Graph Encryption for Shortest Path Queries with k Unsorted Nodes

Li, Meng; Gao, Jianbo; Zhang, Zijian; Fu, Chaoping; Lal, Chhagan; Conti, Mauro

DOI
10.1109/TrustCom56396.2022.00023

Publication date
2022

Document Version
Final published version

Published in
Proceedings - 2022 IEEE 21st International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2022

2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) | 978-1-6654-9425-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/TrustCom56396.2022.00023

Graph Encryption for Shortest Path Queries with k Unsorted Nodes

Meng Li
Hefei University of Technology
Hefei, China
mengli@hfut.edu.cn

Jianbo Gao
Hefei University of Technology
Hefei, China
jianbogao@mail.hfut.edu.cn

Zijian Zhang*
Beijing Institute of Technology
Beijing, China
zhangzijian@bit.edu.cn

Chaoping Fu
Huaqiao University
Xiamen, China
fuchp@hqu.edu.cn

Chhagan Lal
TU Delft
Delft, Netherland
c.lal@tudelft.nl

Mauro Conti
University of Padua
Padua, Italy
conti@math.unipd.it

***Abstract*—Shortest distance queries over large-scale graphs bring great benefits to various applications, i.e., save planning time and travelling expenses. To protect the sensitive nodes and edges in the graph, a user outsources an encrypted graph to an untrusted server without losing the query ability. However, no prior work has considered the user requirement of *the shortest path with k unsorted nodes*. In particular, we are concerned with how to securely find the shortest path by passing k nodes that do not have a fixed traverse order. To solve the problems, we propose Gespun (stands for Graph encryption for shortest path queries with k unordered nodes). It includes an oracle encryption scheme that is provably secure against the semi-honest server. Specifically, we compute the shortest paths and distances for all nodes locally to obtain path-distance oracles. We transform the shortest paths to a sequence of secure codes by using a pseudo-random permutation to protect the *structure privacy*. We encrypt the shortest distance by using additively homomorphic encryption. Second, we pack the oracles in link-list nodes and store them in an array-based dictionary after another permutation. Next, we construct a search graph to compute the shortest path while guaranteeing that the path passes the required k nodes. We formally prove that Gespun is adaptively semantically-secure in the random oracle. We implement a prototype of Gespun and evaluate its performance. Experiments results demonstrate that Gespun is efficient, e.g., a query over 6301 nodes, 20777 edges, and 5 unsorted nodes only needs 483 ms to get queried results. We believe that our research problem span new research that soon promotes a new line of graph encryption schemes.**

***Index Terms*—Graph encryption, Shortest distance query, Unsorted nodes, Security**

I. Introduction

A. Background

Graphs databases are utilized in various applications, including world-wide web, online social networks, and communication networks. For instance, LinkedIn has a social graph consisting of more than 8.1 million users [1]. These applications facilitate querying and analyzing large-scale graphs in an efficient way.

Zijian Zhang is the corresponding author.

With the ubiquity of cloud computing [2], [3], users, including companies and individuals, are encouraged to outsource their graph databases to a remote cloud server. By doing so, their local management costs are reduced [4]–[6]. However, outsourcing graph databases containing sensitive information (e.g., how users are connected in a social network and which roads users visit) to the untrusted cloud server will not only lead to users' loss of data control, but also raise their security concerns [7]–[11]. The concerns promote new methods to safeguard security. Data encryption is a straightforward method, but complicate data analysis.

B. Related Work

To solve the problem above, Chase and Kamara proposed the idea of structured encryption [12]. Generally speaking, a graph encryption scheme encrypts a data structure while allowing users to query it privately. Graph encryption is a special case of structured encryption and there are several prominent works. Meng et al. [13] proposed a graph encryption scheme GRECS to answer shortest distance queries approximately by computing encrypted distance oracles and using somewhat homomorphic encryption [14], Pseudo-Random Permutation (PRP), and Pseudo-Random Function (PRF). Wu et al. [15] presented a multi-round navigation protocol on city streets to enable shortest path queries. They compressed the next-hop matrices for road networks to realize fully private shortest-path computation in road networks and combined garbled circuits [16] to obtain a fully-private navigation protocol. Wang et al. [17] designed SecGDB to answer shortest distance queries over an encrypted graph database by leveraging additively homomorphic encryption [18], PRP, and PRF. SecGDB executes the Dijkstra's algorithm [19] with a Fibonacci heap to compute the shortest path. Ghosh et al. [20] proposed the first graph encryption scheme for single-pair shortest path queries based on SP-matrix M [21]. Given two nodes v_i and v_j, they used a recursive algorithm to

2324-9013/22/$31.00 ©2022 IEEE
DOI 10.1109/TrustCom56396.2022.00023

look up the first node on the shortest path $v_o = M[i, j]$ and recurses on the pair (v_o, v_j). Shortest path queries are one fundamental function of many graph algorithms [15], [17], [20], [22] and they are quite practical in daily lives. Example applications among location-based services [23], [24] include smart parking [25], navigation [26], and user matching in ride-hailing [27]. Some commercial systems include OrientDB [28], Neo4j [29], and Titan [30]. In this work, we focus on the shortest path that has a more practical use.

C. Motivation

Our motivation arises from designing a graph encryption scheme that supports an important graph operation: *finding the shortest path between two nodes while passing k unsorted nodes*. Such an operation echoes a common user requirement in real-world applications. For example, as illustrated in Fig. 1, a user Alice opens the Google Maps app on her smartphone to (1) find the shortest path from a source (green node) to a destination (red node), and (2) this path has to pass through three other locations (blue nodes), i.e., her workplace, a specific gym, and a specific pizza shop. Meanwhile, Alice does not require the order of visiting the three locations. We call them unsorted nodes. Besides the three unsorted nodes, there are also normal nodes (black nodes) along the shortest path, e.g., crossroads.

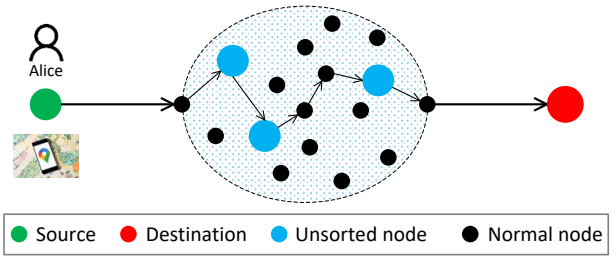


Fig. 1. Motivation: Find the Shortest Path with 3 Unsorted Nodes.

D. Technical Challenges and Proposed Solutions

To contribute to the graph encryption and break through the limitations of existing work, we propose Gespun: Graph encryption for shortest path queries with k unordered nodes. Specifically, our construction addresses three technical challenges. First, how to design an appropriate oracle that lays the foundation of our shortest path queries under restricted conditions. Existing work either neglect the shortest path [13] or only consider the neighboring information [17]. Intuitively, a path-distance oracle is preferable. Second, how to protect *structure privacy* when designing the abovementioned oracle? On one hand, we need to prepare the novel oracle to compute the shortest path. On the other hand, providing too much path information will result in violation on the graph structure, i.e., how nodes are connected to each other. Third, how to answer the shortest path queries with k unsorted nodes? Unlike the classic Travelling Salesman Problem (TSP) [31], [32], which is NP-hard, or the Könisberg bridge problem [33], our requirement is a general case of TSP (which has the same source and destination) and it should be solved in a secure manner.

To tackle the challenges, first we compute the shortest paths and distances for all nodes locally by using the Floyd-Warshall algorithm [34] on the original graph to obtain path-distance oracles. We encrypt the shortest distance by using additively homomorphic encryption. Second, we transform the shortest paths to a sequence of secure codes by using a Pseudo-Random Permutation (PRP) and n secrets to protect the structure privacy. Third, for each node, we store their oracles in linked-list nodes, which are permutated in an array. The first nodes of all arrays are then permuted by using the PRP and a different secret key in a dictionary as separate entries. We construct a search graph based on a source, a destination, and k unsorted nodes to find the shortest path over the encrypted graph while guaranteeing that the path passes the k nodes. Briefly, our main contributions are summarized as follows.

- To the best of our knowledge, this is the first work to address the shortest path queries with k unsorted nodes for graph encryption and we propose a novel graph encryption scheme named Gespun.
- We achieve the new requirement under the graph encryption framework by designing a path-distance oracle, leveraging homomorphic encryption and permutation, and constructing a search graph.
- We formally define security of Gespun and prove that it is adaptively semantically-secure in the random oracle. We develop a prototype of Gespun and conduct extensive experiments to demonstrate its efficiency and practicability.

E. Paper Organization

The remainder of this paper is organized as follows. Section 2 gives the problem formulation including system model, threat model, and design goals. Section 3 revisits some preliminaries. In section 4, we present the details of Gespun. In section 5, we formally prove the security of Gespun. Section 6 conducts the performance evaluation. Finally, we conclude this paper in Section 7.

II. Problem Formulation

In this work, we consider the problem of designing a graph encryption scheme that supports the shortest path queries with k unsorted nodes over an encrypted graph stored on a remote cloud server. Before we dive into the technical details, we elaborate on the system model and security model in this section.

A. System Model

At a high level, as shown in Fig. 2, our system contains two entities, namely the user U and the cloud server CS.

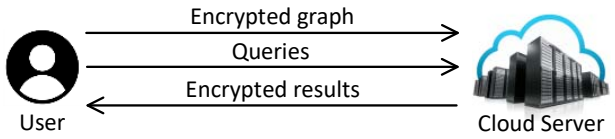


Fig. 2. System Model of Gespun.

In the Setup stage, the user U processes the original graph $\mathcal{G}$ to obtain path-distance oracles of nodes and an encrypted graph $\mathcal{EG}$. U outsources the $\mathcal{EG}$ to the cloud server CS. Next, holding a shortest path query q with k unsorted nodes over the $\mathcal{EG}$, U generates a query token t_q based on q and sends it to the CS. The CS searches $\mathcal{EG}$ by using t_q and returns an encrypted shortest path sp along with the shortest distance sd to the U. We list key notations of Gespun in Table I.

Formally, the core functionalities of our scheme are listed as below.

Definition 1 (Oracle Encryption): An oracle encryption scheme $\Pi = (\mathsf{Setup}, \mathsf{PathQuery}, \mathsf{Dec})$ supporting the shortest path query with k unsorted nodes consists of the following three probabilistic polynomial-time algorithms:

- $\mathsf{Setup}(1^\lambda, \mathcal{G}) \rightarrow (sk, \mathcal{EG})$: is a probabilistic key generation algorithm executed by the user. It takes a security parameter λ and a graph $\mathcal{G}$ as input, and outputs a secret key sk and an encrypted graph $\mathcal{EG}$.
- $\mathsf{PathQuery}(sk, q; \mathcal{EG}) \rightarrow \mathcal{R}$: is a two-party protocol between the user and the cloud server. The user takes a secret key sk and a shortest path query q as input. The cloud server takes a query token t_q and an encrypted graph $\mathcal{EG}$ as input. During the protocol, the user computes a query token t_q based on q and sends it to the cloud server. After the protocol, the user receives outputs a result set $\mathcal{R}$.
- $\mathsf{Dec}(sk, \mathcal{R}) \rightarrow (sp, sd)$: is a deterministic algorithm executed by the user. It takes a secret key sk and an encrypted result set $\mathcal{R}$, and outputs the shortest path sp as well as its shortest distance sd.

B. Security Model

At a high level, the security protection we expect from an oracle encryption scheme is that: (1) given an encrypted oracle, no adversary can acquire any useful information about the underlying oracle; and (2) given the PathQuery execution of a polynomial number of adaptively chosen shortest path queries $Q = (q_1, q_2, \cdots, q_o)$, no adversary can acquire any useful information about $\mathcal{G}$ or Q. We adapt the notion of adaptive semantic security [4], [12], [17], [20], [35] to our distance oracle encryption.

Definition 2 (Adaptive Semantic Security): Let $\Pi = (\mathsf{Setup}, \mathsf{PathQuery}, \mathsf{Dec})$ be an oracle encryption scheme and consider the following two probabilistic experiments with a semi-honest adversary $\mathcal{A}$, a challenger $\mathcal{C}$, a simulator $\mathcal{S}$, and two (stateful) leakage functions $\mathcal{L}_1$, and $\mathcal{L}_2$:

Real$_{\mathcal{A}}(1^\lambda)$:

- $\mathcal{A}$ outputs an orcale $\mathcal{G}$.
- $\mathcal{C}$ calculates $(sk, \mathcal{EG}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{G})$ and sends $\mathcal{EG}$ to $\mathcal{A}$.
- $\mathcal{A}$ generates a polynomial number of adaptively chosen shortest path queries $\mathcal{Q} = (q_1, q_2, \cdots, q_o)$. For each query q_i $(1 \leq i \leq o)$, $\mathcal{A}$ and $\mathcal{C}$ execute $\mathsf{PathQuery}(sk, q_i; \mathcal{EG})$. $\mathcal{C}$ acts as a user and $\mathcal{A}$ acts as a cloud server.
- $\mathcal{A}$ computes a bit b that is output by the experiment.

IDeal$_{\mathcal{A},\mathcal{S}}(1^\lambda)$:

- $\mathcal{A}$ outputs an oracle $\mathcal{G}$.
- Given $\mathcal{L}_1$, $\mathcal{S}$ sends an encrypted graph $\mathcal{EG}$ to $\mathcal{A}$.
- $\mathcal{A}$ generates a polynomial number of adaptively chosen shortest path queries $\mathcal{Q} = (q_1, q_2, \cdots, q_o)$. For each q_i $(1 \leq i \leq o)$, $\mathcal{S}$ is given $\mathcal{L}_2$. $\mathcal{A}$ and $\mathcal{S}$ execute a simulation of PathQuery. $\mathcal{S}$ acts as a user and $\mathcal{A}$ acts as a cloud server.
- $\mathcal{A}$ computes a bit b that is output by the experiment.

We say that Π is adaptively $(\mathcal{L}_1, \mathcal{L}_2)$- semantically secure if for all Probabilistic Polynomial-Time (PPT) adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that

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

III. Preliminaries

In this section, we revisit some preliminaries that lay the foundation for the proposed Gespun, namely homomorphic encryption, pseudo-random function, and pseudo-random permutation.

A. Homomorphic Encryption

A public-key encryption scheme $\Omega = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is homomorphic if it has an evaluation algorithm Eval that takes a function E and a set of ciphertexts $(c_1, c_2, \cdots, c_n)$ where $c_i = \mathsf{Enc}_{pk}(m_i)$ as input, and outputs a ciphertext c such that $\mathsf{Dec}_{sk}(c) = E(m_1, m_2, \cdots, m_n)$. If E supports addition: $\mathsf{Enc}_{pk}(m_1 + m_2) = \mathsf{Eval}(+, \mathsf{Enc}_{pk}(m_1), \mathsf{Enc}_{pk}(m_2))$, Ω is called additive homomorphic encryption. Concrete instantiations of homomorphic encryption schemes include BGN [14] and Paillier cryptosystem [18].

B. Pseudo-Random Function and Pseudo-Random Permutation

Let $F : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ be a PRF, which is a polynomial-time computable function. The output of a PRF cannot be distinguished from the output of a random function by any PPT distinguisher, i.e., $|\Pr[D^{F_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1]| \leq \mathrm{negl}(\lambda)$, where a key k is chosen uniformly from $\{0,1\}^n$. In other words, D is given access to an oracle $\mathcal{O}$ which is either F or f. D can query $\mathcal{O}$ at any point i to receive $\mathcal{O}(i)$. For every D that receives a description of F outputs 1 with almost the same probability as when it receives a description of a random function [36]. A PRF is assumed to be a PRP when it is bijective. We refer the interested readers to [36] for detailed information.

TABLE I
KEY NOTATIONS OF GESPUN

Notation	Definition	Notation	Definition
U	User	$\mathcal{V}$	Node set
CS	Cloud server	$\mathcal{E}$	Edge set
$\mathcal{G}$	Graph	v	Node
$\mathcal{EG}$	Encrypted graph	w	Edge weight
s	Source	n	Number of nodes
d	Destination	$\mathcal{PD}$	Path-distance oracle
q	Query	π	Permutation
q_t	Query token	Arr	Array
sp	Shortest path	DX	Dictionary
sd	Shortest distance	Esp	Encrypted shortest path
k	Number of unsorted nodes	Esd	Encrypted shortest distance
v	Node	pq	Permuted query
K, sk	Secret key	$\mathcal{L}_1, \mathcal{L}_2$	Leakage function
$\mathcal{R}$	Encrypted result set	QP, OP	Query/oracle pattern

IV. PROPOSED SCHEME

In this section, we present our proposed graph encryption scheme Gespun that supports the private shortest path query with k unsorted nodes.

A. Overview

The user has an original graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$. During the setup phase, the user computes a path-distance oracle for each node in $\mathcal{V}$. The path-distance oracle of a node v contains the shortest path from v to other nodes as well as its shortest distance. The shortest path is a sequence of nodes starting from a source s and leading to a destination d. We order the shortest path via PRP and adopt a public-key homomorphic encryption scheme to encrypt the shortest distance of each shortest path. All the path-distance oracles are further permuted and stored in an array for the cloud server to search. In the shortest path query phase, we build a search graph to look through all possible shortest pathes from the source to the destination while passing the k unsorted nodes.

B. Setup

Our scheme $\Pi = (\mathsf{Setup}, \mathsf{PathQuery}, \mathsf{Dec})$ makes use of a public-key homomorphic encryption scheme $\Omega = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, a PRP P, a PRF F, a random oracle H, and a collision-resistant hash function h. We use the Paillier cryptosystem as Ω and generate a key pair $(sk_u, pk_u) \leftarrow \{0,1\}^\lambda$. P is defined as $\{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^*$ and F is defined as $\{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$. H is defined as $H : \{0,1\}^* \rightarrow \{0,1\}^*$. A collision-resistant hash function h is modeled as a random function [13]. An original graph is defined as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ is the node set and $\mathcal{E}$ is a set of triads. For example, (v_i, v_j, w_{ij}) indicates that there is an edge from node v_i to v_j and the edge weight is w_{ij}. Each edge $(v_i, v_j, w_{ij}) \in \mathcal{E}$ linking a node $v_i \in \mathcal{V}$ to a node $v_j \in \mathcal{V}$ is associated with a non-negative weight $w(v_i, v_j)$ The user U executes the Setup algorithm that works as follows.

- U generates two secret keys K_1, K_2, and a key pair (sk_u, pk_u). U sets $sk = (K_1, K_2, sk_u, pk_u)$.
- U computes the shortest paths and distances for all nodes by using the Floyd-Warshall algorithm [34] on $\mathcal{G}$ to obtain path-distance oracles $\{\mathcal{PD}_{v_i}\}$. For example, as portrayed in Fig. 3, there is a shortest path sp_{v_i,v_j} from node v_i to node v_j and the shortest distance is sd_{v_i,v_j}. $\mathcal{PD}_{v_i}^{v_j} = (v_i, v_j, sd_{v_iv_j}, v_{ij}^1, v_{ij}^2, \cdots, v_{ij}^{(ij)})$ and $(v_{ij}^1, v_{ij}^2, \cdots, v_{ij}^{(ij)})$ is the sequence of nodes that sp_{v_i,v_j} traverses by from v_i to v_j. Then, $\mathcal{PD}_{v_i} = \{\mathcal{PD}_{v_i}^{v_j}\}_{(v_j \in \mathcal{V}, v_j \neq v_i)}$.
- U samples a random permutation π over $[\sum_{v_i} |\mathcal{PD}_i|]$, i.e., the total number of path-distance oracles, creates a counter $num = 1$ for π, and initializes an array Arr of size $\sum_{v_i \in \mathcal{V}} |\mathcal{PD}_{v_i}|$.
- U encrypts the path-distance oracle for each $\mathcal{PD}_{v_i}$ as follows.
  - U chooses two random secrets $K_{v_i}, K_{v_i}^*$ for $\mathcal{PD}_{v_i}$ $(1 \leq i \leq n)$. For each node v_j $(j \neq i)$, U creates a linked-list node $N_o = h(v_j)||\mathsf{Enc}_{pk_{v_i}}(sd_{v_i,v_j})||P_{K_{v_i}^*}(v_{ij}^1)||P_{K_{v_i}^*}(v_{ij}^2)||\cdots||P_{K_{v_i}^*}(v_{ij}^{(ij)})||\pi(num+1)$ chooses a random number r_o, and stores $(N_o \oplus H(K_{v_i}||r_o), r_o)$ at location $Arr[\pi(num)]$. For the last pair, U sets the add part as NULL and increments num. We denote $P_{K_{v_i}^*}(v_{ij}^1)||P_{K_{v_i}^*}(v_{ij}^2)||\cdots||P_{K_{v_i}^*}(v_{ij}^{(ij)})$ by Esp_{v_i,v_j}. The reason that we use n secrets $\{K_{v_i}\}_{v_i \in \mathcal{V}}$ and a different key K_1 is because we deliberately permute each path sequence Esp for each node to look different from the permutation in the dictionary explained later. By doing so, we can protect the structure privacy.
  - U creates a dictionary DX to store the pairs $(P_{K_2}(v_i), add_{v_i}^1||K_{v_i} \oplus F_{K_1})$ for all $v_i \in \mathcal{V}$, where $add_{v_i}^1$ is the location in Arr of the head of v_i's linked-list.
- U sends the encrypted graph $\mathcal{EG} = (DX, Arr)$ to the CS.

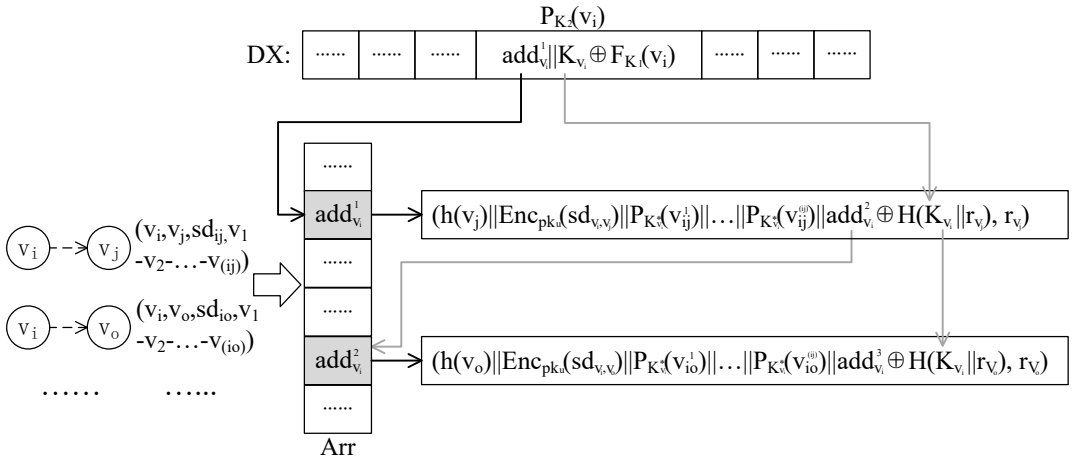


Fig. 3. An Example of Encrypting and Storing $\mathcal{PD}_{v_i}$.

C. Shortest Path Query

There are two types of queries that we should consider. The first one is the normal query where a user does not need to have a specific requirement on the nodes between the source s and the destination d. The second type is the special query where the user has k unsorted nodes to pass through between s and d.

1) Query with no unsorted nodes.: Given s and d, U generates a query token q as follows.

• U computes $q_1 = P_{K_2}(s)$, $q_2 = F_{K_1}(s)$, and $q_3 = h(d)$.

• U sends the query token $q = (q_1, q_2, q_3)$ to the CS.

Given q, CS finds the shortest path and distance for U as follows.

• CS uses retrieves $\alpha = DX[q_1]$ and computes $\beta = \alpha \oplus q_2 = (\gamma_1 || K_s)$.

• CS recovers the lists pointed to by add_s^1. Specifically, starting with $i = 1$ it parses $Arr[\gamma_1]$ as (x_s, y_s). CS decrypts x_s by computing $(h_i||sd_i||sp_i||add_{i+1}) = x_s \oplus H(K_s||r_s)$.

• If $h_i \neq q_3$, CS continues to retrieve the value in the next location by using add_{i+1} until a match is found. Finally, CS returns the query result set $\mathcal{R} = (\mathsf{Enc}_{pk_u}(sd_{s,d}), Esp_{s,d})$ to U. If add is NULL, then there is no path between s and d, CS returns NULL.

Upon receiving $\mathcal{R}$, U decrypts $\mathsf{Enc}_{pk_u}(sd_{s,d})$ by using sk_u to obtain the shortest distance $sd_{s,d}$ and checks the local record to recover the shortest path $sp_{s,d}$ by using Esp.

2) Query with k unsorted nodes.: Given s, d, and k unsorted nodes $(V_1, V_2, \cdots, V_k)$, U generates a query token q as follows.

• U computes $q_1 = (P_{K_2}(s), P_{K_2}(V_1), P_{K_2}(V_2), \cdot \cdot \cdot, P_{K_2}(V_k))$, $q_2 = (F_{K_1}(s), F_{K_1}(V_1), F_{K_1}(V_2), \cdot \cdot \cdot, F_{K_1}(V_k))$, $q_3 = (h(V_1), h(V_2), \cdots, h(V_k), h(d))$.

• U sends the query token $q = (q_1, q_2, q_3)$ to the CS.

Given q, CS finds the shortest path and distance for U as follows.

• CS initiates a shortest path $\mathcal{SP}$ and a shortest distance $\mathcal{SD} = \{Rsd_1, Rds_2, \cdots, Rsd_{k!}\} = \{1, 1, \cdots, 1\}$.

• Given q, CS computes the $k!$ permuted queries $(pq_1, pq_2, \cdots, pq_{k!})$ according to the full permutation of the k unsorted nodes.

• For each $pq_i = (pq_i^1, pq_i^2, pq_i^3), 1 \leq i \leq k!$, CS computes the encrypted shortest path Esp_i and the encrypted shortest distance Esd_i, and updated $\mathcal{SP}$ and $\mathcal{SD}$.

• We take the initial query pq_1, which is also one of the permuted queries, as an example. CS initializes $Esp_1 = \{\}, Esd_1 = 1$, and obtains Esp and Esd by invoking the search process of the normal query on $(P_{K_2}(s), F_{K_1}(s), h(V_1))$. Next, CS appends $P_{K_2}(V_1)$ to Rsp_1 and computes $Rsd_1 = Rsd_1 * Esd$. CS continues the search until the last sub-path $V_k - d$ is processed. Till here, the first path $s - V_1 - V_2 - \cdots - V_k - d$ is complete, CS inserts Rsp_1 to $\mathcal{SP}$ and inserts Esd_1 to $\mathcal{SD}$. CS continues to process the remaining paths to update $\mathcal{SP}$ and $\mathcal{SD}$.

• CS returns $\mathcal{R} = \{\mathcal{SP}, \mathcal{SD}\}$ to U.

Upon receiving $\mathcal{R}$, U decrypts each Esd_i to obtain the shortest distance $sd_{s,d}$ and checks the local record to recover the shortest path $sp_{s,d}$ by using Esp.

V. Security Analysis

The graph encryption scheme leaks information to the cloud server, which we allow to trade for efficiency. Next, we provide a formal description of the two leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$ as below.

• $\mathcal{L}_1(\mathcal{G})$: Given a graph $\mathcal{G}$, $\mathcal{L}_1(\mathcal{G}) = (n, SD_{max}, SP_{max}, PD_{sum})$, where $SP_{max} = \max_{v_i \in \mathcal{V}} \{\max_{(v_i, v_j, sd_{v_i v_j}, v_{ij}^1, v_{ij}^2, \cdots, v_{ij}^{(ij)}) \in \mathcal{PD}_{v_i}} |sp_{v_i v_j}|\}$, $SD_{max} = \max_{v_i \in \mathcal{V}} \{\max_{(v_i, v_j, sd_{v_i v_j}, v_{ij}^1, v_{ij}^2, \cdots, v_{ij}^{(ij)}) \in \mathcal{PD}_{v_i}} sd_{v_i v_j}\}$, and $PD_{sum} = \sum_{v \in \mathcal{V}} |\mathcal{PD}_v|$. • $\mathcal{L}_2(\mathcal{EG}, q)$: Given an encrypted graph $\mathcal{EG}$ and a shortest path query q,

$\mathcal{L}_2(\mathcal{G}, q) = (QP(\mathcal{EG}, q), QP(\mathcal{EG}, q))$, where $QP(\mathcal{EG}, q)$ is the query pattern and $OP(\mathcal{EG}, q)$ is the oracle pattern (similar to sketch pattern [13]).

Definition 3 (Query Pattern): The query pattern reveals whether the nodes in the query have appeared before. For two queries q_0, q_1, define $\mathsf{sim}(q_0, q_1) = (s_0 = s_1, d_0 = d_1, V_{01} = V_{11}, V_{02} = V_{12}, \cdots, V_{0k} = V_{1k})$, i.e., whether each of the nodes $q = (s_0, d_0, V_{01}, V_{02}, \cdots, V_{0k})$ matches each of the nodes of $q = (s_1, d_1, V_{11}, V_{12}, \cdots, V_{1k})$. Let $\mathbf{q} = (q_1, q_2, \cdots, q_z)$ be a non-empty sequence of queries. $\mathcal{L}_1(\mathbf{q})$ outputs a $z \times z$ matrix Z with $Z_{ij} = \mathsf{sim}(q_i, q_j)$.

Definition 4 (Oracle Pattern): For an encrypted graph $\mathcal{EG}$ and a shortest path query q, the oracle pattern is defined as $\mathcal{L}_2(\mathcal{EG}, q) = \{id(\mathcal{R})\}$, where $id(\mathcal{R})$ is the identifiers of nodes in the encrypted result set $\mathcal{R}$.

Theorem 1: If P and F are pseudo-random, and Ω is CPA-secure, Gespun is adaptively $(\mathcal{L}_1, \mathcal{L}_2)$-semantically secure in the random oracle model, where $\mathcal{L}_1(\mathcal{G}) = (n, SP_{max}, SD_{max}, PD_{sum})$ and $\mathcal{L}_2(\mathcal{EG}, g) = (QP(\mathcal{EG}, q),\ OP(\mathcal{EG},\ q))$.

Now we construct a simulator $\mathcal{S}$ as below.

- Given $\mathcal{L}_1(\mathcal{G}) = (n, SP_{max}, SD_{max}, PD_{sum})$, for all $1 \leq i \leq PD_{sum}$, $\mathcal{S}$ samples $\delta_i \xleftarrow{\$} \{0,1\}^{|h|+|\mathsf{Enc}|+rand|P|+\log PD_{sum}+\lambda}$. Here $|h|$, $|\mathsf{Enc}|$, $rand|P|$, $\log PD_{sum}$, and λ correspond to the bit length of the random function, ciphertext, nodes on the shortest path (excluding s and d), number of all oracles, and security parameter, respectively. $rand$ is chosen from $[0, SP_{max}]$ randomly, where 0 refers to a direct and shortest path from s to d. The shortest distance is chosen from $(0, SD_{max}]$. $\mathcal{S}$ stores $\{\delta_i\}$ in an array Arr^* of size PD_{sum}. For $1 \leq i \leq n$, $\mathcal{S}$ samples $dx_i \xleftarrow{\$} \{0,1\}^{\log n}$ and sets $DX^*[dx_i] \xleftarrow{\$} \{0,1\}^{\log PD_{sum}+\lambda}$. Finally, $\mathcal{S}$ outputs $\mathcal{EG}^* = (DX^*, Arr^*)$.
- Given $\mathcal{L}_2(\mathcal{EG}, g) = (QP(\mathcal{EG}, q),\ QP(\mathcal{EG},\ q))$, $\mathcal{S}$ checks whether q has been queries before. If so, $\mathcal{S}$ returns the previously token; otherwise, $\mathcal{S}$ samples an non-negative integer k in PPT.
  - If $k = 0$, $\mathcal{S}$ sets $q_1^* = dx_i$, samples an unused $add \in PD_{sum}$ and a key $K_s \xleftarrow{\$} \{0,1\}^{\lambda}$, sets $q_2^* = DX[dx_i] \oplus add || K_s$, and generates a random value from $\{0,1\}^{|h|}$ as q_3^*. $\mathcal{S}$ records (dx_i, add, K_s). To simulate H, $\mathcal{S}$ checks whether (1) K has been queried in H and (2) if any location in Arr stores (N', r) where N' is a $|h| + |\mathsf{Enc}| + rand|P| + \log PD_{sum}$-bit string. If K is new, $\mathcal{S}$ initiates $num = 0$. If there is an appropriate entry (N', r) in Arr, $\mathcal{S}$ outputs $N' \oplus (e, \mathsf{Enc}_{pk_u}(0), add)$, where e is the num^{th} element of $\pi(PD_{sum})$ and add is a random and unused address in Arr or NULL if $num = |PD|$, where $|PD|$ is the path-distance oracle size associated with K.
  - If $k > 0$, $\mathcal{S}$ simulates the query token q^* similar to the case of $k = 0$ where $q = (q_1, q_2, q_3)$. In specific, there are $k+1$ items in q_i $(1 \leq i \leq 3)$ and $\mathcal{S}$ processes each triad (q_{1j}, q_{2j}, q_{3j}) $(1 \leq j \leq k+1)$, separately. Given (q_{1j}, q_{2j}, q_{3j}), $\mathcal{S}$ considers it as a individual normal query that asks a shortest path from s under q_{1j} to d under q_{3j}. By following the operations in the above step, the $\mathcal{S}$ acquires a simulated query $(q_{1j}^*, q_{2j}^*, q_{3j}^*)$. After processing all $q_i (1 \leq i \leq k+1)$, $\mathcal{S}$ has the complete simulated query token $q^* = (q_1^*, q_2^*, q_3^*) = ((q_{11}^*, q_{12}^*, \cdots, q_{1k+1}^*), (q_{21}^*, q_{22}^*, \cdots, q_{2k+1}^*), (q_{31}^*, q_{32}^*, \cdots, q_{3k+1}^*))$.

The simulated view and the real view are indistinguishable by $\mathcal{A}$ such that Gespun is adaptively $(\mathcal{L}_1, \mathcal{L}_2)$-semantically secure in the random oracle model against an adaptive adversary, i.e., we deduce the correctness of Theorem 1 from the pseudo-randomness of P and F and the CPA-security of Ω. We complete the proof. □

Regarding the structure privacy, we use n different secrets in the path permutations for n nodes and a different secret key K_1 in the dictionary permutation. By doing so, the cloud server cannot differentiate (1) the nodes on the shortest path for a node from the nodes on the shortest path of any other nodes and (2) the nodes in DX from the nodes in the Arr. Therefore, the structure privacy is preserved.

VI. Performance Evaluation

In this section, we evaluate the performance of the Gespun in terms of computational costs and communication overhead.

A. Experimental Settings

Datasets. We use three real-world graph datasets available from the Stanford SNAP (https://snap.stanford.edu/data): *email-Eu-core*, an E-mail network, *soc-sign-bitcoin-alpha*, a Bitcoin Alpha web of trust network, and *p2p-Gnutella08*, a Gnutella peer to peer network, where the numbers of their total nodes are 1005, 3783, and 6301.

We list the key features of the three datasets in Table II. We upload all source codes and an instruction file to https://github.com/UbiPLab/Gespun.

Parameters. We choose the SHA256 and the SHA512 to construct h and H, respectively. The length of secret key K and random number r is 1024 bits. The number of unsorted nodes k is drawn from 0 to 5. We list the detailed parameters in Table III.

Metrics. We measure the time of graph encryption, token generation, and result search, and evaluate the communication overhead of encrypted graph, token, and result set. We conduct each set of experiment twenty times.Communication overhead is computed by measuring the size of the transmitted messages.

Setup. We use JPBC library [37] to implement the basic cryptographic primitives. We use the Paillier cryptosystem as the public-key encryption scheme. We implement Gespun in Java and conducted experiments on a PC server running on Windows 10 Professional with a 3.5GHz-11th Gen Intel(R) Core(TM) i9-11900K x64 processor and 64 GB RAM.

TABLE II
EXPERIMENTAL PARAMETERS

Dataset	Node	Edge	Size (GB)[1]
email-Eu-core	1,005	25,571	0.0475
soc-sign-bitcoin-alpha	3,783	24,186	0.7242
p2p-Gnutella08	6,301	20,777	1.2512

1: Size after preprocessing the original dataset with the Floyd-Warshall algorithm.

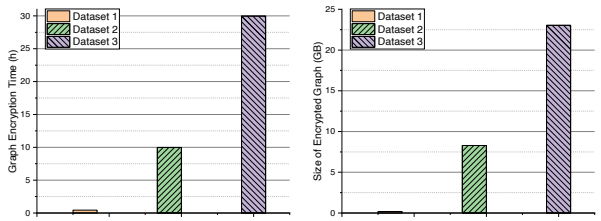
TABLE III
EXPERIMENTAL PARAMETERS

Parameter	Value
n	$\{1005, 3783, 6301\}$
k	$[0, 5]$
$\mathcal{E}$	$\{25571, 24186, 20777\}$
sk_u, pk_u	406, 929
$\|K\|$, $\|r\|$	1024
h; H	SHA256; SHA512
$\|P\|$, $\|F\|$	32

B. Computational Costs and Communication Overhead

We first calculate the shortest paths and distances for all nodes at the user side by using the Floyd-Warshall algorithm [34] on the original graph to obtain path-distance oracles. For example, the time cost is only 4.89 seconds when $n = 1005$ and it is 16.85 minutes when $n = 6301$. This step is a one-time cost and does not impact the overall efficiency greatly. Next, we text the cost of the shortest path queries for the cases where $k = 0$ and $k > 0$. We conduct each set of experiments ten times to draw the average values in Fig. 4, Fig. 5, and Fig. 6.

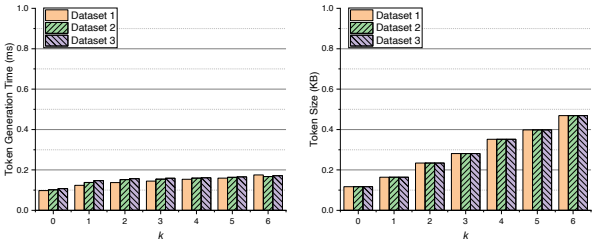
Graph encryption. After acquiring the distance-path oracle, we begin to encrypt the processed graph. As shown in Fig. 4, it is obvious that the computational cost and communication overhead increase with n. When $n = 6301$, the encryption time is 29.9 hours and the encrypted graph is 23 GB. Again, this step is also an one-time cost.



(a) Computational Cost (b) Communication Overhead

Fig. 4. Performance of Graph Encryption.

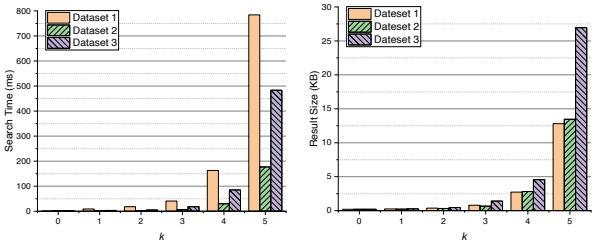
Token generation. The cost of token generation increases with k. Recall that the query q with a $k > 0$ is in the form of a triad (q_1, q_2, q_3). A larger k will lead to more computations on (q_1, q_2, q_3) as well as their size. The results are shown in Fig. 5. When $n = 6301$ and $k = 5$, the token generation time is only 0.175 ms. The token size of the three datasets at the same k are almost the same because the token stays unchanged.



(a) Computational Cost (b) Communication Overhead

Fig. 5. Performance of Token Generation By Varying k.

Shortest path query. As shown in Fig. 6, the cloud server spends little time in computing the shortest path when $k = 0$ due to our special design of the distance-path oracle. Given a source, a destination, and 0 unsorted node, the cloud server can immediately find the shortest path. When k increases, the search time increases because the cloud server has to check each possible path according to the $k!$ permutations. The search time varies for the three datasets at the same k because their distance-path oracles are different. The reason that the size of the returned results does not strictly follow the k-fold relationship is because the length of the obtained shortest paths does not follow the relationship when k is incremented by 1 every time. After receiving the query results, the user recovers the shortest paths locally by first decrypting the distance ciphertext and recovering the path of the smallest distance. The time costs are 134 ms, 149 ms, and 188 ms for the three datasets when $k = 5$. The time cost for the three datasets are different because different k unsorted nodes lead to different lookup time for the cloud server. The result size for the three datasets are different because different k unsorted nodes end up with different shortest paths.



(a) Computational Cost (b) Communication Overhead

Fig. 6. Performance of the Shortest Path Query By Varying k.

VII. CONCLUSIONS

In this work, we first put forth the problem of finding the shortest path in a graph while passing k nodes with no specific order requirements. To protect security in solving the problem, we propose a novel graph encryption scheme Gespun to support the shortest path queries with k unsorted nodes. In our scheme, we design a new distance-path oracle

and encrypt the original graph by using homomorphic encryption, hash function, PRP, and PRF. On top of this structure, we realize the shortest distance queries with k unsorted nodes. We formally prove that Gespun is adaptively semantically-secure in the random oracle model. We implement Gespun and evaluate its efficiency on three real-world datasets, showing that our design is efficient and practical for large-scale graph datasets.

ACKNOWLEDGEMENTS

The work is supported by National Natural Science Foundation of China (NSFC) under the grant No. 62002094 and Anhui Provincial Natural Science Foundation under the grant No. 2008085MF196. It is partially supported by EU LOCARD Project under Grant H2020-SU-SEC-2018-832735.

REFERENCES

[1] "LinkedIn Usage and Revenue Statistics (2022)." Available: https://www.businessofapps.com/data/linkedin-statistics. [Accessed on Feb. 15, 2022]

[2] C. Alcaraz, "Cloud-assisted dynamic resilience for cyber-physical control systems," *IEEE Wireless Communications*, 2018, 25 (1): 76-82.

[3] L. Zhao and L. Chen, "On the privacy of matrix masking-based verifiable (outsourced) computation," *IEEE Trans. Cloud Computing (TCC)*, 2020, 8 (4): 1296-1298.

[4] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," *Proc. 19rd ACM Conference on Computer and Communications Security (CCS)*, October 2012: 965-976, Raleigh, USA.

[5] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," *Proc. Network and Distributed System Security (NDSS) Symposium*, February 2014: 1-15, San Diego, USA.

[6] X. Lei, A. X. Liu, R. Li, and G.-H. Tu, "SecEQP: A secure and efficient scheme for SkNN query problem over encrypted geodata on cloud," *Proc. 35th IEEE International Conference on Data Engineering (ICDE)*, April 2019: 662-673, Macao, China.

[7] K. Mouratidis and M. L. Yiu, "Shortest path computation with no information leakage," *Proc. VLDB Endowment*, August 2012: 692-703, Istanbul, Turkey.

[8] M. Li, L. Zhu, Z. Zhang, C. Lal, M. Conti, and M. Alazab, "User-defined privacy-preserving traffic monitoring against n-by-1 jamming attack," *IEEE/ACM Trans. Networking (ToN)*, 2022, PP (99): 1-1. DOI: 10.1109/TNET.2022.3157654.

[9] M. Li, Y. Chen, Chhagan Lal, M. Conti, F. Martinelli, and M. Alazab, "Nereus: Anonymous and secure ride-hailing service based on private smart contracts," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2022, PP (99): 1-1. DOI: 10.1109/TDSC.2022.3192367

[10] M. Li, Y. Chen, C. Lal, and M. Conti, M. Alazab, and D. Hu, "Eunomia: Anonymous and secure vehicular digital forensics based on blockchain," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2021, PP (99): 1-1. DOI: 10.1109/TDSC.2021.3130583

[11] L. Zhu, M. Li, Z. Zhang, and Z. Qin, "ASAP: An anonymous smart-parking and payment scheme in vehicular networks," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2020, 17 (4): 703-715. DOI: 10.1109/TDSC.2018.2850780.

[12] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," *Proc. 16th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, December 2010: 577-594, Singapore, Singapore.

[13] X. Meng, S. Kamara, K. Nissim, and G. Kollios, "GRECS: Graph encryption for approximate shortest distance queries," *Proc. 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, October 2015: 504-517, Denver, USA.

[14] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," *Proc. Theory of Cryptography Conference (TCC)*, February 2005: 325-342, Cambridge, USA.

[15] D. J. Wu, J. Zimmerman, J. Planul, and J. C. Mitchell, "Privacy-preserving shortest path computation," *Proc. The Network and Distributed System Security Symposium (NDSS)*, February 2016: 1-15, San Diego, USA.

[16] A. C. Yao, "How to generate and exchange secrets (extended abstract)," *Proc. 23rd Symposium on Foundations of Computer Science (FOCS)*, November 1986: 162-167, Chicago, USA.

[17] Q. Wang, K. Ren, M. Du, Q. Li, and A. Mohaisen, "SecGDB: Graph encryption for exact shortest distance queries with efficient updates," *Proc. 21st International Conference on Financial Cryptography and Data Security (FC)*, 2017: 79-97, Sliema, Malta.

[18] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," *Proc. 16th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, May 1999: 223-238, Prague, Czech Republic.

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, 1959, 1 (1): 269-271.

[20] E. Ghosh, S. Kamara, and R. Tamassia, "Efficient graph encryption scheme for shortest path queries," *Proc. ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2021: 516-525, Hong Kong, China.

[21] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, "Introduction to Algorithms (2nd edition)." *The MIT Press*, 2001.

[22] K. C. K. Lee, W.-C. Lee, H. V. Leong, and B. Zheng, "Navigational path privacy protection," *Proc. 18th ACM conference on Information and knowledge management (CIKM)*, November 2009: 691-700, Hong Kong, China.

[23] Y. Chen, M. Li, S. Zheng, D. Hu, C. Lai, and M Conti, "One-time, oblivious, and unlinkable query processing over encrypted data on cloud" *Proc. 22nd International Conference on Information and Communications Security (ICICS)*, August 2020: 350-365, Copenhagen, Denmark.

[24] A. H. and E. Ayday, "Profile matching across online social networks," *Proc. 22nd International Conference on Information and Communications Security (ICICS)*, August 2020: 54-70, Copenhagen, Denmark.

[25] J. Ni, K. Zhang, Y. Yu, X. Lin, and X. Shen, "Privacy-preserving smart parking navigation supportingeEfficient driving guidance retrieval," *IEEE Trans. Vehicular Technology (TVT)*, 2018, 67 (7): 6504-6517.

[26] M. Li, Y. Chen, S. Zheng, D. Hu, C. Lal, and M. Conti, "Privacy-preserving navigation supporting similar queries in vehicular networks," *IEEE Trans. Dependable and Secure Computing (TDSC)*, 2020, PP (99): 1-1. DOI: 10.1109/TDSC.2020.3017534.

[27] H. Yu, X. Jia, H. Zhang, and J. Shu, "Efficient and privacy-preserving ride matching using exact road distance in online ride hailing services," *IEEE Trans. Services Computing (TSC)*, 2020, PP (99): 1-1.

[28] "GraphDB." Available: https://graphdb.ontotext.com.

[29] "Neo4j Graph Data Platform-The Fastest Path to Graph." Available: https://neo4j.com.

[30] "Titan-Distributed Graph Database." Available: http:titan.thinkaurelius.com.

[31] M. M. Flood, "The Ttraveling-salesman problem," *Operations research*, 1956, 4 (1): 1-75.

[32] G. Laporte and S. Martello, "The selective travelling salesman problem," *Discrete Applied Mathematics*, 1990, 26 (2-3): 193-207.

[33] "The seven bridges of Königsberg," *The world of mathematics*, 1956: 1-8.

[34] R. W. Floyd, "Algorithm 97: Shortest path," *Communications of the ACM*, 5 (6): 345.

[35] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *Proc. 13th ACM Conference on Computer and Communications Security (CCS)*, October/November 2006: 79-88, Alexandria, USA.

[36] J. Katz and Y. Lindell, "Introduction to Modern Cryptography-Third Edition." *Chapman & Hall/CRC Press*, 2021.

[37] "The Java Pairing Based Cryptography Library (JPBC)." Available: http://gas.dia.unisa.it/projects/jpbc.