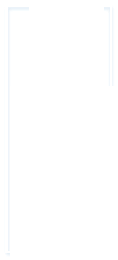


Fig. 2. A 2HCLI example of the basic shortest distance query. Each entity *d* in 2HCLI alongside the arrow indicates the shortest distance from the starting vertex to the ending vertex, e.g., the shortest distance from *a* to *e* is 3.

1. *Constructing Labeling Index*

The encrypted index designed in this paper is mainly constructed based on the well-known 2HCLI, which is a special data structure that supports the shortest distance query efficiently [2], [32], [33]. Here we briefly describe the basic idea of the 2HCLI, and illustrate its application in building a constrained labeling index.

Given a graph *G (V, E)* with a vertex set *V* and an edge set *E* , each vertex *v V* is associated with an in-label set and an out-label set, which are denoted by *6in (v)* and *6out(v)*, respectively. Each entity in *6in(v)* corresponds to the shortest distance from a vertex *u V* to *v*. It implies that *v* is reachable from *u* by one or more paths, but is not necessarily a neighbor, or 2-hop neighbour, of *u*. Similarly, each entity in *6out(v)* corresponds to the shortest distance from *v* to another vertex *u* in *V* . To answer a shortest distance query from an origin *s* to a destination *t*, we first find the common vertices in the labels *6out(s)* and *6in (t)*, and then select the shortest distance from *s* to *t*. Note that the entities in *6in (v)* and *6out(v)* are carefully selected [33] so that the distance of any two vertices *s* and *t* can be computed by *6out(s)* and *6in (t)*. Considering the graph in Fig. 1, if we ignore the cost criterion of edges, the *basic* unconstrained shortest distance query with an origin *a* and a destination *c* can be answered with the help of the 2HCLI, as shown in Fig. 2. Given the labels *6out(a)* and *6in(c)*, it is easy to obtain the set of common vertices, which consists of vertices *b* and *e*. The final answer to the basic shortest distance query should be 5,

∈

=

∈

because *d(a, e)* + *d(e, c)* = 5 *< d(a, b)* + *d(b, c)* = 6.

Fig. 3. A 2HCLI example of the exact CSD query. Each entity *(dis, cost)* in the 2HCLI alongside the arrow indicates the distance and cost, respectively. The shortest distance from *a* to *e* with a cost constraint *θ* = 4 is 5.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |
|  |  |  |
|  |  |

Fig. 4. The resulting 2HCLI after performing the offline filtering on the original 2HCLI in Fig. 3. Each entity *(u, d, c)* in the 2HCLI indicates the vertex identifier, distance and cost, respectively. The answer to the approximate CSD query (i.e., the origin *a*, the destination *c*, *α* = 1*.*5,

and *θ* = 4) is 6, which happens to be the answer to the exact CSD query.

of vertices could increase dramatically in large-scale graphs, which results in a higher complexity in constructing the 2HCLI and calculating the answers to a CSD query.

In order to improve the querying efficiency, we adopt a methodology that combines an *offline* filtering operation and an *online* filtering operation.

The offline filtering aims at reducing the construction com- plexity of the 2HCLI and decreasing the number of entries in the in-label and out-label sets as many as possible. We adopt the method proposed in [2]. The entities in the 2HCLI are carefully selected in such a way that for any CSD query from *u* to *v* with a cost constraint *θ* , the query can be answered correctly using only the 2HCLI. Since the construction of the 2HCLI should be independent of the cost constraint in specific CSD queries, we can use the definition of *α*-*domination* to filter out redundant entries in the in- and out-label sets.

Taking for example the two entries from *e* to *c* with *α* = 1*.*5

*ec*

=

Although it is simple and straightforward to construct the 2HCLI for a graph with only the distance criterion, construct- ing a labeling index based on the 2HCLI for the CSD query is much more complex. That is because in the CSD query setting with two types of edge criteria, there might be multiple combinations of distance and cost for each pair of vertices in the labels *6in(v)* and *6out(v)*. For ease of illustration, we also take as an example the graph, as well as the CSD query, in Fig. 1. The corresponding 2HCLI is shown in Fig. 3, where the 2-tuple alongside each arrow represents the distance and cost from the starting vertex to the ending vertex. Note that in the shortest distance query in Fig. 2, the shortest distance from *a* to *c* via *e* is unique. However, in the CSD query setting depicted in Fig. 3, there are four possible distances with different costs from *a* to *c* via *e*. Due to the existence of the cost criterion, the number of possible distances for each pair

in Fig. 3, the path *P*1 *(e, b, c)* with the *(dis, cost)*-tuple of (3,2) *α*-*dominates* another path *P*2 *(e, c)* with the *(dis, cost)*-tuple of (2,6). Therefore, the entry corresponding to the path *P*2 can be filtered out (as depicted by a dashed arrow), which helps to reduce the number of entries in *6in (c)*. The resulting 2HCLI is exhibited in Fig. 4. We refer the reader to [2] for more construction details.

The online filtering aims at selecting the possibly valid answers to a given CSD query, based on only the 2HCLI. For instance, given an *α*-CSD query from *a* to *c* with a

*ec*

*ec*

=

cost constraint *θ* 4, we can first find the common vertex set *V* j between *6out(a)* and *6in(c)*, and then return the minimum *d(a,v) d(v, c)* with *c(a,v) c(v, c) θ* for each

=

+ + ≤

*v V* j. Since the above comparisons should be conducted with the corresponding ciphertexts, an efficient online filtering

∈

approach will be devised in Section VI.

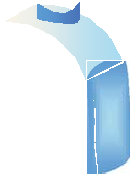
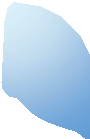


Fig. 5. The system model of privacy-preserving CSD query scheme.

the following probabilistic experiments where is a semi- honest adversary, is a simulator, and *Setup* and *Quer y* are (stateful) leakage functions.

1. *RealM,A(λ):*

*S L L*

*A*

* outputs a graph *G*, an approximation ratio *α* and an amplification factor *φ*.

*A*

* The challenger begins by running *Gen(*1*λ)* to gen- erate a secret key *K* and a public/secret-key pair

*( pk, sk)*, and then computes the encrypted index *6*˜

* 1. PROBLEM FORMULATION

˜

This section presents the system model and the security model of the privacy-preserving *α*-CSD querying, as well as the preliminaries of the proposed graph encryption scheme.

1. *System Model*

We adopt the general system model in the litera- ture [1], [17] for the privacy-preserving *α*-CSD querying, as illustrated in Fig. 5, which mainly involves two types of entities, namely a *user* and a *cloud server*.

The *user* constructs the secure searchable index for the graph and outsources the encrypted index along with the encrypted graph to the cloud server. When the user, say Alice, performs an *α*-CSD query over her encrypted graph, she first generates a query token and then submits it to the cloud server. Upon receiving Alice’s query token, the cloud server executes the pre-designed query algorithms to match entries in the secure index with the token. Finally, the cloud server replies the user with the answer to the *α*-CSD query.

The graph encryption scheme is formally defined as follows. *Definition 2 (GRAPH ENCRYPTION):* A graph encryption scheme *M (Key Gen, Setup, Query)* consists of three

=

polynomial-time algorithms that work as follows:

* + *(K, pk, sk) Key Gen(λ)*: is a probabilistic secret key generation algorithm that takes as input a security para- meter *λ* and outputs a secret key *K* and a public/secret- key pair *( pk, sk)*.

←

˜ ←

* + *6 Setup(α, K, pk, sk,φ, G)*: is a graph encryption algorithm that takes as input an approximation ratio *α*, a secret keys *K* , a key pair *( pk, sk)*, an amplification factor *φ* and a graph *G*, and outputs a secure index *6*.

˜

* + *(distq, ) Query((K, pk, sk, 0, q), 6)*: is a two- party protocol between a user that holds a secret key *K* , a key pair *( pk, sk)* and a query *q*, and a cloud server that holds an encrypted graph index *6*. After executing this protocol, the user receives the distance *distq* as the query result and the cloud server receives a terminator ⊥.

˜

⊥ ← ˜

1. *Security Model*

Graph encryption is a generalization of symmetric search- able encryption (SSE) [34]–[38]. Thus, we adopt the security definition of SSE settings in our graph encryption scheme. This security definition is consistent with the latest proposed security definition in [17], [35] and [39] which is also known as CQA2-security (i.e., the chosen-query attack security). Now we present the formal CQA2-security definition as follows.

*Definition 3 (CQA2-Security Model):* Let *M (Key Gen*, *Setup, Query)* be a graph encryption scheme and consider

=

by *Setup(α, K, pk, sk,φ, G)*. The challenger sends the encrypted index *6* to *A*.

* + makes a polynomial number of adaptive queries, and for each query *q*, and the challenger execute *Query((K, pk, sk, 0, q), 6)*.

˜

*A*

*A*

* + computes a bit *b* 0*,* 1 as the output of the experiment.

*A* ∈ { }

1. *IdealM,A,S(λ):*
   * outputs a graph *G*, an approximation ratio *α* and an amplification factor *φ*.

*A*

* + Given the leakage function *LSetup(G)*, *S* simulates a secure graph index *6*∗ and sends it to *A*.

˜

* + *A* makes a polynomial number of adaptive queries.

For each query *q*, *S* is given the leakage function *LQuer y(G, Q)*, and *A* and *S* execute a simulation of *Query*, where *A* is playing the role of the cloud server and *S* is playing the role of the user.

* + computes a bit *b* 0*,* 1 as the output of the experiment.

*A* ∈ { }

We say that the graph encryption scheme *M* = *(Key Gen*, *Setup, Query)* is *(LSetup, LQuer y)*-secure against the adaptive

chosen-query attack, if for all PPT adversaries *A*, there exists a PPT simulator *S* such that

|**Pr**[**Real***M,A(λ)* = 1]− **Pr**[**Ideal***M,A,S(λ)* = 1]| ≤ *negl(λ),*

where *negl(λ)* is a negligible function.

1. *Preliminaries*

Now we briefly introduce an encryption technique employed in our design, i.e., the order-revealing encryption.

*Order-revealing encryption (ORE)* is a generalization of the order-preserving encryption (OPE) scheme, but provides stronger security guarantees. As pointed by Naveed et al. [40], the OPE-encrypted databases are extremely vulnerable to *inference attacks*. To address this limitation, the ORE scheme has been proposed [14], [15], which is a tuple of three algo- rithms *M (ORE.Setup, ORE.Encrypt, ORE.Compare)* described as follows:

=

* + ORE.Setup(1*λ*) *sk*: Input a security parameter *λ*, output the secret key *sk*.

→

* + ORE.Encrypt(*sk, m*) *ct*: Input a secret key *sk* and a message *m*, output a ciphertext *ct*.

→

* + ORE.Compare(*ct*1*, ct*2) *z*: Input two ciphertexts *ct*1 and *ct*2, output a bit *r* 0*,* 1 , which indicates the greater-than or less-than relationship of the corresponding plaintexts *m*1 and *m*2.

∈ { }

→