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 performing 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 Institute 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).

J. 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.

Digital Object Identifier 10.1109/TIFS.2017.2774451

I. Introduction

RECENT years have witnessed the prosperity of applications based on graph-structured data [1], [2], such 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 academia (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 query[1] 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

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

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, multiplication, 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 distance 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 ciphertexts 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 prototype 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 problem 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 extensive experiments in Section VIII, and conclude this paper in Section IX.

II. 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.

A. Plain CSD Queries

The constrained shortest distance/path querying over plain graphs has attracted many research attentions. Hansen [9] proposed 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.

B. 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 introduced 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 approaches [13], [25], [26] utilized the differential privacy technique to query graphs privately, which might suffer from weak security. These studies, however, introduced prohibitively 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 protocols 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 communication 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.

III. Background

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

A. Approximate CSD Query

Let $G = (V, E)$ be a directed graph[2] with a vertex set V and an edge set E. Each edge $e \in E$ is associated with a *distance* $d(e) \geq 0$ and a *cost* $c(e) \geq 0$. 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 $P = (e_1, e_2, \ldots, e_k)$, its distance $d(P)$ is defined as $d(P) = \sum_{i=1}^{k} d(e_i)$, which indicates the distance from its origin to its destination. Similarly, we define the cost of P as $c(P) = \sum_{i=1}^{k} c(e_i)$. The notations throughout the paper are summarized in Table I.

Given a graph G, an origin vertex $s \in V$, a destination vertex $t \in V$, 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.

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

TABLE I
List of Notations

Notation	**Meaning**
$G = (V, E)$	Input graph
n, m	Number of vertices and edges in G
$d(e), c(e)$	Distance and cost of an edge e
$d(u, v), c(u, v)$	Distance and cost of the edge from u to v
$s, t, \alpha, \phi, \theta$	Origin, destination, approximation ratio, amplification factor and cost constraint in an α-CSD query
$\Delta, \widetilde{\Delta}$	Plain and encrypted graph index
$\Delta_{in}(v), \Delta_{out}(v)$	In- and out-label set associated with vertex v
d_θ	Depth of a cost constraint tree
β	Length of a path code
$E(m)$	ORE ciphertext of m
λ	Security parameter
k	Output length of ORE encryption
z	Output length of the SWHE algorithm
$\tau_{s,t}$	Query token
Y	Candidate sets as the outputs of the cost constraint filtering
$\mathcal{B}$	Maximum distance over all the sketches

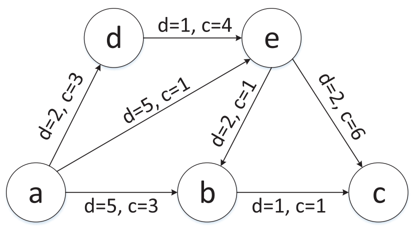


Fig. 1. An example illustrating the α-CSD query over a graph.

Definition 1 (α-CSD QUERY): Given an origin s, a destination 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 $c(P) \leq \theta$ and $d(P) \leq \alpha \cdot d_{opt}$, where d_{opt} 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 an origin a, a destination c, a cost constraint $\theta = 4$, the exact CSD query returns the optimal distance $d_{opt} = 6$, where the corresponding path is (a, b, c). For an approximation ratio $\alpha = 1.5$, a valid answer to the α-CSD query with the same parameters (e.g, the origin a, the destination c, and $\theta = 4$) is 8, with the corresponding path $P_\alpha = (a, e, b, c)$. That is because $d(P_\alpha) = 8 < \alpha \cdot d_{opt} = 9$ and $c(P_\alpha) = 3 < \theta$.

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 $c(P_1) \leq c(P_2)$ and $d(P_1) \leq \alpha \cdot d(P_2)$. 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.

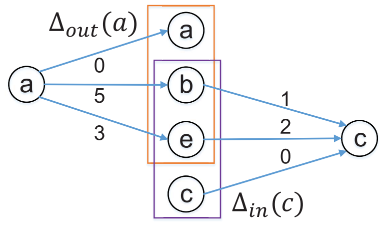


Fig. 2. A 2HCLI example of the basic shortest distance query. Each entity d in 2HCLI alongside the arrow indicates the shortest distance from the starting vertex to the ending vertex, e.g., the shortest distance from a to e is 3.

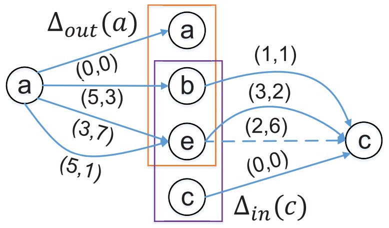


Fig. 3. A 2HCLI example of the exact CSD query. Each entity $(dis, cost)$ in the 2HCLI alongside the arrow indicates the distance and cost, respectively. The shortest distance from a to e with a cost constraint $\theta = 4$ is 5.

$\Delta(a)$	$\Delta_{in}(a)$	(a, 0, 0)
	$\Delta_{out}(a)$	(a, 0, 0), (b, 5, 3), (e, 5, 1), (e, 3, 7)
$\Delta(c)$	$\Delta_{in}(c)$	(b, 1, 1), (c, 0, 0), (e, 3, 2)
	$\Delta_{out}(c)$	(c, 0, 0)

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, $\alpha = 1.5$, and $\theta = 4$) is 6, which happens to be the answer to the exact CSD query.

B. 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 $v \in V$ is associated with an in-label set and an out-label set, which are denoted by $\Delta_{in}(v)$ and $\Delta_{out}(v)$, respectively. Each entity in $\Delta_{in}(v)$ corresponds to the shortest distance from a vertex $u \in V$ 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 $\Delta_{out}(v)$ 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 $\Delta_{out}(s)$ and $\Delta_{in}(t)$, and then select the shortest distance from s to t. Note that the entities in $\Delta_{in}(v)$ and $\Delta_{out}(v)$ are carefully selected [33] so that the distance of any two vertices s and t can be computed by $\Delta_{out}(s)$ and $\Delta_{in}(t)$.

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 $\Delta_{out}(a)$ and $\Delta_{in}(c)$, 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 $d(a, e) + d(e, c) = 5 < d(a, b) + d(b, c) = 6$.

Although it is simple and straightforward to construct the 2HCLI for a graph with only the distance criterion, constructing 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 $\Delta_{in}(v)$ and $\Delta_{out}(v)$. 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 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 complexity 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 $\alpha = 1.5$ in Fig. 3, the path $P_{ec}^1 = (e, b, c)$ with the $(dis, cost)$-tuple of (3,2) *α-dominates* another path $P_{ec}^2 = (e, c)$ with the $(dis, cost)$-tuple of (2,6). Therefore, the entry corresponding to the path P_{ec}^2 can be filtered out (as depicted by a dashed arrow), which helps to reduce the number of entries in $\Delta_{in}(c)$. 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 cost constraint $\theta = 4$, we can first find the common vertex set V' between $\Delta_{out}(a)$ and $\Delta_{in}(c)$, and then return the minimum $d(a, v) + d(v, c)$ with $c(a, v) + c(v, c) \leq \theta$ for each $v \in V'$. Since the above comparisons should be conducted with the corresponding ciphertexts, an efficient online filtering approach will be devised in Section VI.

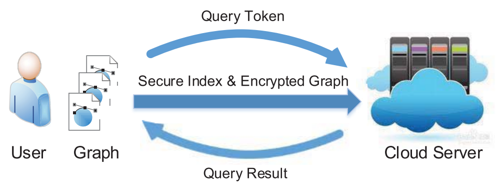


Fig. 5. The system model of privacy-preserving CSD query scheme.

IV. 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.

A. System Model

We adopt the general system model in the literature [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 $\Pi = (KeyGen, Setup, Query)$ consists of three polynomial-time algorithms that work as follows:

- $(K, pk, sk) \leftarrow KeyGen(\lambda)$: is a probabilistic secret key generation algorithm that takes as input a security parameter λ and outputs a secret key K and a public/secret-key pair (pk, sk).
- $\widetilde{\Delta} \leftarrow Setup(\alpha, K, pk, sk, \phi, G)$: 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 $\widetilde{\Delta}$.
- $(dist_q, \perp) \leftarrow Query((K, pk, sk, \Phi, q), \widetilde{\Delta})$: 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 $\widetilde{\Delta}$. After executing this protocol, the user receives the distance $dist_q$ as the query result and the cloud server receives a terminator $\perp$.

B. Security Model

Graph encryption is a generalization of symmetric searchable 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 $\Pi = (KeyGen, Setup, Query)$ be a graph encryption scheme and consider the following probabilistic experiments where $\mathcal{A}$ is a semi-honest adversary, $\mathcal{S}$ is a simulator, and $\mathcal{L}_{Setup}$ and $\mathcal{L}_{Query}$ are (stateful) leakage functions.

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

- $\mathcal{A}$ outputs a graph G, an approximation ratio α and an amplification factor ϕ.
- The challenger begins by running $Gen(1^{\lambda})$ to generate a secret key K and a public/secret-key pair (pk, sk), and then computes the encrypted index $\widetilde{\Delta}$ by $Setup(\alpha, K, pk, sk, \phi, G)$. The challenger sends the encrypted index $\widetilde{\Delta}$ to $\mathcal{A}$.
- $\mathcal{A}$ makes a polynomial number of adaptive queries, and for each query q, $\mathcal{A}$ and the challenger execute $Query((K, pk, sk, \Phi, q), \widetilde{\Delta})$.
- $\mathcal{A}$ computes a bit $b \in \{0, 1\}$ as the output of the experiment.

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

- $\mathcal{A}$ outputs a graph G, an approximation ratio α and an amplification factor ϕ.
- Given the leakage function $\mathcal{L}_{Setup}(G)$, $\mathcal{S}$ simulates a secure graph index $\widetilde{\Delta}^*$ and sends it to $\mathcal{A}$.
- $\mathcal{A}$ makes a polynomial number of adaptive queries. For each query q, $\mathcal{S}$ is given the leakage function $\mathcal{L}_{Query}(G, Q)$, and $\mathcal{A}$ and $\mathcal{S}$ execute a simulation of $Query$, where $\mathcal{A}$ is playing the role of the cloud server and $\mathcal{S}$ is playing the role of the user.
- $\mathcal{A}$ computes a bit $b \in \{0, 1\}$ as the output of the experiment.

We say that the graph encryption scheme $\Pi = (KeyGen, Setup, Query)$ is $(\mathcal{L}_{Setup}, \mathcal{L}_{Query})$-secure against the adaptive chosen-query attack, if for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that

$$|\mathbf{Pr}[\mathbf{Real}_{\Pi,\mathcal{A}}(\lambda) = 1] - \mathbf{Pr}[\mathbf{Ideal}_{\Pi,\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq negl(\lambda),$$

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

C. 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 algorithms $\Pi = (ORE.Setup, ORE.Encrypt, ORE.Compare)$ described as follows:

- ORE.Setup(1^{λ})$\rightarrow sk$: Input a security parameter λ, output the secret key sk.
- ORE.Encrypt(sk, m)$\rightarrow ct$: Input a secret key sk and a message m, output a ciphertext ct.
- ORE.Compare(ct_1, ct_2)$\rightarrow z$: Input two ciphertexts ct_1 and ct_2, output a bit $r \in \{0, 1\}$, which indicates the greater-than or less-than relationship of the corresponding plaintexts m_1 and m_2.

Algorithm 1 Setup Algorithm for $GraphEnc_1$

```
Input: A secret key $K$, a key pair $(pk, sk)$, an approximation
    ratio $\alpha$, an amplification factor $\phi$, and an original graph $G$.
Output: The encrypted graph index $\widetilde{\Delta}$.
1: Generate the 2-hop labeling index $\Delta = \{\Delta_{out}, \Delta_{in}\}$ from
     $G$.
2: Initialize two dictionaries $I_{out}$ and $I_{in}$.
3: Let $\mathcal{B}$ be the maximum distance over all the sketches and
     set $N = 2\mathcal{B} + 1$.
4: for each $u \in G$ do
5:     Set $T_{out,u} = h(K, u||1)$, $T_{in,u} = h(K, u||2)$.
6:     for each $(v, d_{u,v}, c_{u,v}) \in \Delta_{out}(u)$ do
7:         Compute $V = h(K, v||0)$.
8:         Compute $D_{u,v} = \text{SWHE.Enc}(pk, 2^{N-d_{u,v}})$.
9:         Compute $C_{u,v} = \text{ORE.Enc}(K, \phi c_{u,v})$.
10:        Insert $(V, D_{u,v}, C_{u,v})$ into the dictionary $I_{out}[T_{out,u}]$.
11:    end for
12:    Repeat the above procedure for each sketch in $\Delta_{in}(u)$
           and add entries into $I_{in}[T_{in,u}]$.
13: end for
14: return $\widetilde{\Delta} = \{I_{out}, I_{in}\}$ as the encrypted graph index.
```

V. Construction of Connor

In this section, we introduce our graph encryption scheme Connor for the privacy-preserving α-CSD querying.

A. Construction Overview

The construction process is based on two particular pseudo-random functions h and g, and a somewhat homomorphic encryption (SWHE) scheme. In this paper, we adopt the concrete instantiation of a SWHE scheme in the literature [16]. The parameters of h and g are illustrated in Equation (1),

$$h : \{0, 1\}^{\lambda} \times \{0, 1\}^{*} \rightarrow \{0, 1\}^{\lambda} \quad (1a)$$

$$g : \{0, 1\}^{\lambda} \times \{0, 1\}^{*} \rightarrow \{0, 1\}^{\lambda+z+k} \quad (1b)$$

where λ is the security parameter, and k and z are the output lengths of the ORE and SWHE encryptions, respectively.

We start with a straightforward construction $GraphEnc_1 = (KeyGen, Setup, Query)$ as follows, including:

- **KeyGen:** Given the security parameter λ, the *user* randomly generates a secret key K and a pair of public and secret keys (pk, sk) for SWHE.
- **Setup:** Given an original graph G, an approximation ratio α, and an amplification factor ϕ, the *user* obtains the encrypted graph index by using Algorithm 1. The 2HCLI $\Delta = \{\Delta_{out}, \Delta_{in}\}$ of G can be generated by the method described in Section III-B.
  Let $\mathcal{B}$ be the maximum distance over all the sketches and $N = 2\mathcal{B}+1$. Motivated by the literature [1], each distance $d_{u,v}$ is encrypted as $2^{N-d_{u,v}}$ by the SWHE to protect its real value (line 8). Considering that $2^x + 2^y$ is bounded by $2^{max(x,y)-1}$, the SWHE encryption of distance allows for obtaining the minimum sum over a certain number of distance pairs.
  Each cost $c_{u,v}$, multiplied by the amplification factor ϕ, is encrypted by the ORE encryption (line 9). ϕ is a big integer and should be carefully selected to enlarge the plaintext space of $c_{u,v}$. In practice, the product of ϕ and the maximum cost value over all the sketches should be sufficiently large (e.g., at least 2^{80}), which is used to provide a sufficient randomness to the inputs. Since ϕ is kept private by the *user*, the *cloud server* cannot learn the real values of $c_{u,v}$.
- **Query:** To perform an α-CSD query with an origin s, a destination t, and a cost constraint θ, the *user* generates query tokens $\tau_s = h(K, s||1)$ and $\tau_t = h(K, t||2)$, and sends them to the *cloud server*. The *cloud server* obtains $I_{out}[\tau_s]$ and $I_{in}[\tau_t]$ from the index. For each encrypted vertex identifier v that appears in both $I_{out}[\tau_s]$ and $I_{in}[\tau_t]$, the *cloud server* performs a cost constraint filtering operation (which will be described in details in Section VI), and adds each pair $(D_{s,v}, D_{v,t})$ which satisfies the cost constraint $\phi\theta$ into a candidate set Y. Note that the cost constraint is multiplied by ϕ because we encrypt the cost $\phi c_{u,v}$, instead of $c_{u,v}$.
  Then, the *cloud server* directly obtains $d = \sum_{i=1}^{|Y|} d_i$, where $d_i = \text{SWHE.Eval}(\times, D^i_{s,v}, D^i_{v,t})$ for each pair $(D^i_{s,v}, D^i_{v,t})$ in Y. The correctness of the above calculation follows homomorphic properties of SWHE. We refer the readers to [1] for more details.
  Finally, the *cloud server* returns d to the *user*, who, in turn, obtains the answer to the α-CSD query by decrypting d with its secret key sk.

Note that this straightforward approach does not only correctly answer the α-CSD query over encrypted graphs, but also protects the vertex identifier, distance, and cost information.

However, the encrypted graph index obtained from Algorithm 1, without performing any queries, still results in information leakage. On one hand, it reveals the length of each encrypted sketch, i.e., $I_{out}[u]$ and $I_{in}[u]$, as well as the order information of ORE-encrypted costs in all sketches. On the other hand, it also discloses the number of common vertices between $I_{out}[u]$ and $I_{in}[v]$, which indicates the number of vertices that connect u to v. In particular, if the *cloud server* knows that there is no common vertex between $I_{out}[u]$ and $I_{in}[v]$, it learns that u cannot reach v.

B. Privacy-preserving α-CSD Querying

In order to enhance protection of sensitive information, we construct a privacy-preserving α-CSD querying scheme $GraphEnc_2 = (KeyGen, Setup, Query)$, where the key generation procedure is the same as in $GraphEnc_1$, with improved index construction and CSD query procedures as exhibited in Algorithms 2 and 3, respectively.

The *Setup* for $GraphEnc_2$ works as follows. The *user* first builds the 2HCLI Δ of graph G, and then encrypts sketches associated with $u \in G$ (i.e., $\Delta_{out}(u)$ and $\Delta_{in}(u)$), as described in lines 2-17.

Note that in order to prevent the leakage of the sketch size in the previous straightforward approach, we split each encrypted sketch $I_{out}(u)$ and $I_{in}(u)$, and ensure that they are stored in the dictionary separately, with a size of one. More precisely, we utilize a counter ω and generate the unique

Algorithm 2 Setup Algorithm for $GraphEnc_2$

```
Input: A secret key $K$, a key pair $(pk, sk)$, an approximation ratio $\alpha$, an amplification factor $\phi$, and an original graph $G$.
Output: The encrypted graph index $\widetilde{\Delta}$.
1: Generate the 2HCLI $\Delta = \{\Delta_{out}, \Delta_{in}\}$ of $G$.
2: Initialize two dictionary $I_{out}$ and $I_{in}$.
3: Let $\mathcal{B}$ be the maximum distance over the sketches and set $N = 2\mathcal{B} + 1$.
4: for each $u \in G$ do
5:     Set $S_{out,u} = h(K, u||1)$, $T_{out,u} = h(K, u||2)$, $S_{in,u} = h(K, u||3)$, and $T_{in,u} = h(K, u||4)$.
6:     Initialize a counter $\omega = 0$
7:     for each $(v, d_{u,v}, c_{u,v}) \in \Delta_{out}(u)$ do
8:         Compute $V = h(K, v||0)$.
9:         Compute $D_{u,v} = \text{SWHE.Enc}(pk, 2^{N-d_{u,v}})$.
10:        Compute $C_{u,v} = \text{ORE.Enc}(K, \phi c_{u,v})$.
11:        Set $T_{out,u,v} = h(T_{out,u}, \omega)$ and $S_{out,u,v} = g(S_{out,u}, \omega)$.
12:        Compute $\Psi_{u,v} = S_{out,u,v} \oplus (V||D_{u,v}||C_{u,v})$.
13:        Set $I_{out}[T_{out,u,v}] = \Psi_{u,v}$.
14:        Set $\omega = \omega + 1$.
15:    end for
16:    Repeat the above procedure for each sketch in $\Delta_{in}(u)$ and obtain $I_{in}[T_{in,u,v}]$, except that: (i) set $T_{in,u,v} = h(T_{in,u}, \omega)$ and $S_{in,u,v} = g(S_{in,u}, \omega)$, and (ii) compute $\Psi_{u,v} = S_{in,u,v} \oplus (V||D_{u,v}||C_{u,v})$.
17: end for
18: return $\widetilde{\Delta} = \{I_{out}, I_{in}\}$ as the encrypted graph index.
```

$T_{out,u,v}$ and $S_{out,u,v}$ for each entity in $\Delta_{out}(u)$ (line 11). Similarly, the unique $T_{in,u,v}$ and $S_{in,u,v}$ for each entity in $\Delta_{in}(u)$ can be generated (line 16). The $T_{out,u,v}$ (or $T_{in,u,v}$) indicates the position that this entity will be stored in I_{out} (or I_{in}), which ensures each position in the dictionary I_{out} (or I_{in}) having only one entity.

$S_{out,u,v}$ (or $S_{in,u,v}$) is used to make an XOR operation with $(V||D_{u,v}||C_{u,v})$. Since $S_{out,u,v}$ (or $S_{in,u,v}$) is different for each sketch, the XOR operation makes the resulting $\Psi_{u,v}$ indistinguishable, which guarantees that the *static* encrypted graph index $\widetilde{\Delta}$ reveals neither the number of common vertices between $I_{out}(u)$ and $I_{in}(v)$, nor the order information of costs.

The *Query* in Algorithm 3 works as follows. Assume that the *user* asks for the shortest distance between s and t, whose total cost does not exceed θ. She first generates the query token $\tau_{s,t}$ and sends it to the *cloud server* (lines 1-3). Upon receiving the token $\tau_{s,t}$, the *cloud server* searches in the index and obtains L_s and L_t (lines 5-22). That is, the *cloud server* iteratively judges whether the dictionary I_{out} (I_{in}) contains the key $T_{out,s,v}$ ($T_{in,v,t}$) or not. If it exists, then it adds the corresponding entity into the set L_s (L_t).

Once L_s and L_t are obtained, the *cloud server* performs the cost constraint filtering (line 23) and computes d (line 24), which are the same as described in the straightforward approach. Finally, the *user* gets the final answer by decrypting d, which is returned by the *cloud server*, using its sk.

Algorithm 3 Query Algorithm for $GraphEnc_2$

```
Input: The user's input are the secret key $K$, secret key pair $(pk, sk)$, an amplification factor $\Phi$, and the query $q = (s, t, \theta)$. The cloud server's input is the encrypted index $\widetilde{\Delta}$.
Output: user's output is $dist_q$ and cloud server's output is $\perp$.
1: user generates $S_{out,s} = h(K, s||1)$, $T_{out,s} = h(K, s||2)$, $S_{in,t} = h(K, t||3)$ and $T_{in,t} = h(K, t||4)$.
2: user constructs a cost constraint tree $T_\theta$ based on $\phi * \theta$ using secret $K$ as described in Section VI.
3: user sends $\tau_{s,t} = (S_{out,s}, T_{out,s}, S_{in,t}, T_{in,t}, T_\theta)$ to cloud server.
4: cloud server parses $\tau_{s,t}$ as $(S_{out,s}, T_{out,s}, S_{in,t}, T_{in,t}, T_\theta)$.
5: cloud server initializes a set $L_s$ and a counter $\omega = 0$.
6: cloud server computes $T_{out,s,v} = h(T_{out,s}, \omega)$.
7: while $I_{out}[T_{out,s,v}] \neq \perp$ do
8:     cloud server computes $S_{out,s,v} = g(S_{out,s}, \omega)$.
9:     cloud server performs $(V||D_{s,v}||C_{s,v}) = \Psi_{s,v} \oplus S_{out,s,v}$.
10:    cloud server add $(V, D_{s,v}, C_{s,v})$ into $L_s$.
11:    Set $\omega = \omega + 1$.
12:    cloud server computes $T_{out,s,v} = h(T_{out,s}, \omega)$.
13: end while
14: cloud server initializes a set $L_t$ and a counter $\omega = 0$.
15: cloud server computes $T_{in,v,t} = h(T_{in,t}, \omega)$.
16: while $I_{in}[T_{in,v,t}] \neq \perp$ do
17:    cloud server computes $S_{in,v,t} = g(S_{in,t}, \omega)$.
18:    cloud server performs $(V||D_{v,t}||C_{v,t}) = \Psi_{v,t} \oplus S_{in,v,t}$.
19:    cloud server add $(V, D_{v,t}, C_{v,t})$ into $L_t$.
20:    Set $\omega = \omega + 1$.
21:    cloud server computes $T_{in,v,t} = h(T_{in,t}, \omega)$.
22: end while
23: For each encrypted vertex identifier $v$ that appears in both in $L_s$ and $L_t$, the cloud server performs the cost constraint filtering operation through Algorithm 4, and add the pair $(D_{s,v}, D_{v,t})$ which satisfies the cost constraint $\phi\theta$ into a set $Y$. The pair that Algorithm 4 cannot verify is also added into $Y$.
24: For each pair in $Y$, the cloud server first computes $d_i = \text{SWHE.Eval}(\times, D^i_{s,v}, D^i_{v,t})$, and then computes $d = \sum_{i=1}^{|Y|} d_i$.
25: cloud server returns $d$ to the user.
26: user decrypts $d$ with $sk$.
27: return Decrypted value of $d$ as $dist_q$.
```

VI. Tree-Based Ciphertexts Comparison Approach

This section introduces a tree-based ciphertexts comparison approach, which is used for cost constraint filtering in the graph encryption scheme described in Section V.

A. Scenarios

Assume that there is a *user* (i.e., $\mathcal{U}$) and a *server* (i.e., $\mathcal{R}$). $\mathcal{U}$ has many integers which are encrypted by a kind of cryptography algorithm and then outsourced to $\mathcal{R}$. Now, $\mathcal{U}$ wants to ask for $\mathcal{R}$ to obtain integer pairs, e.g., (x, y), whose sum does not exceed θ. Note that the plaintexts of x, y and θ could not be disclosed to $\mathcal{R}$, except for the greater-than, equality, or less-than relationship. A naive approach is to download all the integers, calculate the summation locally, and choose the integer pairs satisfying the constraint. This method, however, is meaningless if one wants to offload the computation to the cloud. Hence, it is desirable to have a practical solution to this problem.

Note that this scenario is different from the well-known SMC scheme. In the setting of SMC [41], [42], a set of (two or more) parties with private inputs wish to compute a function of their inputs while revealing nothing but the result of the function, which is used for many practical applications, such as exchange markets. SMC is a collaborative computing problem that solves the privacy preserving problem among a group of mutually untrusted participants. The ciphertexts of all pairs of (x, y) and the cost constraint θ are outsourced to the cloud server, which is responsible for the inequality tests. Furthermore, we could reveal the relationship between the sum of two ciphertexts and another ciphertext to the *server*, which is referred to as *controlled disclosure* in the literature [17].

It seems that we might leverage the homomorphic encryption technique, since it supports a sum operation of calculating $x + y$. Nevertheless, as the homomorphic encryption is probabilistic, we are unable to determine the relationship between $x + y$ and θ over their ciphertexts.

B. Main Idea

The main idea of the tree-based ciphertexts comparison protocol is to encode an integer with the ORE primitive. To the best of our knowledge, none of the existing approaches can support ORE and homomorphism properties simultaneously. Hence, we design a novel method to address this problem, which is motivated by the following facts.

If we want to compare $x+y$ with θ, we can compare x with $\theta/2$ and y with $\theta/2$, respectively. Now, we result in 4 possible cases corresponding to combinations of the two relationships. If $x > \theta/2$ ($x \leq \theta/2$) and $y > \theta/2$ ($y \leq \theta/2$), we can know that $x + y > \theta$ ($x + y \leq \theta$). In the rest two cases, i.e., $x > \theta/2$ and $y < \theta/2$, or $x \leq \theta/2$ and $y \geq \theta/2$, we cannot achieve a deterministic result. At this point, we can further divide $\theta/2$ into $\theta/4$. And then we can compare x and y with $\theta/4$ and $3\theta/4$, respectively.

By iteratively performing such an operation, we can determine the relationship between $x + y$ and θ with an increasing probability. Due to the ORE property, it is easy to perform the above operations over ciphertexts. Next, we will show how to implement this idea efficiently by utilizing a tree structure.

C. Details of Protocol

To implement the comparison of $x + y$ and θ over their ciphertexts, we construct a *cost constraint tree*, whose nodes represent specific values that are related to θ. For clarity, we define $E(m)$ as the ORE ciphertext of m.

An example of the tree structure is depicted in Fig. 6. For each node, we assign 0 to its left child path, while 1 to the right child path. If an integer is not greater than the value of this node, we take the left child path for further comparison; otherwise, we take the right child path. Thus, for any path from the root node to a leaf node, we can obtain a path code, which is an effective representation of the comparison procedure. For instance, an incoming integer $5\theta/16$ would traverse Nodes $E(\theta/2)$, $E(\theta/4)$, and $E(3\theta/8)$, and thereby end with a path code of 010. We define the length (i.e., the number of bits) of a path code as β. Note that β is actually equal to the depth of the tree which is denoted by d_θ.

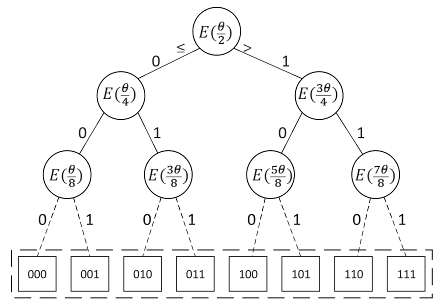


Fig. 6. An example of the cost constraint tree with a depth of 3, where circles represent nodes. The boxes in the dashed rectangle indicate path codes for all possible comparison results. Note that these boxes are not a part of the tree.

Now the relationship between $x+y$ and θ can be determined as follows. We first get the ORE ciphertexts of x and y, as well as their path codes c_x and c_y by traversing the tree separately. When computing $c_x + c_y$, if an overflow occurs (i.e., $c_x + c_y \geq 2^\beta$), we know that $x + y > \theta$ with confidence. If $c_x + c_y \leq 2^\beta - 2$, we also know that $x + y \leq \theta$ with confidence. Otherwise, we are unable to determine the relationship and end up with an *uncertainty*. We summarize this procedure in Algorithm 4.

Algorithm 4 Tree-Based Ciphertexts Comparison Algorithm

```
Input: Two ORE ciphertexts $E(x)$, $E(y)$ and a cost constraint
    tree whose depth is $d_\theta$.
Output: The relationship between $x + y$ and $\theta$.
 1: Initialize a counter $\omega = 1$ and two empty strings $c_x$ and
        $c_y$.
 2: while $\omega \leq d_\theta$ do
 3:     Visit the $\omega$-th level of the tree with $E(x)$ and concatenate
            $c_x$ with corresponding 0 or 1.
 4:     Visit the $\omega$-th level of the tree with $E(y)$ and concatenate
            $c_y$ with corresponding 0 or 1.
 5:     Set $\omega = \omega + 1$.
 6: end while
 7: if $c_x + c_y \geq 2^\omega$ then
 8:     return $>$.
 9: end if
10: if $c_x + c_y \leq 2^\omega - 2$ then
11:     return $\leq$.
12: end if
13: return uncertainty.
```

Discussion: Observe that when we go through a cost constraint tree, one more step can further reduce the uncertainty of the relationship between $x + y$ and θ by half. We denote the probability of *uncertainty* as

$$Pr[\neg certainty] = (\frac{1}{2})^{\beta}.$$

where β is the length of the path code. We can easily know the probability of certainty is

$$Pr[certainty] = 1 - Pr[\neg certainty] = 1 - (\frac{1}{2})^{\beta}.$$

When the tree depth is 6 (e.g., $\beta = 6$), the probability of certainty could reach about 0.9844.

Another observation is the comparison procedure reveals the order information between x (or y) and θ. Thus, the *server* can infer the interval that x belongs to with precision of $2^{-\beta}$. To prevent the *server* from inferring the real value of x, in Connor, the *user* randomly picks a big integer number ϕ that is applied to x, y, and θ simultaneously, which significantly enlarges the plaintext and ciphertext spaces (e.g., 2^{128}). The value of β is generally a small integer (e.g., 6 in our implementation) that is determined by the *user*, and both ϕ and θ are kept secret by the *user*. Therefore, the *server* cannot infer the real value of x (or y) from the order relationship among ciphertexts. We will formally analyze the leakage functions and security issues in the next section.

VII. Complexity and Security Analyses

This section presents the complexity and security analyses on the proposed graph encryption scheme Connor.

A. Complexity Analysis

Connor mainly consists of the *Setup* and *Query* algorithms, as described in Algorithms 2 and 3.

The dominant component in determining the complexity of the *Setup* algorithm is the encryption of the plain 2HCLI generated from a graph G. Let μ be the total sketch for all vertices in G, then the time complexity and space complexity are both $\mathcal{O}(n\mu)$, where n is the number of vertices in G.

The *Query* algorithm consists of a query token generation process on the *user* side and a CSD query process on the cloud *server* side. Let η be the maximum size of the sketch associated with each vertex in G. The complexity of the query token generation process is mainly determined by the construction of a cost constraint tree, whose time complexity and space complexity are both $\mathcal{O}(2^{d_\theta})$. For the CSD querying process, the time complexity of getting L_s and L_t, performing cost constraint filtering, and performing distance computation are $\mathcal{O}(\eta)$, $\mathcal{O}(\eta d_\theta)$, and $\mathcal{O}(\eta)$, respectively. The space complexity of the above three components are $\mathcal{O}(\eta)$, $\mathcal{O}(\eta+2^{d_\theta})$, and $\mathcal{O}(\eta)$, respectively. Therefore, the total time complexity and space complexity of the CSD querying process are $\mathcal{O}(\eta d_\theta)$ and $\mathcal{O}(\eta + 2^{d_\theta})$, respectively.

B. Security Analysis

We now present the security analysis on Connor. For clarity, we first discuss the leakage functions, and then prove that Connor is secure under the CQA2-security model.

1) Setup Leakage: The leakage function $\mathcal{L}_{Setup}$ of our construction reveals the information that can be deduced from the secure 2HCLI $\widetilde{\Delta}$ of graph G, including the total number of vertices in the graph n, the maximum distance over all the sketches

$$\mathcal{B} = max_{u \in V}\ max_{\{(v,d_{u,v},c_{u,v}) \in \Delta_{out},(v,d_{u,v},c_{u,v}) \in \Delta_{in}\}} d_{u,v},$$

and the size of $\widetilde{\Delta}$. More precisely, the size of $\widetilde{\Delta}$ consists of the total number of sketch entities in I_{out} and I_{in}, which are denoted by Ω_{out} and Ω_{in}, respectively. Thus, the leakage function $\mathcal{L}_{Setup} = (n, \mathcal{B}, \Omega_{out}, \Omega_{in})$.

Note that the order relationship of pairwise costs and the order relationship between the cost and cost constraint are not included in $\mathcal{L}_{Setup}$, because for each entity in sketches, we make an XOR operation using a unique integer value after we encrypt it, and this makes each entity in sketches are indistinguishable.

2) Query Leakage: The leakage function $\mathcal{L}_{Query}$ of our construction consists of the query pattern leakage, the sketch pattern leakage, and the cost pattern leakage. Intuitively, the query pattern leakage reveals whether a query has appeared before. The sketch pattern leakage reveals the sketch associated to a queried vertex, the common vertices between two different sketches, and the size of the sketches of queried vertices. The cost pattern leakage reveals 1) the order relationship among costs, and 2) the order relationship between costs and the cost constraint during the query procedure. We formalize these leakage functions as follows.

Definition 4 (QUERY PATTERN LEAKAGE): Let $\mathbf{q} = (q_1, q_2, \ldots, q_m)$ *be a non-empty sequence of queries. Each query* q_i *specifies a tuple* (u_i, v_i, θ_i). *For any two queries* q_i *and* q_j, *define* $Sim(q_i, q_j) = (u_i = u_j, v_i = v_j, \theta_i = \theta_j)$, *i.e., whether each element of* $q_i = (u_i, v_i, \theta_i)$ *matches each element of* $q_j = (u_j, v_j, \theta_j)$, *respectively. Then, the query pattern leakage function* $\mathcal{L}_{QP}(\mathbf{q})$ *returns an* $m \times m$ *(symmetric) matrix, in which each entry* (i, j) *equals* $Sim(q_i, q_j)$. *Note that* $\mathcal{L}_{QP}(\mathbf{q})$ *does not leak the identities of the query vertices.*

Definition 5 (SKETCH PATTERN LEAKAGE): Given a secure 2HCLI $\widetilde{\Delta}$ *of a graph G and a query* $q = (u, v, \theta)$, *the sketch pattern leakage function* $\mathcal{L}_{SP}(\widetilde{\Delta}, q)$ *is defined as* (Σ, Υ). Σ *is a list, each element of which is the sketches associated to the queried vertices, and* Υ *is a pair* (X, Z), *where* $X = h(v) : (v, d, c) \in I_{out}$ *and* $Z = h(v) : (v, d, c) \in I_{in}$ *are multi-sets and* $h : \{0, 1\}^{\lambda} \times \{0, 1\}^{*} \rightarrow \{0, 1\}^{\lambda}$ *is a particular pseudo-random function.*

Definition 6 (COST PATTERN LEAKAGE): The cost constraint θ *in a query* q *can essentially be represented by a certain number of uniform intervals. Let* d_θ *be the depth of the cost constraint tree* T_θ *(c.f. Section VI). The intervals associated with* θ *are* $[(i-1)\theta/2^{d_\theta}, i\theta/2^{d_\theta}]$, *where* $1 \leq i \leq 2^{d_\theta}$. *Assign each interval with a list* μ, *i.e., the* i*-th interval is associated with* μ_i, *which stores all the cost values belong to this interval. The leaked interval information forms an array* Arr, *of which the* i*-th element is* μ_i *(i.e.,* $Arr[i] = \mu_i$*). In addition, assume that* z *is the total number of entries in the sketches of the queried vertices. For each pair of costs* c_i *and* c_j, *its order relationship of the greater-than, equality, and less-than can be represented by* 1, 0, *and* -1, *respectively.*

The leaked order information of costs is a $z \times z$ (symmetric) matrix ∇ with each entry (i, j) being 1, 0, *or* −1. *Therefore, the cost pattern leakage function $\mathcal{L}_{CP}(\widetilde{\Delta}, q) = (Arr, \nabla)$.*

Thus, $\mathcal{L}_{Query} = (\mathcal{L}_{QP}(\boldsymbol{q}), \mathcal{L}_{SP}(\widetilde{\Delta}, q), \mathcal{L}_{CP}(\widetilde{\Delta}, q))$.

The leakage functions are defined over the 2HCLI rather than the original graph. In fact, the information leakage of the original graph is limited to the minimum number of paths for the queried source-destination vertices. It can be defined as an $n \times n$ (symmetric) matrix Λ, where n is the number of vertices in the graph. Each element in Λ is NULL, 0, or a positive integer, which indicates an uncertain status (i.e., topology is well protected), disconnection, or the minimum number of paths of the two queried vertices, respectively.

For the cost values in the 2HCLI, we introduce a *user*-held amplification factor ϕ to enlarge the plaintext and ciphertext spaces. Thus, the *server* cannot infer the real cost values just from their order information revealed by the leakage function $\mathcal{L}_{CP}(\widetilde{\Delta}, q)$). For the distance values in the 2HCLI, we use the SWHE encryption to protect their real values from the *server*.

Theorem 1: If the cryptography primitives g, h, ORE, and the SWHE are secure, then the proposed graph encryption scheme $\Pi = (KeyGen, Setup, Query)$ is $(\mathcal{L}_{Setup}, \mathcal{L}_{Query})$-secure against the adaptive chosen-query attack.

Proof: The key idea is constructing a simulator $\mathcal{S}$. Given the leakage functions $\mathcal{L}_{Setup}$ and $\mathcal{L}_{Query}$, $\mathcal{S}$ constructs a fake encrypted 2HCLI structure $\widetilde{\Delta}^* = \{I^*_{out}, I^*_{in}\}$ and a list of query q^*. If for all PPT adversaries $\mathcal{A}$, they cannot distinguish between the two games **Real** and **Ideal**, we can say that our graph encryption scheme is $(\mathcal{L}_{Setup}, \mathcal{L}_{Query})$-secure against the adaptive chosen-query attack.

3) Simulating $\widetilde{\Delta}^*$: $\mathcal{S}$ handles each vertex u_i $(1 \le i \le n)$ to generate a fake I^*_{out} in 2HCLI based on the leakage function $\mathcal{L}_{Setup}$. $\mathcal{S}$ randomly chooses w_i for u_i with $\sum_1^n w_i = \Omega_{out}$, and samples $l_i \leftarrow \{0, 1\}^\lambda$ and $\eta_i \leftarrow \{0, 1\}^\lambda$ uniformly without repetition. For all $0 \le i < w_i$, $\mathcal{S}$ takes the following steps to simulate each sketch: $\mathcal{S}$ computes $l_{w_i} = h(l_i, w_i)$ and $\eta_{w_i} = h(\eta_i, w_i)$, where h is a particular pseudo-random function. Then, it encrypts each vertex v in the sketch of u_i by computing $V^* = h(K^*, v||0)$, where K^* is a fake secret key. It randomly generates two integers d and c and obtains ciphertexts D^* and C^* by encrypting 2^{N-d} $(N = 2\mathcal{B} + 1)$ and c using the SWHE and ORE schemes. Let $\Psi^*_i = \eta_{w_i} \oplus (V^*||D^*||C^*)$. $\mathcal{S}$ stores Ψ^* in the index I^*_{out}. That is, $I^*_{out}[l_{w_i}] = \Psi^*_i$. Similarly, $\mathcal{S}$ generates a fake I^*_{in} and finally obtains the fake 2HCLI $\widetilde{\Delta}^* = \{I^*_{out}, I^*_{in}\}$.

Simulating q^*. Given the leakage function $\mathcal{L}_{Query} = (\mathcal{L}_{QP}(q), \mathcal{L}_{SP}(\widetilde{\Delta}, q), \mathcal{L}_{CP}(\widetilde{\Delta}, q))$, $\mathcal{S}$ simulates the query token as follows. $\mathcal{S}$ first checks if either of the queried vertices s and t has appeared in any previous query. If s appeared previously, $\mathcal{S}$ sets $S^*_{out,s}$ and $T^*_{out,s}$ to the values that were previously used. Otherwise, it sets $T^*_{out,s} = l_i$ and $S^*_{out,s} = \eta_i$ for some previously unused l_i and η_i. It then remembers the association among η_i, l_i, and s. $\mathcal{S}$ takes the same steps for the queried vertex t: setting $S^*_{in,t}$ and $T^*_{in,t}$ analogously and associating t with the selected η_i and l_i.

To simulate a fake cost constraint tree T^*_θ, $\mathcal{S}$ first checks if the queried θ appeared in any previous query. If θ appeared previously, $\mathcal{S}$ sets the T^*_θ to the value that was previously used. Otherwise, $\mathcal{S}$ constructs a full binary tree based θ and encrypts each tree node by using the ORE scheme with a randomly generated key. $\mathcal{S}$ returns this encrypted tree as T^*_θ.

$\mathcal{S}$ simulates the query procedure as follows. Given the query token $(S^*_{out,s}, T^*_{out,s}, S^*_{in,t}, T^*_{in,t}, T^*_\theta)$, $\mathcal{S}$ first checks if the query has been queried before. If yes, $\mathcal{S}$ returns the value that was previously used as the query result. Otherwise, $\mathcal{S}$ checks whether the queried vertex s (or t) has been queried before. If the query vertex s has appeared in a previous query, $\mathcal{S}$ sets L^*_s to the values that were previously used from Σ of $\mathcal{L}_{SP}(\widetilde{f}, q)$. Otherwise, for a newly appeared vertex s, $\mathcal{S}$ takes the following steps: To generate the sketches associated with s, $\mathcal{S}$ first initializes a set L^*_s and a counter $\omega^* = 0$, Then, it iteratively computes $T^*_{out,s,v} = h(T^*_{out,s}, w_*)$ and $S^*_{out,s,v} = g(S^*_{out,s}, w_*)$, and adds the tuple $(V^*, D^*_{s,v}, C^*_{s,v})$ into L^*_s, until $I^*_{out}[T^*_{out,s,v}]$ does not exist, where $(V^*, D^*_{s,v}, C^*_{s,v}) = I^*_{out}[T^*_{out,s,v}] \oplus S^*_{out,s,v}$. Similarly, $\mathcal{S}$ obtains the set L^*_t for vertex t. Upon obtaining L^*_s and L^*_t, $\mathcal{S}$ performs cost constraint filtering operation based on T^*_θ to get the candidate set Y^*. The theorem then follows from the CPA-security of SWHE. That is, $\mathcal{S}$ performs the SWHE computation over Y^* and returns the query result.

Since the cryptography primitives g, h, ORE, and SWHE are secure, the fake 2HCLI structure $\widetilde{\Delta}^*$ and the query sequence q^* are indistinguishable from the real ones. Therefore, for all PPT adversaries $\mathcal{A}$, they cannot distinguish between the two games **Real** and **Ideal**. Thus, we have

$$|\mathbf{Pr}[Real_{\Pi,\mathcal{A}}(\lambda) = 1] - \mathbf{Pr}[Ideal_{\Pi,\mathcal{A},\mathcal{S}}(\lambda) = 1]| \le negl(\lambda).$$

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

VIII. Performance Evaluation

This section presents the evaluation of our graph encryption scheme through experiments on real-world datasets.

A. Setup

1) Testbed: We implement the method introduced in [2] for building the 2HCLI. The ORE and SWHE in our implementation follow the methods described in [15] and [16], respectively. The GMP library is used for big integer arithmetic. We set the security parameter $\lambda = 128$ and use the OpenSSL library for all the basic cryptographic primitives. All the algorithms in our experiment are implemented in C++. The experiments are conducted on a desktop PC equipped with Intel Xeon processor at 2.6 GHz and 8 GB RAM.

2) Graph Sets: The datesets used in our experiments are listed in Table II. All these datasets are publicly available from the Standford SNAP website[3] and modeled as directed graphs. For the datasets soc-Epinions1 and Email-EuAll, we randomly select their subsets to make the index construction feasible with the limited computational resources. Since these graphs are unweighted, we generate a distance and a cost for each edge, the value of which follows a uniform distribution between 1 and 100. The cost criterion is used as the constraint.

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

TABLE II
THE GRAPH DATASETS USED IN OUR EXPERIMENTS

Dataset	Nodes	Edges	Storage
Email-EuAll	21,721	34,351	335KB
soc-Epinions1	6,506	47,062	418KB
p2p-Gnutella25	22,687	54,705	632KB
p2p-Gnutella04	10,876	39,994	422KB

3) *Methods to Compare:* Since this is the first work to address the CSD querying problem over encrypted graphs, we compare our method with the one over unencrypted graphs. We implement such a method following the state-of-the-art method over plaintext graphs introduced in [2]. The only difference is that we construct 2HCLI over the original graph, instead of an overlay graph. As a result, our implementation has a higher query efficiency but leads to a higher complexity of the index construction.

4) *Query Sets:* We randomly generate 200 queries over each dataset. The origin s and destination t in each query are also randomly selected. The cost constraint θ for each (s, t) pair is set as follows. We denote the *lower* bound c_{min} as the minimum cost of all paths from s to t, and the *upper* bound c_{max} as the minimum cost of the paths with the shortest distance from s to t. If the cost constraint $\theta < c_{min}$, there will be no feasible answer to the query; and if the cost constraint $\theta > c_{max}$, the shortest distance is always a valid answer to the query. To mitigate the impact of θ on the performance, we randomly choose 50 values of θ for each query, which falls in the interval $[c_{min}, c_{max}]$.

Another important parameter is α, which determines the approximation guarantees of α-CSD queries. Since α is a constant value for all queries, we view it as a system parameter rather than part of specific queries. In order to achieve a balance between query accuracy and system efficiency, we set the approximation ratio $\alpha = 1.5$ for all queries.

B. Evaluation of Secure 2HCLI and Query Token

1) *Index Size and Construction Time:* The index construction of the graph is a one-time and offline computation. This process consists of two steps: one is constructing the plain 2HCLI, which is the same as the index construction process of the original plain CSD querying, and the other is encrypting the plain 2HCLI, which is the focus of this paper. Therefore, we consider the outputs of the first step as the index of unencrypted graph.

The index size and construction time are depicted in Table III. Note that the index size and construction time of different datasets have a great difference, which is mainly caused by the difference in graph topologies. Different from the original shortest distance query, where there is only one shortest path between any two vertices, in the CSD querying problem, there usually exist multiple constrained shortest paths between any two vertices. Intuitively, a dense graph may bear a higher index construction cost than a sparse one.

In general, the size of each encrypted index is roughly 6× larger than that of the corresponding plain index. The most important observation is that the index construction time of

TABLE III
SUMMARY OF INDEX CONSTRUCTION COST

Metrics	Plain Graph Query		Connor	
	Time (mins)	Size (MB)	Time (mins)	Size (MB)
D1: Email-EuAll	862.03	8.99	869.84	48.14
D2: soc-Epinions1	7093.25	5.76	7098.44	32.79
D3: p2p-Gnutella25	4206.96	138.50	4306.31	514.46
D4: p2p-Gnutella04	3007.91	63.12	3054.55	297.95

TABLE IV
THE QUERY TOKEN GENERATION TIME FOR DIFFERENT d_θ

d_θ	1	2	3	4	5	6	7	8
Time (ms)	0.15	0.31	0.65	1.22	2.46	4.95	9.88	19.77

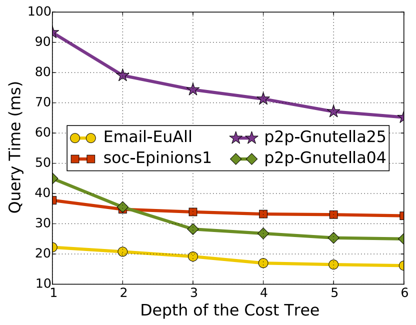


Fig. 7. The query time over encrypted 2HCLI with varying d_θ.

encrypted graphs is slightly higher than the one of unencrypted graphs. Thus, the key point of improving the index construction efficiency over an encrypted graph is accelerating the process of constructing the plain 2HCLI of that graph. We leave this attempt as the future work.

2) *Query Token Generation:* The construction of query tokens is independent of specific graphs, we now analyze the size and generation time of a query token. The query token mainly consists of 5 elements, namely $S_{out,s}$, $T_{out,s}$, $S_{in,t}$, $T_{in,t}$, and T_θ. Each of the first 4 elements has a length of 16 bytes. Since the size of each ORE ciphertext is 16 bytes, a cost tree T_θ whose depth is d_θ has a size of $16 \times (2^{d_\theta} - 1)$ bytes. Therefore, the total size of a query token is $16\times (2^{d_\theta}+3)$ bytes. Since d_θ is a relatively small value, the size of a query token is usually less than 1 KB. The query token generation time with varying d_θ is depicted in Table IV. Although the query token generation time increases significantly with d_θ, the time cost is moderate for general cases (e.g., when $d_\theta \leq 6$).

C. Evaluation of Query Efficiency and Accuracy

1) *Query Efficiency:* To evaluate the query efficiency, for each θ, we generate the cost constraint tree with a different depth d_θ. The query time is defined as the time interval from the submission of a query token to the receival of its query results. We compute the average query time of 200 queries.

The average query time with varying d_θ over the encrypted 2HCLI is depicted in Fig. 7, where d_θ increases from 1 to 6. We can see that the query time varies a lot for different graph datasets. For each dataset, increasing d_θ can result in a decrease in the query time. This is because a larger d_θ can filter out more distance pairs exceeding the cost constraint

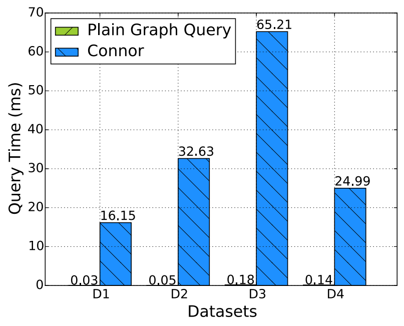


Fig. 8. The query time over the plain 2HCLI and the encrypted 2HCLI ($d_\theta = 6$).

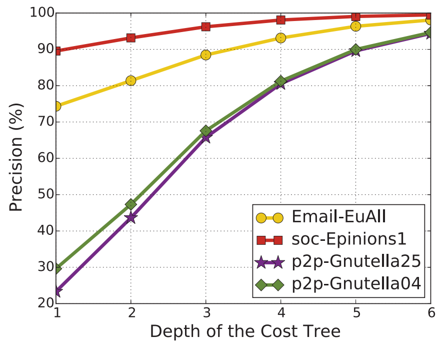


Fig. 9. The query precision for different depth d_θ of the cost tree.

and thereby reduce the number of candidates for distance computation using SWHE, which is the dominant operation in time consumption.

Fig. 8 presents the query time in the plain and encrypted scenarios for different datasets. The query time over the encrypted 2HCLI is higher than that over the plain 2HCLI because of the time-consuming operations on ciphertexts (e.g., the cost filtering and distance computation). Also, the time complexity of these operations is closely related to the size of a graph index listed in Table III, which leads to the difference among four datasets in Fig. 8.

2) Query Accuracy: In Connor, there are two components that affect the query accuracy, namely the tree-based ciphertexts comparison and the distance computation. The former may keep some distance pairs that do not satisfy the cost constraint in the candidate set Y, while the latter leverages the property of SWHE to obtain an approximate, but not exact, shortest distance based on all candidates in Y.

We use the well-known metric *Precision* ($\mathcal{P}$) to evaluate the accuracy of the cost constraint filtering process. $\mathcal{P} = \frac{T_p}{T_p+F_p}$, where T_p and F_p represent the numbers of distance pairs in Y whose costs truly satisfy or exceed the cost constraint, respectively. We use the same query as introduced above, and compute the $\mathcal{P}$ for each query. Finally, we can obtain the average precision $\bar{\mathcal{P}}$ for all the queries.

Fig. 9 presents the relationship between the query precision $\bar{\mathcal{P}}$ and the depth of the cost constraint tree d_θ over different datasets. We can see that for all the datasets, $\bar{\mathcal{P}}$ increases with d_θ, because the cost constraint tree with a larger depth d_θ helps us to detect constraint violations with a higher probability, as discussed in Section VI. In particular, $\bar{\mathcal{P}}$ is more than 94% for all datasets when $d_\theta = 6$.

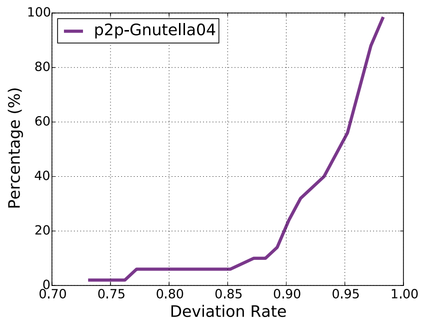


Fig. 10. The CDF of deviation rate for different query ($d_\theta = 6$).

To evaluate the accuracy of the final query results, we propose a metric named the *deviation rate*. Let r_e and r_p be the query results returned by Connor and the algorithm over the corresponding plain graphs, respectively. Then, we define the *deviation rate* $\xi = r_e/r_p$, which indicates how far r_e deviates from r_p. Obviously, a *deviation rate* closer to 1 depicts more accurate query results.

Fig. 10 presents the cumulative distribution functions (CDFs) of the *deviation rate* over the dataset p2p-Gnutella04. We can see that ξ is larger than 0.90 for over 80% of the query results, and larger than 0.73 in the worst cases. Therefore, Connor is capable of achieving a relatively high accuracy with moderate computation complexity.

IX. Conclusion

In this paper, we have presented Connor, the first graph encryption scheme that enables the cloud-based approximate CSD queries. In particular, we proposed a tree-based ciphertexts comparison protocol for cost constraint filtering with controlled disclosure. The security analysis showed that Connor could achieve the CQA2-security. We implemented a prototype and evaluated the performance using the real-world graph datasets. The evaluation results demonstrated the effectiveness of Connor. In the future work, we plan to design techniques to support dynamic index updates.

References

[1] X. Meng, S. Kamara, K. Nissim, and G. Kollios, "GRECS: Graph encryption for approximate shortest distance queries," in *Proc. ACM CCS*, New York, NY, USA, 2015, pp. 504–517.
[2] S. Wang, X. Xiao, Y. Yang, and W. Lin, "Effective indexing for approximate constrained shortest path queries on large road networks," *Proc. VLDB Endowment*, vol. 10, no. 2, pp. 61–72, 2016.
[3] M. Shen, M. Wei, L. Zhu, and M. Wang, "Classification of encrypted traffic with second-order Markov chains and application attribute bigrams," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 8, pp. 1830–1843, Aug. 2017.
[4] M. Shen, K. Xu, K. Yang, and H.-H. Chen, "Towards efficient virtual network embedding across multiple network domains," in *Proc. IEEE 22nd Int. Symp. Quality Service*, May 2014, pp. 61–70.
[5] K. Xu, M. Shen, H. Liu, J. Liu, F. Li, and T. Li, "Achieving optimal traffic engineering using a generalized routing framework," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 1, pp. 51–65, Jan. 2016.
[6] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, E. C. Guestrin, and J. M. Hellerstein, "GraphLab: A new framework for parallel machine learning," *Proc. 26th Conf. Uncertainty Artif. Intell. (UAI)*, Jun. 2010, pp. 340–349.
[7] G. Malewicz *et al.*, "Pregel: A system for large-scale graph processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 135–146, 2010.

[8] W.-S. Han *et al.*, "TurboGraph: A fast parallel graph engine handling billion-scale graphs in a single PC," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 77–85.

[9] P. Hansen, "Bicriterion path problems," in *Multiple Criteria Decision Making Theory and Application*. Berlin, Germany: Springer, 1980, pp. 109–127.

[10] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Math. Oper. Res.*, vol. 17, no. 1, pp. 36–42, 1992.

[11] G. Tsaggouris and C. Zaroliagis, "Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications," *Theory Comput. Syst.*, vol. 45, no. 1, pp. 162–186, 2009.

[12] S. Storandt, "Route planning for bicycles-exact constrained shortest paths made practical via contraction hierarchy," in *Proc. ICAPS*, vol. 4. 2012, p. 46.

[13] A. Sealfon, "Shortest paths and distances with differential privacy," in *Proc. ACM SIGMOD*, New York, NY, USA, 2016, pp. 29–41.

[14] K. Lewi and D. J. Wu, "Order-revealing encryption: New constructions, applications, and lower bounds," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2016, pp. 1167–1178.

[15] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu, "Practical order-revealing encryption with limited leakage," in *Proc. IACR-FSE*, 2016, pp. 474–493.

[16] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. TCC*, 2005, pp. 325–341.

[17] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *Proc. ASIACRYPT*, 2010, pp. 577–594.

[18] C. Chen *et al.*, "An efficient privacy-preserving ranked keyword search method," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 951–963, Apr. 2016.

[19] X. Huang and X. Du, "Achieving big data privacy via hybrid cloud," in *Proc. Int. Conf. INFOCOM*, Apr. 2014, pp. 512–517.

[20] Y. Cheng, X. Fu, X. Du, B. Luo, and M. Guizani, "A lightweight live memory forensic approach based on hardware virtualization," *Inf. Sci.*, vol. 379, pp. 23–41, Feb. 2017.

[21] L. Wu, X. Du, and J. Wu, "MobiFish: A lightweight anti-phishing scheme for mobile phones," in *Proc. Int. Conf. Comput. Commun. Netw.*, 2014, pp. 1–8.

[22] L. Wu, X. Du, and X. Fu, "Security threats to mobile multimedia applications: Camera-based attacks on mobile phones," *IEEE Commun. Mag.*, vol. 52, no. 3, pp. 80–87, Mar. 2014.

[23] X. Du, Y. Xiao, M. Guizani, and H.-H. Chen, "An effective key management scheme for heterogeneous sensor networks," *Ad Hoc Netw.*, vol. 5, no. 1, pp. 24–34, 2007.

[24] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou, "Privacy-preserving query over encrypted graph-structured data in cloud computing," in *Proc. 31st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2011, pp. 393–402.

[25] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith, "Analyzing graphs with node differential privacy," in *Theory of Cryptography*, A. Sahai, Ed. Berlin, Germany: Springer, 2013, pp. 457–476, doi: 10.1007/978-3-642-36594-2_26.

[26] E. Shen and T. Yu, "Mining frequent graph patterns with differential privacy," in *Proc. SIGKDD*, New York, NY, USA, 2013, pp. 545–553.

[27] M. Blanton, A. Steele, and M. Alisagari, "Data-oblivious graph algorithms for secure computation and outsourcing," in *Proc. 8th ACM SIGSAC Symp. Inf., Comput. Commun. Secur. (ASIA CCS)*, New York, NY, USA, 2013, pp. 207–218.

[28] A. Aly, E. Cuvelier, S. Mawet, O. Pereira, and M. V. Vyve, *Securely Solving Simple Combinatorial Graph Problems*. Berlin, Germany: Springer, 2013.

[29] M. Keller and P. Scholl, *Efficient, Oblivious Data Structures for MPC*. Berlin, Germany: Springer, 2014.

[30] D. Gupta *et al.*, "A new approach to interdomain routing based on secure multi-party computation," in *Proc. ACM Workshop Hot Topics Netw.*, 2012, pp. 37–42.

[31] F. Bayatbabolghani, M. Blanton, M. Aliasgari, and M. Goodrich. (Feb. 2017). "Secure fingerprint alignment and matching protocols." [Online]. Available: https://arxiv.org/abs/1702.03379

[32] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *SIAM J. Comput.*, vol. 32, no. 5, pp. 937–946, 2002.

[33] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proc. SIGMOD*, 2013, pp. 349–360.

[34] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, May 2000, pp. 44–55.

[35] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. ACM CCS*, New York, NY, USA, 2006, pp. 79–88.

[36] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 965–976.

[37] D. Cash *et al.*, "Dynamic searchable encryption in very-large databases: Data structures and implementation," in *Proc. NDSS*, 2014, pp. 1–32.

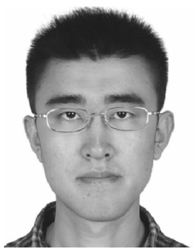
[38] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. NDSS*, 2014, pp. 23–26.

[39] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *J. Comput. Secur.*, vol. 19, no. 5, pp. 895–934, 2011.

[40] M. Naveed, S. Kamara, and V. Charles Wright, "Inference attacks on property-preserving encrypted databases," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2015, pp. 644–655.

[41] A. Ben-Efraim, Y. Lindell, and E. Omri, "Optimizing semi-honest secure multiparty computation for the Internet," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 578–590.

[42] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: A system for secure multi-party computation," in *Proc. ACM CCS*, Alexandria, VA, USA, Oct. 2008, pp. 257–266.



Meng Shen (M'14) received the B.Eng. degree in computer science from Shandong University, Jinan, China, in 2009, and the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2014. He is currently an Assistant Professor with the Beijing Institute of Technology, Beijing, China. His research interests include privacy protection of cloud-based services, network virtualization, and traffic engineering. He was a recipient of the Best Paper Runner-Up Award at IEEE IPCCC 2014.

Baoli Ma received the B.Eng. degree in computer science from Beijing Institute of Technology, Beijing, China, in 2015, where he is currently pursuing the master's degree with the School of Computer Science. His research interest is secure searchable encryption.

Liehuang Zhu is a Professor with the School of Computer Science, Beijing Institute of Technology. He is selected into the Program for New Century Excellent Talents in University from the Ministry of Education, China. His research interests include Internet of Things, cloud computing security, and internet and mobile security.

Rashid Mijumbi (M'17) received the Ph.D. degree in telecommunications engineering from the Universitat Politecnica de Catalunya (UPC), Barcelona, Spain. He was a Post-Doctoral Researcher with UPC and the Telecommunications Software and Systems Group, Waterford, Ireland, where he participated in several Spanish national, European, and Irish National Research Projects. He is currently a Software Systems Reliability Engineer with Bell Labs CTO, Nokia, Dublin, Ireland. His current research focus is on various aspects of 5G, Network Functions Virtualization, and Software Defined Networking systems. He was a recipient of the 2016 IEEE Transactions Outstanding Reviewer Award recognizing outstanding contributions to the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT.

Xiaojiang Du (SM'09) received the B.S. and M.S. degrees from Tsinghua University, Beijing, China, in 1996 and 1998, respectively, and the M.S. and Ph.D. degrees from the University of Maryland at College Park, in 2002 and 2003, respectively, all in electrical engineering. He is currently a tenured Professor with the Department of Computer and Information Sciences, Temple University, Philadelphia, USA. His research interests are wireless communications, wireless networks, security, and systems. He has authored over 200 journal and conference papers in these areas, as well as a book published by Springer. He is a Life Member of the ACM. He received over $5 million U.S. dollars research grants from the U.S. National Science Foundation, Army Research Office, Air Force, NASA, the State of Pennsylvania, and Amazon. He was a recipient of the Best Paper Award at IEEE GLOBECOM 2014 and the Best Poster Runner-Up Award at ACM MobiHoc 2014. He serves on the editorial boards of three international journals.

Jiankun Hu is a Professor with the School of Engineering and IT, University of New South Wales, Canberra, Australia. He is an invited expert of Australia Attorney-Generals Office assisting the draft of Australia National Identity Management Policy. He has served at the Panel on Mathematics, Information and Computing Sciences, Australian Research Council ERA (The Excellence in Research for Australia) Evaluation Committee 2012. His research interest is in the field of cyber security covering intrusion detection, sensor key management, and biometrics authentication. He has many publications in top venues including IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, *Pattern Recognition*, and IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS. He is an Associate Editor of IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY.