Time-Restricted, Verifiable, and Efficient Query Processing over Encrypted Data on Cloud

Li, Meng; Gao, Jianbo; Zhu, Liehuang; Zhang, Zijian; Lal, Chhagan; Conti, Mauro

DOI
10.1109/TSC.2023.3311586

Publication date
2024

Document Version
Final published version

Published in
IEEE Transactions on Services Computing

Time-Restricted, Verifiable, and Efficient Query Processing Over Encrypted Data on Cloud

Meng Li, *Senior Member, IEEE*, Jianbo Gao, *Student Member, IEEE*, Liehuang Zhu, *Senior Member, IEEE*, Zijian Zhang, Chhagan Lal, and Mauro Conti, *Fellow, IEEE*

***Abstract*—Outsourcing data users' location data to a cloud server (CS) enables them to obtain k nearest points of interest. However, data users' privacy concerns hinder the wide-scale use. Several studies have achieved Secure k Nearest Neighbor (SkNN) query, but do not address *time-restricted access* or *result privacy*, and randomly partition data items which degrades efficiency. In this article, we propose Time-restricted, verifiable, and efficient Query Processing (TiveQP). TiveQP has three distinguishing features. 1) Expand SkNN: data users can query k nearest locations open at a specific time. 2) Adopt a stronger threat model: we assume the CS is malicious and propose *complementary set* (i.e., transform proving "in" a set to proving "in" its complementary set) to allow data users to verify results without leaking unqueried data items' information. 3) Improve efficiency: we design a space encoding technique and a pruning strategy to improve efficiency in query processing and result verification. We formally proved the security of TiveQP in the random oracle model. We conducted extensive evaluations over a Yelp dataset to show that TiveQP significantly improves over existing work, e.g., top-10NN query over 100 thousand data items only needs 10 ms to get queried results and 1.4 ms for verification.**

***Index Terms*—Efficiency, query processing, security, time-restricted access, verification.**

Manuscript received 9 June 2023; revised 28 July 2023; accepted 31 August 2023. Date of publication 4 September 2023; date of current version 12 June 2024. This work was supported by the National Natural Science Foundation of China (NSFC) under the Grant 62372149 and Grant U23A20303, and also by the National Key Research and Development Program of China under Grant 2021YFB2701202. It is partially supported by EU LOCARD Project under Grant H2020-SU-SEC-2018-832735. Recommended for acceptance by S. Deng. *(Corresponding author: Zijian Zhang.)*

Meng Li is with the Key Laboratory of Knowledge Engineering with Big Data, Hefei University of Technology, Ministry of Education, Hefei, Anhui 230601, China, and with the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, Anhui 230601, China, and with the Anhui Province Key Laboratory of Industry Safety and Emergency Technology, Hefei, Anhui 230601, China, and also with the Intelligent Interconnected Systems Laboratory Anhui Province, Hefei University of Technology, Hefei, Anhui 230601, China (e-mail: mengli@hfut.edu.cn).

Jianbo Gao was with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Xiamen, Fujian 351100, China. He is now with the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, Anhui 230601, China (e-mail: jianbogao@mail.hfut.edu.cn).

Liehuang Zhu is with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Xiamen, Fujian 351100, China (e-mail: liehuangz@bit.edu.cn).

Zijian Zhang is with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100811, China, and also with the Southeast Institute of Information Technology, Beijing Institute of Technology, Xiamen, Fujian 351100, China (e-mail: zhangzijian@bit.edu.cn).

Chhagan Lal is with the Department of Mathematics and HIT Center, University of Padua, 35131 Padua, Italy (e-mail: c.lal@tudelft.nl).

Mauro Conti is with the Department of Mathematics and HIT Center, University of Padua, 35131 Padua, Italy, and also with the Department of Intelligent Systems, CyberSecurity Group, TU Delft, 2628 CD Delft, The Netherlands (e-mail: mauro.conti@unipd.it).

Digital Object Identifier 10.1109/TSC.2023.3311586

I. Introduction

LOCATION-BASED Services (LBSs) enable users to upload their current location and location query to a Cloud Server (CS), which returns a query result (e.g., ten nearest cafes) to the users [1], [2], [3], [4]. However, privacy is a major concern because locations are related to user privacy and CSs are not fully trusted [5], [6], [7], [8], [9], [10]. The data stored on a cloud may be analyzed or leaked. As reported [11], a British firm admitted that some of its location data was collected without seeking permission from users. Recently, Secure k Nearest Neighbor (SkNN) [6], [12], [13], [14], [15], [16], [17], [18] is proposed to address this problem.

We have four motivations to advance the state-of-the-art. 1) We observe a new requirement *time-restricted access* in kNN, that is, data owners have an access time limit on their data items and data users visit a location at a specific time. For example, a patient makes an appointment with an oncologist (a doctor specialized in cancer) who only provides medical service on Monday mornings. The integration of location and time refers to matching opening time with access time and matching locations. It is essential to kNN since no users want to visit a closed clinic. 2) We focus on the *privacy concerns* of data owners and data users. It includes the spatial attributes of location, time attribute of location, attached data items, and queries. 3) We find that *result privacy is not provided* even if result verifiability is enforced when the CS misbehaves. For example, a data user can acquire the privacy of some unqueried data items from the result. 4) Existing work randomly partition data items and ignore their spatial attributes that are useful for *improving query efficiency*.

We compare TiveQP with SkNN schemes [6], [12], [13], [14], [15], [16], [17], [18] and privacy-preserving range query schemes [19], [20], [21] in Table I.

First, we expand SkNN to support privacy-preserving time-restricted access. Existing SkNN schemes [6], [12], [13], [14], [15], [16], [17], [18] focus on querying the k nearest locations while not considering the time attribute. Access time is related to data users' daily schedules, which are highly sensitive. As in the example above, the access time along reveals the patient's health condition to some extent. Therefore, time should be protected. Second, we adopt a stronger threat model and achieve privacy-preserving result verification. Most existing work [6], [12], [13], [14], [15], [16], [17] only use a semi-honest (honest-but-curious) model for the CS. A stronger security model calls for result verification. SVkNN [18] proposes a verifiable SkNN framework, but it uses two servers and does not address access

TABLE I
COMPARISON AMONG PREVIOUS STUDIES AND TIVEQP

Scheme \ Feature	Single server	Round	Tree	Strong security	Sublinear query	Access time	Type& city	Result verifiability	Result accuracy
Wong et al. [12]	✓	1							Accurate
Hu et al. [13]	✓	$O(\log n)$	✓		✓				Accurate
Yao et al. [14]	✓	2		✓	✓				Accurate
Yi et al. [15]	✓	1							Accurate
Elmehdwi et al. [16]		1	✓	✓					Accurate
Wang et al. [6]	✓	1	✓		✓				Accurate
SecEQP [17]	✓	1	✓	A[1]-IND-CKA	$O(k\log n)$				Approximate
SVkNN [18]		1	✓	Semantic				✓	Accurate
PBtree [19]	✓	1	✓	IND-CKA	$O(k\log n)$	n/a	n/a		Low FPR[2]
IBtree [20]	✓	1	✓	A-IND-CKA	$O(k\log n)$	n/a	n/a		Low FPR
ServeDB [21]	✓	1	✓	L-security[3]	$O(k\log n)$	n/a	n/a	✓	Low FPR
Our solution TiveQP	✓	**1**	✓	**A-IND-CKA**	$\boldsymbol{O(k\log(\frac{n}{tc}))}$[4]	✓	✓	✓	**Low FPR**

1: Adaptive; 2: false positive rate; 3: L-secure against adaptive attacks; 4: t: number of types, c: number of cities.

time or achieve sublinear query latency. ServeDB [21] can verify the query result but leaks other locations' information to data users. Third, we improve query efficiency. The tree construction is crucial for query processing, but SecEQP [17], SVkNN [18] and (range query schemes) PBtree [19], IBtree [20], and ServeDB [21] randomly partition the dataset. We propose that the attribute be exploited to improve efficiency. Therefore, the limitation of these schemes is that they cannot satisfy the entire conditions. Embracing such properties will make the SkNN more appealing to data owners and users [22], [23], [24]. Different from the previous SkNN works, the TiveQP scheme allows for time-restricted access while providing strong security, result verification, and improved efficiency.

There are three technical challenges to be addressed.

Challenge 1: How to achieve SkNN query processing with secure time-restricted access? Space transformation and encoding techniques are commonly used to solve the location query problem. The secure time-restricted access is essentially a privacy-preserving range query problem. A straightforward approach is to process the spatial attributes and time attribute separately. This, however, results in low efficiency. We must integrate the two problems and perform queries uniformly. To address this issue, we use a space encoding technique to process locations and leverage prefix encoding to handle location and time. All generated codes of a data item are inserted into an Indistinguishable Bloom Filter (IBF) [20], that is, a secure index. Each query is encoded into a trapdoor. In this way, we integrate the two problems by converting them into a joint keyword query problem.

Challenge 2: How to achieve result verification without violating result privacy under the existing SkNN framework? The seminal paper in this line of research is ServeDB [21]. The authors proposed an effective method to verify the correctness and completeness of results without introducing a new structure. For completeness, the CS generates proof information related to the search paths for the data user to reproduce each step on these paths. Unfortunately, some key nodes that are leaf nodes are included in the returned proof. This allows the data user to freely query any range on the nodes, violating the privacy of unqueried data items. To address this problem, we propose the concept of Complementary Set. **Instead of proving that a location does not fall in a region A, we prove that it resides in A's complementary region**. Specifically, the data owner computes for each data item three complementary sets for location, type, and opening time. Each set is transformed into a minimum set of prefixes. Each prefix is queried into the node IBF to produce a bit segments. Further, a keyed hash message authentication code (HMAC) of the segment is computed as a signature. When proving mismatch, we return the matched bit segment and its HMAC, instead of the entire IBF or segments, to data user.

Challenge 3: How to improve the query efficiency further without sacrificing security? Improving query efficiency without sacrificing security is a difficult task [25], [26], [27]. To improve, when constructing a TiveTree of secure indexes, we design a pruning strategy by first classify the data items according to their spatial attributes, e.g., type and city, and then constructing a subtree for each type of location from the bottom to up. By doing so, we eliminate unnecessary search paths and reduce query processing time and result verification time. The query complexity reduces to $O(k\log(\frac{n}{t*c})$ where t and c are the numbers of types and cities, respectively.

Our contributions are summarized as follows.

- To the best of our knowledge, this is the first work to integrate time-restricted access with SkNN query processing.
- We propose a privacy-preserving result verification method. Specifically, we propose the concept of complementary set: instead of proving that a data item does not satisfy a condition, we prove that it satisfies its complementary conditions, preventing privacy leakage.
- We design a simple-but-effective space encoding technique to process data items and design a pruning strategy to improve efficiency in query processing and result verification.
- We perform formal security analysis, including data privacy, query privacy, correctness and completeness of query results. We conduct extensive experiments to demonstrate that TiveQP has a significant efficiency improvement.

The remainder of this article is organized as follows. We review some related work in Section II. Section III introduces some preliminaries. Section IV formalizes the problem. In Section V, we elaborate TiveQP, followed by the security analysis in Section VI and performance evaluation in Section VII, respectively. Finally, we provide some discussions in Section VIII and conclude the paper in Section IX.

II. Related Work

In this section, we revisit some related work on SkNN query processing and privacy-preserving range query processing.

A. SkNN Query Processing

Lei et al. [17] proposed a secure and efficient query processing protocol called SecEQP. They leveraged a set of primitive projection functions to convert a location into several feasible regions. Next, it integrates the regions into an irregular polygon. Given two locations, the CS compute the proximity of them by comparing whether their corresponding composite projection function output codes equal. The codes are inserted into an IBF. All locations' IBFs are used to construct an IBF Tree. A data user computes projection function values and generates a set of codes as a token. The CS conducts the query processing by searching the token in the IBF Tree. However, SecEQP cannot guarantee result verification.

Cui et al. [18] proposed a secure and verifiable k nearest neighbor query processing protocol SVkNN. They adopted Voronoi diagram to divide the whole area. This space encoding method is a replaceable unit that is not directly related to their core design. They design a new data structure, i.e., verifiable and secure index VSI to support fast query and verification. Next, they propose several secure protocols and a compact verification method to facilitate the operation over VSI to support performing the search over the secure index. However, it uses two CSs and incurs extra computation and communication overhead.

We claim that most data users look for locations in a general area, i.e., they do not need to find locations in areas of such complex shapes (irregular polygon and Voronoi cell). Moreover, these techniques require data users to spend more time pre-processing the whole area. Thus, we design a simple-but-effective space encoding technique in this work to save the computation time for both data users in pre-processing locations and the CS in searching the index tree.

B. Privacy-Preserving Range Query Processing

Li et al. [19] proposed the first range query processing protocol that achieves index indistinguishability under the indistinguishability against chosen keyword attack. A data owner converts each data item (number) by prefix encoding [28]. Each prefix is hashed, randomized, and inserted into a Bloom filter. A PBtree is constructed for all data items. A data user converts a query range into a minimum set of prefixes and computes a trapdoor. The CS searches the trapdoor in the PBtree to find a match. Based on PBtree, Li et al. [20] concerned privately processing conjunctive queries including keyword conditions and range conditions and proposed a privacy-preserving conjunctive query processing protocol supporting adaptive security, efficient query processing, and scalable index size at the same time. Specifically, they adopt prefix encoding [19] and design an indistinguishable Bloom filter (IBF), i.e., twin Bloom filter. A pseudo-random hash function determines a cell location, i.e., which twin cell stores '1.' Instead of building a PBtree, they construct an IBF Tree.

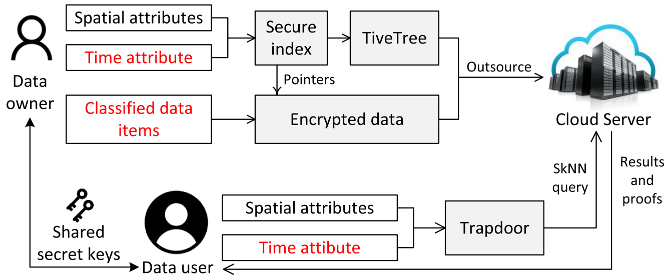


Fig. 1. System model of TiveQP.

Wu et al. [21] proposed a secure and scalable protocol ServeDB to support multi-dimensional range queries over encrypted data. It also considers the malicious assumption for the CS and then put forth a verification mechanism. ServeDB opens a new way for result verification in tree-based protocols by leveraging the tree index to verify the correctness and completeness of query results. Specifically, data owners generate verification information and the CS generates proofs for the data users to check the results. However, the returned proofs includes leaf nodes, i.e., Bloom filters, which make it possible for the data user to query other ranges on the Bloom filters. This is a direct privacy violation on other data items.

III. Problem Statement

In this section, we introduce our system model, threat model, and design objectives.

A. System Model

Our system model consists of data owner, data user, and CS, as depicted in Fig. 1.

• **Data Owner** has a dataset with n data items (locations): $\mathcal{D} = \{L_1, L_2, \ldots, L_n\}$. The data owner encrypts each L_i and builds a secure index B. Each location has two spatial attributes (location type and a pair of coordinates) and one time attribute (allowed access time period). The data owner extracts the attributes of L_i to build a secure index B by using secret keys and a unique random number r. All indexes are classified and treated as leaf nodes to construct a TiveTree TR with verification information from bottom to top. The Chosen Plaintext Attack (CPA)-secure encryption algorithm Enc encrypts all data items. Next, the data owner outsources the TiveTree and encrypted data items to CS, and delegates the query service to authenticated data users by sharing secret keys with them. The Pointer in Fig. 1 means that each node index has an address pointing to the encrypted data for retrieval.

• **Data User** extracts spatial attributes and time attribute (access time) of each query location to build a trapdoor to be sent to the CS. After receiving the search result from the CS, the data user uses the shared keys and random number to verify the correctness and completeness of the results, and decrypt the encrypted data items.

• **CS** stores the TiveTree and the encrypted data items, and processes SkNN queries for the data users. When it searches a trapdoor in the TiveTree, it generates corresponding proofs for result verification. Finally, it returns the query results and the proofs to the data user.

Now we give the formal definitions of the data model, index tree model, and the query model.

Definition 1 (Data model): A dataset $\mathcal{D}$ is defined as a table with five attributes $\{id, typ, lat, lon, per\}$ and n records $\{L_1, L_2, \ldots, L_n\}$. id is the record identity and typ is the location type, e.g., bank, store, and cafe. lat is the latitude of location and lon is the longitude of location. per is the allowed access time period.

Definition 2 (Index Tree model): A index tree $\mathcal{TR}$ is defined as a binary tree with n leaf nodes and $n-1$ non-leaf nodes. A leaf node consists of an IBF B, a hash value HV, a pair of string set and HMAC set (bit, H) for its Location Complementary Set (LCS), a pair of string set and HMAC set for its Time Complementary Set (TCS), a node number, and a pointer to an encrypted data item. A non-leaf node in a subtree of a specific type consists of an IBF, a hash value HV, two similar pairs of string set and HMAC set. A non-leaf node above the subtree consists of an IBF, a hash value HV, a pair of string set and HMAC set for its tYpe Complementary Set (YCS). $\mathcal{TR}$ is built in a bottom-up fashion in two phases: 1) Within each subtree, the data owner merges two child nodes until the subtree root by computing new B, HV, (bit, H) for LCS, (bit, H) for TCS. 2) Staring from the subtree roots, the data owner merges two nodes by computing new B, HV, (bit, H) for YCS.

Definition 3 (Query Model): A query Q is defined as $Q = (typ, lat, lon, time, k)$ where typ is the location type, lat and lon are the latitude and longitude of current location, $time$ is the time at which the data user will visit the location. For example, Alice wants to securely query a CS whether her nearest three cafes are open at 9:00 am. A SQL query is $\mathsf{select} * \mathsf{from\ data\ items\ where\ type} = \text{“}\mathsf{cafe}\text{''}\ \mathsf{and\ grid} = \mathsf{G(lat\ ,lon)\ and\ time} = \mathsf{0900\ and\ k} = \mathsf{3}$. Here, $\mathsf{g}(\cdot,\cdot)$, given a location, outputs a grid identity.

Now we formulate the five algorithms of TiveQP in Definition 3. Setup generates secret keys and a random number. Index builds the secure indexes and a tree. $\mathsf{Trapdoor}$ builds a trapdoor. Query assists the CS in searching encrypted data items by using the tree and the trapdoors. Verify verifies the correctness and completeness of query results.

Definition 4 (Verifiable Multi-Dimensional Query): Our verifiable multi-dimensional query processing scheme is a tuple of five polynomial time algorithms:

$\mathsf{Setup}(1^\lambda) \rightarrow (sk, sk', \mathcal{SK}, r)$: is a probabilistic key generation algorithm executed by data owner. It takes a security parameter λ as input, and outputs two secret keys sk, sk' to encrypt data items and compute HMAC, a set of hash keys $\mathcal{SK} = \{k_1, k_2, \ldots, k_{m+1}\}$ and a random number r.

$\mathsf{Index}(\mathcal{D}, sk, sk', \mathcal{SK}, r) \rightarrow (\mathcal{E}, \mathcal{TR})$: is a probabilistic algorithm executed by data owner. It takes a location dataset $\mathcal{D}$, two secret keys sk, sk', a set of hash keys $\mathcal{SK}$, and a random number r as inputs, and outputs the encrypted data items $\mathcal{E} = \{E_1, E_2, \ldots, E_n\}$ and a TiveTree $\mathcal{TR}$. $\mathcal{TR}$ is constructed by nodes of IBF and each leaf node v has a secure index B_v and verification information.

$\mathsf{Trapdoor}(\mathcal{SK}, Q) \rightarrow \mathcal{T})$: is a deterministic algorithm executed by data user. It takes a set of hash keys $\mathcal{SK}$ and a query Q as inputs, and outputs a trapdoor $\mathcal{T}$.

$\mathsf{Query}(\mathcal{TR}, \mathcal{E}, r, \mathcal{T}) \rightarrow (\mathcal{R}, \pi)$: is a deterministic algorithm executed by the CS. It takes the TiveTree $\mathcal{TR}$, a ciphertext set $\mathcal{E}$, a random number r, and the trapdoor $\mathcal{T}$ as inputs, and outputs the query result set $\mathcal{R}$ and a proof set π.

$\mathsf{Verify}(\mathcal{T}, \mathcal{R}, \pi, sk, sk', \mathcal{SK}) \rightarrow (1/0)$: is a deterministic algorithm executed by data user to verify the query results. It takes a trapdoor $\mathcal{T}$, a query result set $\mathcal{R}$, a proof set π, two secret keys sk, sk', and a set of hash keys $\mathcal{SK}$ as inputs, and outputs $1/0$ for accept/reject.

B. Threat Model

Different than the semi-honest model in existing work [17], [19], [20], we consider a stronger adversary. Besides being curious about data items, indexes, and queries, it can tamper with the query results or search a part of the dataset. In reality, the attacks are from the CS being compromised by adversaries or some rogue employee. We adopt the adaptive INDistinguishable under Chosen Keyword Attack (IND-CKA) threat model [29].

C. Design Objectives

Security: For attacks from semi-honest entities, we leverage secure indexes and trapdoors to protect privacy. *Data privacy*. The adversary cannot learn anything useful about the data items from the secure index and encrypted data items; *Query privacy*. The adversary cannot infer anything useful about the location query from the trapdoor; *Result privacy*. The data user cannot know anything about the unmatched data items more than the fact that they do not match.

Verifiability: For the attack from malicious CS, we design a privacy-preserving and efficient method to allow data users to verify the query results $\mathcal{R}$. There are two requirements to be satisfied. *Correctness*. Each returned data item L is not tampered and is the real date item in the original dataset $L \in \mathcal{D}$.

Completeness: All returned data items are answers to the SkNN query while and all other data items are not.

Efficiency: TiveQP should achieve low query latency, i.e., a data user receives a result set and a proof set within sublinear query processing time.

IV. Preliminaries

In this section, we briefly revisit four underlying techniques for structuring TiveQP, namely space encoding, prefix encoding, IBF, and Merkle tree.

A. Space Encoding

In the proposed space encoding technique, a data owner extracts the location (lat, lon) of each data item di. The location is projected into a fixed grid g with a grid identity and then expanded to a wider area $G(lat, lon)$, e.g., a set of nine grids covering g. Next, the data owner transforms $G(lat, lon)$ into a minimum set of prefixes MS. A data user extracts a grid identity

TABLE II
KEY NOTATIONS OF TIVEQP

Notation	Definition
$L, D, E,$	Data items, data set, encrypted data item
B, r	IBF, random number
$k, \mathcal{TR}$	Number of query items, TiveTree
$di, id, typ, lat, lon, per$	Data item, record identity, location type
lat, lon, per	Latitude, longitude, access time period
HV, H, bit	Hash value, HMAC, bits of HMAC
$g, G(.), \mathcal{G}$	Grid, grid function, a set of girds
$MS, F(.), \mathcal{PF}$	Minimum set of prefixes, prefix family
$w, \{x_1, x_2, \cdots, x_l\}, pr$	Bit length, prefix string, prefix
N, m	Number of cell twins and hash functions
$\{h_1, h_2, \cdots, h_m\}$	Pseudo random hash function
$\mathcal{SK}$	Secret keys
$\mathcal{T}, \mathcal{T}_1$	Trapdoor, type trapdoor
$\mathcal{T}_2, \mathcal{T}_3$	Location trapdoor, time trapdoor
$v, root$	Node, root node
$result\_num, \mathcal{R}$	Element number in result set, result set
ε	Encrypted data item
z	Number of row in a trapdoor
π	Proof set
$\mathcal{A}$	Adversary

covering the current location and computes a prefix family that will be matched with MS.

B. Prefix Encoding

Given a number x of w bits with a binary format being $x_1x_2 \ldots x_w$, its prefix family $F(x)$ is the group of $w+1$ prefixes $\{x_1x_2 \cdots x_w, x_1x_2 \cdots x_{w-1}*, \cdots, x_1 * \cdots *, * * \cdots *\}$. Given a range $[A, B]$, we transform it to a minimum set of prefixes $MF([A, B])$ satisfying the condition that the union of the prefixes is the same as $[A, B]$. For a number x and range $[A, B]$, $x \in [A, B]$ if and only if $F(x) \cap MF([A, B]) \neq \emptyset$.

C. IBF

An IBF is an array B of N cell twins, m pseudo random hash functions $h_1, h_2, \cdots, h_m$, and a random oracle H. H is used to determine which cell stores '1'. A keyword kw is hashed to m twin cells $B[h_1(kw)], B[h_2(kw)], \cdots, B[h_m(kw)]$. For each of the m twins, H determines the chosen cell location $H(h_{m+1}(h_i(kw)) \oplus r)$ and r is a random number. We set its value to be 1 and the other cell's value to 0.

D. Merkle Tree

Merkle tree is a complete binary tree equipped with a function hash H and an arbitrary function A. For two child nodes, n_l and n_r, of any non-leaf node, n_p, the function A of n_p is $A(n_p) = H(A(n_l)||A(n_r))$.

V. PROPOSED SCHEME

In this section, we first give an overview of our proposed scheme and then dive into the details. We list the key notations of TiveQP in Table II.

A. Overview

We present index tree in Section V-B. We show how to compute a trapdoor in Section V-C. We show how to answer multi-dimensional location queries and generate proofs in Section V-D. We discuss result verification process in Section V-E. The TiveQP overview is depicted in Fig. 2.

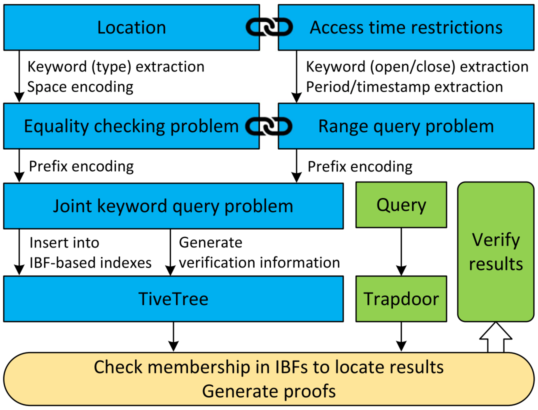

Location
Access time restrictions
Keyword (type) extraction
Space encoding
Keyword (open/close) extraction
Period/timestamp extraction
Equality checking problem
Range query problem
Prefix encoding
Prefix encoding
Joint keyword query problem
Query
Verify results
Insert into IBF-based indexes
Generate verification information
TiveTree
Trapdoor
Check membership in IBFs to locate results
Generate proofs

Fig. 2. Overview of the TiveQP scheme.

We use space encoding to convert the location information of data items into a set of grid set encodings, and use prefix encoding to convert the access time of data items into a set of time minimum prefix set encodings. We integrate the problems of access time and location information into a joint keyword query problem by inserting these two sets of encodings into the same IBF. Only trapdoors that satisfy both access time and location can retrieve data items.

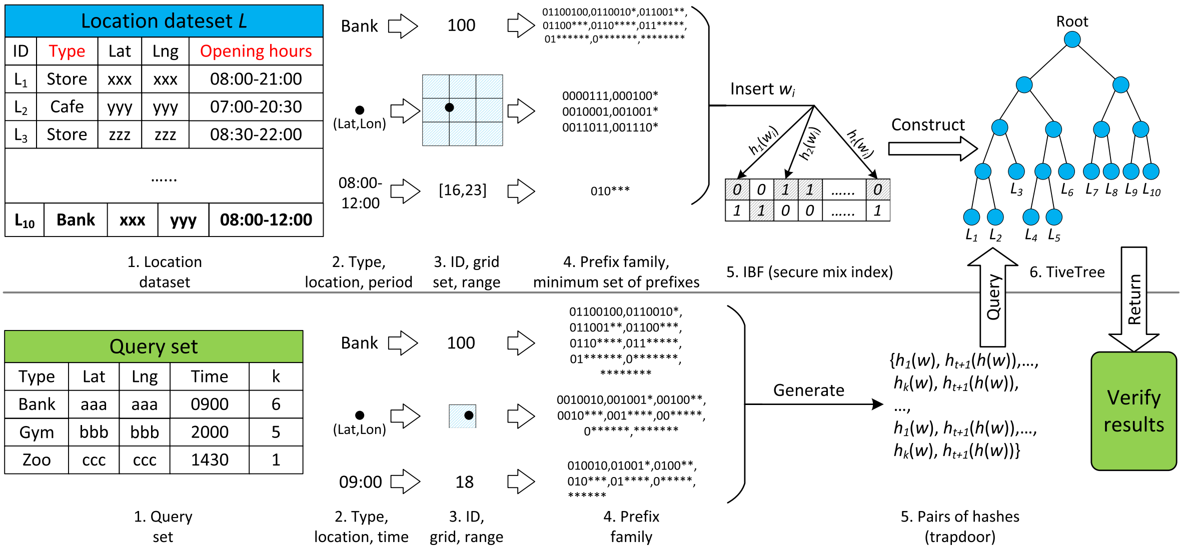
B. Index Building

We assume that the service region is divided uniformly into a set of girds $\mathcal{G} = \{g_1, g_2, \ldots\}$. We consider time period from "00:00" to "24:00" and encode each half an hour as a unit, which turns the time period to [0, 47].

We assume that the data owner and the data users share $m+1$ secret keys $\mathcal{SK} = \{sk_1, sk_2, \ldots, sk_{m+1}\}$ and a random number r. m pseudorandom hash functions $h_1, h_2, \ldots, h_m$ are built as $h_i(.) = \mathsf{HMAC}(.)\%N$ $(1 \leq i \leq m)$. Another pseudorandom hash function is defined as $h_{m+1}(.) = \mathsf{HMAC}_{m+1}(.)$. H is a hash function defined as $H(.) = \mathsf{SHA256}(.)\%2$. Check (B, kw) checks whether a keyword kw exists in an IBF B. Here, a keyword is an element in a trapdoor. For example, $\mathcal{T} = \{110, 11*, 1**\}$ has three keywords. $\mathsf{QueLoc}(B, kw)$ outputs the a set of location values by querying a keyword kw in an IBF B.

A data owner has a location dataset $\mathcal{D} = \{L_1, L_2, \ldots, L_n\}$, where each data item L_i is a location with identity, location type, latitude, longitude, and opening hours as shown in Fig. 3. For example, "Bank" is transformed into type "100" and then a minimum set of prefixes. The similar operations are applied for location and opening time. Then the data owner computes, stores, and uploads the encrypted data items $\mathcal{E}$ and a TiveTree $\mathcal{TR}$. First, the data owner encrypts each L_i to a ciphertext $\mathsf{Enc}(sk, L_i)$. Second, the data owner computes a secure index for each L_i as follows.

- Extract $(typ_i, lat_i, lon_i, per_i)$.
- Given the two coordinates, locate the location in a grid set $G(i) = G(lat_i, lon_i)$, e.g., a burger shop is located at a grid g_4 and its service area covers $\mathcal{G} = \{g_0, g_1, \ldots, g_8\}$.



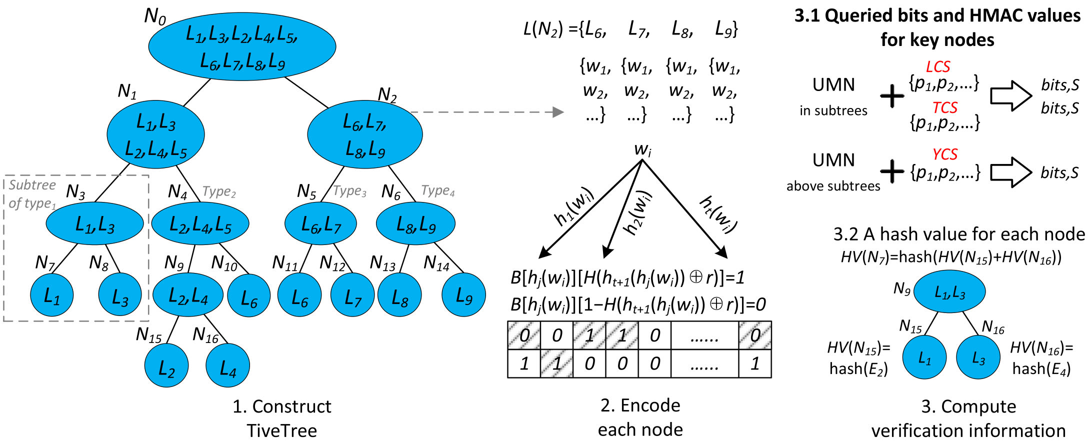
Location dataset *L*

ID	Type	Lat	Lng	Opening hours
L1	Store	xxx	xxx	08:00-21:00
L2	Cafe	yyy	yyy	07:00-20:30
L3	Store	zzz	zzz	08:30-22:00
		……		
L10	Bank	xxx	yyy	08:00-12:00

Query set

Type	Lat	Lng	Time	k
Bank	aaa	aaa	0900	6
Gym	bbb	bbb	2000	5
Zoo	ccc	ccc	1430	1

Fig. 3. Example of TiveQP.



N_0
$L_1,L_3,L_2,L_4,L_5,$
L_6,L_7,L_8,L_9
N_1
L_1,L_3
L_2,L_4,L_5
N_2
$L_6,L_7,$
L_8,L_9
Subtree of type1
N_3
L_1,L_3
N_4 Type2
L_2,L_4,L_5
N_5 Type3
L_6,L_7
N_6 Type4
L_8,L_9
N_7 N_8 N_9 N_{10} N_{11} N_{12} N_{13} N_{14}
L_1 L_3 L_2,L_4 L_6 L_6 L_7 L_8 L_9
N_{15} N_{16}
L_2 L_4
1. Construct TiveTree
$L(N_2)=\{L_6,\ L_7,\ L_8,\ L_9\}$
$\{w_1, w_2, …\}$ $\{w_1, w_2, …\}$ $\{w_1, w_2, …\}$ $\{w_1, w_2, …\}$
w_i
$h_1(w_i)$
$h_2(w_i)$
$h_k(w_i)$
$B[h_j(w_i)][H(h_{t+1}(h_j(w_i))\oplus r)]=1$
$B[h_j(w_i)][1-H(h_{t+1}(h_j(w_i))\oplus r)]=0$
0 0 1 1 0 …… 0
1 1 0 0 0 …… 1
2. Encode each node
3.1 Queried bits and HMAC values for key nodes
UMN in subtrees
LCS $\{p_1,p_2,…\}$
TCS $\{p_1,p_2,…\}$
bits,S
bits,S
UMN above subtrees
YCS $\{p_1,p_2,…\}$
bits,S
3.2 A hash value for each node
$HV(N_7)$=hash($HV(N_{15})$+$HV(N_{16})$)
N_9 L_1,L_3
N_{15} N_{16}
$HV(N_{15})$= hash(E_2)
L_1 L_3
$HV(N_{16})$= hash(E_4)
3. Compute verification information

Fig. 4. TiveTree index and query processing.

- Generate a random number r, insert each prefix pr in $G(i)$'s minimum set of prefixes into an IBF B_i by setting $B_i[h_o(pr)][H(h_{m+1}(h_o(pr))\oplus r)]=1$ and $B_i[h_o(pr)][1-H(h_{m+1}(h_o(pr))\oplus r)]=0$ $(1\leq o\leq m)$.
- Encode per_i into a minimum set of prefixes $\mathcal{MS}_i$ [19] and insert each prefix into B_i, e.g., "08:00–12:00" is encoded to $\{010***\}$.

Next, the data owner constructs a TiveTree $\mathcal{TR}$ as follows.

- Classify and order indexes according to type and coordinates; for each type, construct a subtree from bottom to up with corresponding indexes as leaf nodes as shown in Fig. 4.
- In each subtree, for the each data item di on a leaf node v, compute the Location Complementary Set (LCS) and Time Complementary Set (TCS). LCS is the grid identity set of the regions that are outside the regions covering the location of di. For example, there are 15 grids in total and the current data item locates in grid 8, then its LCS is $[1,7]\cup[9,15]$. Its minimum set of prefixes is $\{0001, 001*, 01**, 1001, 101*, 11**\}$. If a query's location falls into the complementary region of a node, we can prove that the query does not match the IBF of the node. TCS is the time period set that are the closed hours of di. For example, the opening hour $per_v=$ "$08:00-12:00''$, then its TCS is $[0000, 0800]\cup[1200, 2400]$ transformed into $[0,15]\cup[24,47]$. Its minimum set of prefixes is $\{0***, 11**, 10*1*\}$. For each keyword w_j in LCS, compute $bits_{vj}=\mathsf{QueLoc}(B_v, w_j)$ and $S_{vj}=\mathsf{HMAC}_{sk'}(bits_{vj})$. Define $bits_v$ and S_v as two sets containing all computed $\{bits_{vj}\}$, $\{S_{vj}\}$. For TCS, compute $bits_v$ and S_v similarly. $\{bits, S\}$ is a **proof of completeness**. Afterward, compute a hash value $HV_v=\mathsf{hash}(E_v)$ as a **proof of correctness**.
- In each subtree, for each non-leaf node v except subroot, merge the complementary regions of the two child nodes and build a new LCS (TCS) to compute $bits_v$ and S_v; compute $B_v=B_{left}+B_{right}$ and $HV_v=\mathsf{hash}(HV_{left}+HV_{right})$.
- For all subroots v and their father nodes, compute $B_v=B_{le}+B_{ri}$ and $HV_v=\mathsf{hash}(HV_{le}+HV_{ri})$, insert typ_i into B_i, insert the tYpe Complementary Set (YCS) to B_v and compute $\{bits_v, S_v\}$, e.g., if there are

63 types of locations, $typ_v = 20$, YCS is the encoding of "[1,19]∧[21,63]" which is $\{000001, 00001*, 0001**, 001***, 0100**, 010101, 01011*, 011***, 1*****\}$.

Finally, the data owner outsources $(\mathcal{E}, \mathcal{TR}, r)$ to the CS and shares HV_{root} with data users.

C. Trapdoor Computation

A data user has a location query Q and computes a trapdoor $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$. Here, the three sets are prepared for type, coordinates, and time, respectively.

- Extract $(typ, lat, lon, time)$ of Q, compute three prefix families $\mathcal{PF}_1$, $\mathcal{PF}_2$, and $\mathcal{PF}_3$ [19] for typ, $g(lat, lon)$, and $time$, e.g., "09:00" is encoded to $\{010010, 01001*, 0100**, 010***, 01****, 0*****, ******\}$.
- Insert $(h_o(pr), H(h_{m+1}(h_o(pr))))$ $(1 \leq o \leq m,\ pr \in \mathcal{PF}_1)$ into $\mathcal{T}_1$, insert $(h_o(pr), H(h_{m+1}(h_o(pr))))$ $(1 \leq o \leq m,$ $pr \in \mathcal{PF}_2)$ into $\mathcal{T}_2$, and insert $(h_o(pr),\ H(h_{m+1}(h_o(pr))))$ $(1 \leq o \leq m, pr \in \mathcal{PF}_3)$ into $\mathcal{T}_3$.
- Send $\mathcal{T}$ to the CS and await the query results and proofs.

D. Query Processing

We answer a query by looking up the codes in IBFs of the nodes in the TiveTree. For each node v, we check whether a query has the common element with the prefixes in v through mapping each element in the trapdoor on the IBF of v. During the query, the CS also searches proofs for three types of *evidence nodes. The idea is to locate a prefix in the trapdoor that can be queried to match a string in $bits_v$, i.e., the prefix matches a complementary set (s) of the index.*

Before we dive into the search details, we introduce three types of evidence nodes. • **Matched Leaf Node (MLN)**. The leaf nodes that correctly answer the location query. All potential query results are stored in these nodes. • **UnMatched Node (UMN)**: The unmatched non-leaf nodes and leaf nodes that do not satisfy the query. The search stops on these nodes. • **UnNecessary Node (UNN)**: The nodes that do not need to be searched when we have obtained k matched data items. The three evidence nodes are defined to classify nodes and help the CS generate verification proofs during search. Their differences reside in how they match the search conditions.

The CS searches the $\mathcal{TR}$ from top to bottom as follows.

- Set the current search node v as the root $\mathcal{TR}.root$ and set $result\_num = 0$.
- Before the search enters a subtree, check $\mathsf{Check}(\mathsf{B_v}, \mathcal{T}_1)$. 1) If $\mathsf{Check}(B_v, \mathcal{T}_1) = 1$ and v is a left node, set v's right child as UNN temporarily and search $\mathcal{T}$ along v's left subtree and then right subtree; if $\mathsf{Check}(B_v, \mathcal{T}_1) = 1$ and v is a right node, remove the previously set UNN mark and search v's subtrees. 2) If $\mathsf{Check}(B_v, \mathcal{T}_1) = 0$, i.e., among the $|\mathcal{T}_1| = z * m$ pair of hashes $\{(T^1_{ij}, T^2_{ij})\}$, for all i and one j, $1 \leq i \leq z, 1 \leq j \leq m$, $B_v[T^1_{ij}][H(T^2_{ij} \oplus r)] = 0$, mark v as UMN and stop searching.

For the UMN, we generate proofs as follows. 1) Locate the first prefix $w_i \in \mathcal{T}_1$ that satisfies $\mathsf{QueLoc}(B_v, \mathcal{T}_1) \in bits_v$. In this way, we obtain a proof that Q does not match $\mathcal{D}$ by proving that typ falls in the type complementary set. 2) Insert $\{w_i, bits_{vi}, S_{vj}, HV_v\}$ into π.

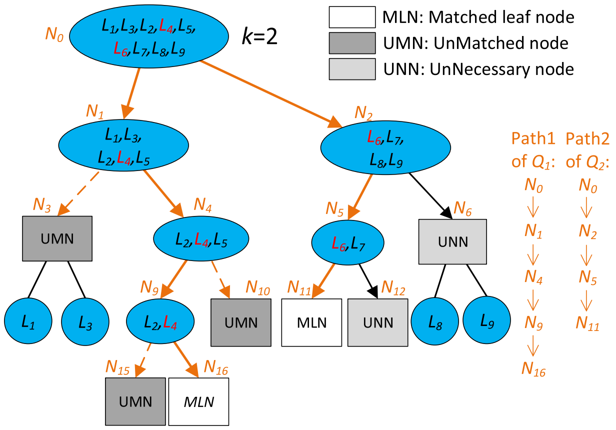


Fig. 5. Result verification tree for UMNs, MLNs, and UNNs.

- The above search recursively applies to the subtrees until the subroots, check the $\mathcal{T}_1$ the last time to decide whether to search $\mathcal{T}_1$ and $\mathcal{T}_2$ in the subtrees.
- In each subtree, a node v, check $\mathsf{Check}(B_v, \mathcal{T}_2)$ and $\mathsf{Check}(B_v,\ \mathcal{T}_3)$. There two conditions. 1) If both queries are successful: v is not a leaf node, set/remove UNN and continue to search the subtrees similarly; v is a leaf node, mark v as MLN, insert $\{HV_v\}$ into π, insert $\{E_v\}$ into result set $\mathcal{R}$, and add 1 to $result\_num$. If k data items are found, we stop the recursion and return to insert HV of all UNNs to π. 2) If there exists one unsuccessful query, mark v as UMN and stop searching, search w_i in the other trapdoor set, and update π.
- Return $(\mathcal{R}, \pi)$.

E. Result Verification

With the returned proof π, the data user uses π to verify the correctness and completeness of the result set $\mathcal{R}$.

Verifying Correctness: The data user needs to verify both whether $\mathcal{R}$ is correct and whether the CS creates $\mathcal{R}$ itself. First, the data user decrypts the $\mathcal{E}$ from $\mathcal{R}$ and checks whether her/his query matches the data items in plaintext. Second, the data user recomputes the value of the root $\mathsf{hash}'(root)$ from bottom to up based on the Merkle Tree [30] by using the hash values $\{HV\}$ of evidence nodes from π. If $\mathsf{hash}'(root)$ equals to the data owner's hash value $\mathsf{hash}(root)$, the data user is convinced that $\mathcal{R}$ is authentic and the evidence nodes are true nodes of the TiveTree.

Verifying Completeness: In query processing, a trapdoor $\mathcal{T}$ is processed from the root toward the leafs. The query process terminates until the trapdoor matches a leaf node or the trapdoor does not match the TiveTree index. Each matched data item has a search path, for which we mark the evidence nodes. Hence, using the UMNs, MLNs, and UNNs, a data user can reproduce the query process from bottom to up for each path.

As shown in Fig. 5, we use the Path 1 and Path 2 as two examples. Assume the query with $k = 2$ matches L_4 and L_6. N_1 matches the query for having L_4 and its left child node is marked as UMN for not matching the query. The search continues in the right subtree of N_1 until the node N_{16} matches the query which is marked as MLN. Finally, the Path 1 is $N_0 \rightarrow N_1 \rightarrow N_4 \rightarrow N_9 \rightarrow N_{16}$. On Path 2, we notice there are two UNNs, N_{12} and N_6. They are marked because we obtain $k = 2$ data items when the search reaches N_{11}. Therefore, we do not need to search N_{12} or N_6. For each search path, the union of UMNs, MLNs, and UNNs must reproduce the search faithfully. If the CS ignores some nodes on purpose, the result set will be incomplete.

VI. Security Analysis

We prove that the TiveQP is secure under the adaptive IND-CKA model. We use HMAC and SHA256 to implement the hash functions $h_1, h_2, \ldots, h_{m+1}$ and H, respectively. A function is a pseudo-random function if and only if the function output and the truly random function output cannot be distinguished by a Probabilistic Polynomial Time (PPT) adversary [31], [32]. A PPT adversary $\mathcal{A}$ can view its past queries and corresponding trapdoors, results, and proofs before selecting a future query in simulation.

To prove TiveQP is secure under adaptive IND-CKA model, we first construct a PPT simulator $\mathcal{S}$ that can simulate future queries. Next, we show that $\mathcal{A}$ which interacts with $\mathcal{S}$ is challenged to distinguish between the real secure index and ones from $\mathcal{S}$ with a non-negligible probability. Formally, a query processing scheme is secure if a PPT adversary $\mathcal{A}$ cannot distinguish the a real index generated by pseudo-random functions from a simulated index generated by truly random functions, with a non-negligible probability:

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

where negl(λ) is a negligible function. We define two leakage functions as follows:

- $\mathcal{L}_1(\mathcal{D}) = (n, N, \mathcal{TR}, |E|)$: Given the location dataset $\mathcal{D}$, $\mathcal{L}_1$ outputs the dataset size n, the IBF bit length N, the TiveTree $\mathcal{TR}$, and the ciphertext bit length $|E|$.
- $\mathcal{L}_2(\mathcal{D}, Q) = (\alpha(Q), \beta(Q), \gamma(Q))$: Given a location dataset $\mathcal{D}$ and a location query Q, $\mathcal{L}_2$ outputs the data item id returned by a query $\alpha(Q)$, the search pattern $\beta(Q)$, and the path pattern $\gamma(Q)$.

Theorem 1 (Security): TiveQP is IND-CKA $(\mathcal{L}_1, \mathcal{L}_2)$-secure in the random oracle model against an adaptive $\mathcal{A}$.

Proof: We first construct $\mathcal{S}$ that simulates a view $V^* = (\mathcal{TR}^*, \mathcal{T}^*, \mathcal{E}^*)$ based on the information returned by $\mathcal{L}_1(\mathcal{TR}, \mathcal{D})$ and $\mathcal{L}_2(\mathcal{TR}, \mathcal{D}, Q)$. Next, we show that $\mathcal{A}$ cannot distinguish between V^* and the real adversary view A.

- To simulate $\mathcal{TR}^*$, $\mathcal{S}$ first builds an identically structured TiveTree. $\mathcal{S}$ acquires N from $\mathcal{L}_1$ and sets up an IBF B_v for each node v in the $\mathcal{TR}$. In the ith cell of B_v, $\mathcal{S}$ sets $B_v[i][0] = 1$ and $B_v[i][1] = 0$ or vice versa. The twin is determined by tossing a coin. Next, the $\mathcal{S}$ associates B_v with a randomly chosen number r. For the verification information, $\mathcal{S}$ randomly chooses a grid for each leaf node, computes corresponding LCS and TCS, and generates $bits_v$ and S_v. $\mathcal{S}$ continues this steps until the subtree nodes and computes YCS for each node until the root. Finally, $\mathcal{S}$ returns $\mathcal{TR}^*$ to the $\mathcal{A}$. Each IBF B_v in $\mathcal{TR}^*$ has the same size as the one in $\mathcal{TR}$. Their '0's and '1's are equally distributed. Therefore, the $\mathcal{A}$ cannot distinguish the simulated $\mathcal{TR}^*$ from the real $\mathcal{TR}$.

TABLE III
Parameter Settings (Bold: Default Values)

Parameter	Value
Number of data items n	**20000** 40000 60000 80000 100000
Location type t	**20** 40 60 80 100
Grid width gw (km)	**1** 2 3 4 5
Query parameter k	1 5 **10** 15 20
$\lvert sk\rvert$, $\lvert sk'\rvert$, $\lvert k_i\rvert$, $\lvert r\rvert$	1024
m	5
FPR	1%

1: t corresponds with n.

- To simulate $\mathcal{T}^*$, $\mathcal{S}$ knows if a received Q has been processed from L_2. If so, $\mathcal{S}$ returns the previous trapdoor $\mathcal{T}$ to $\mathcal{A}$. Otherwise, $\mathcal{S}$ generates a new trapdoor $\mathcal{T}^*$ that is a set of m-pair of hashes. Given the access pattern from $\mathcal{L}_2$, $\mathcal{S}$ knows which data items match $\mathcal{T}$. For the MLF, $\mathcal{S}$ generates the output by using H to select e-pair of hashes while satisfying that the selected hash pairs match the MLF. For the unmatched leaf node, $\mathcal{S}$ generates the output by using the random oracle to mismatch the $\mathcal{T}$ with the leaf node. The e-pair of hashes is $\mathcal{T}^*$. Since the trapdoor is generated by the random hash functions, $\mathcal{A}$ cannot distinguish $\mathcal{T}^*$ from the real trapdoor.
- To simulate $\mathcal{E}^*$, $\mathcal{S}$ first acquires n and $|E|$ from $\mathcal{L}_1$. Next, $\mathcal{S}$ simulates the ciphertext set with random plaintexts and the known CPA-secure encryption algorithm Enc. $\mathcal{S}$ has to make sure that the size of the simulated ciphertext is the same as the one of the real ciphertext.

To sum up, the simulated view and the real view are indistinguishable by $\mathcal{A}$. Therefore, TiveQP is adaptive IND-CKA $(\mathcal{L}_1, \mathcal{L}_2)$-secure in the random oracle model against an adaptive adversary, i.e., the index privacy and query privacy are achieved. Furthermore, the CS does not return the IBF of leaf node to the data user as [21] did but only returns one bit string and one HMAC. The data owner can only validate the mismatching of query, but cannot insert the trapdoor of other queries on the IBF. Therefore, result privacy is achieved. □

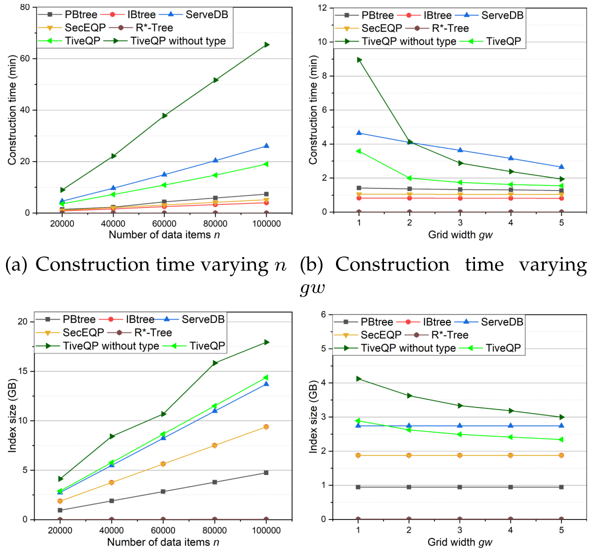
VII. Performance Evaluation

In this section, we introduce our experiment settings (dataset, parameters, baselines, and setup) and analyze the performance of each phase in detail.

A. Experiment Settings

Dataset: We use a dataset of Yelp's businesses and user data from 836 cities in the US and Canada [33], [34]. We upload all source codes, processed datesets, and an instruction file to https://github.com/UbiPLab/TiveQP. Given that the size of the original dataset is too large (4.9 GB), we only provide a link.

Parameters: We vary n from 10,000 to 100,000, location type t from 20 to 100, gird width gw from 1 to 5 km, and k from 1 to 50. According to the FPR equation [19], [35], the resulting size of an IBF ranges from 24 to 120 KB. The detailed parameter setting is listed in Table III. The default value of n is set to 20000



(a) Construction time varying n (b) Construction time varying gw

(c) Index size varying n (d) Index size varying gw

Fig. 6. Performance of tree construction.

after we review the experimental settings of state-of-the art and it is a relatively high value for query processing.

Baselines: To evaluate the query performance of secure multi-dimension query framework, we compare TiveQP with six baselines: 1) PBtree [19] supports range query on single dimensional data. 2) IBtree [20] achieves conjunctive query processing by using IBFs. 3) SecEQP [17] is based on projection-based functions and IBFs to encode locations. 4) ServeDB [21] facilitates multi-dimensional and verifiable range query. 5) R*-Tree [36] is a dynamic tree for indexing spatial information. 6) TiveQP without type is similar to TiveQP but it has not type in the data items.

Setup: We implemented TiveQP in Java and conducted experiments on a PC server running Windows Server 2021 R2 Datacenter with a 3.7-GHz Intel(R) Core(TM) i7-8770 K processor and 32 GB RAM. We used HMAC-SHA256 as the pseudo-random function to implement the hash functions of IBF. We used AES as the symmetric encryption algorithm. Since AES can be used for TiveQP and other schemes when encrypting data items, we focus on the index and remove the encryption results in comparison.

B. Tree Construction

With n increasing from 20 to 100 K and gw increasing from 1 to 5 km, the construction time of TiveQP grows from 3.58 to 19 minutes, and decreases from 3.58 to 1.54 minutes, respectively. In Fig. 6(a) and (b), the tree size, w.r.t. different n and gw, grows from 2.89 to 14.37 GB, and decreases from 2.89 to 2.34 GB, respectively.

It is obvious that the construction costs increase linearly with n. The costs reduce with gw because each grid covers more space and the number of grids decreases when gw becomes bigger, thus reducing the size of minimum set of prefixes $\mathcal{MS}$ and the size of location complementary sets. The effect of gw on tree construction is small since the total size of $bits$ and Hv is much shorter than the IBFs.

We notice that the leftmost green dot drops rapidly to the next one in Fig. 6(b). This is because when gw increases from 1 to 2, the size difference of the two corresponding complementary sets is bigger than the ones after, leading to more hash operations.

The generation of verification information dominate the construction time because we need to compute the complementary sets of each node and then compute $bits$ and Hv. Such computations consume more time than the hash operations of IBFs on leaf nodes.

The size of an IBF corresponds to the total number of prefixes inserted to the IBF. The IBFs dominate the size of the tree. For example, when $n = 20,000$ and $gw = 1$ km, the tree size is 2.893 GB while the size of IBFs is 1.863 GB. This is because 1) an IBF has a long size in order to maintain the FPR give fixed n and m; 2) each of the three complementary sets for a leaf node produces a small set of prefix family that leads to a small communication overhead, and the three complementary sets will become smaller as the nodes merge upward; 3) each string in $bits$ has m bits with an identifier, both the elements in the HMAC set S_v and the hash value HV are 256 bits. Experimental results show that the construction cost of TiveTree is acceptable.

C. Query Processing

The time complexity of query processing is $O(k\log(\frac{n}{tc}))$.

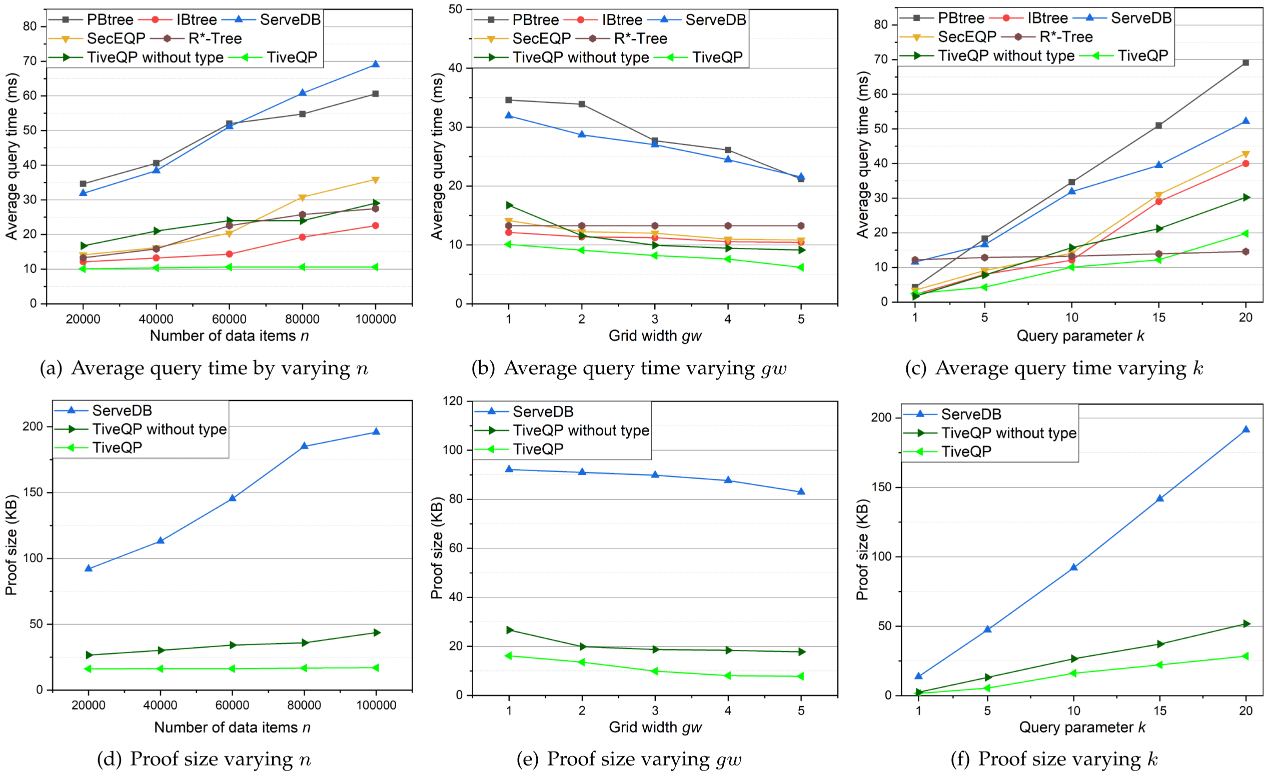
Average time: Fig. 7(a) to (c) show the query delays w.r.t. varied n, gw, and k, respectively. Experimental results indicate that the query processing time is in ms scale.

1) Varying n. When $gw = 1$ and $k = 10$, the average query processing time of TiveQP requires around 10 milliseconds. It does not increase with n because 1) we use all data items of queried type in the five sets of experiments such that their total number stays unchanged and 2) the desired data items are arranged closer on the leaf level after we organize the tree according to type and city, i.e., they are placed in a small subtree from the overall structure of the index tree. **2) Varying gw.** When $n = 20,000$ and $k = 10$, with the grid width increasing from 1 to 5 km, the average query processing time decreases from 10.1 to 6.2 milliseconds. This happens because as w increases, the prefix number in the location trapdoor becomes less, which leads to less hash operations and then less search time. **(3) Varying k.** When $n = 20,000$ and $gw = 1$, with k increasing from 1 to 20, the average query processing time grows from 2.5 to 19.8 milliseconds. This is obvious since more queries will lead to more search paths and time.

Communication Overhead: Fig. 7(d)–(f) show the communication overhead w.r.t. varied n, gw, and k, respectively. The communication overhead does not change much with n either because for the same reason in query processing. When gw increases, the grid number decreases and the size of grid codes decreases, thus reducing the communication overhead.

D. Result Verification

The data user verifies the correctness and completeness of the result. We record the average verification time w.r.t. n, gw, and k. Fig. 8(a)–(c) show that the time is almost constant with varied n and gw. We attribute this advantage, i.e., near "immunity" to



(a) Average query time by varying n

(b) Average query time varying gw

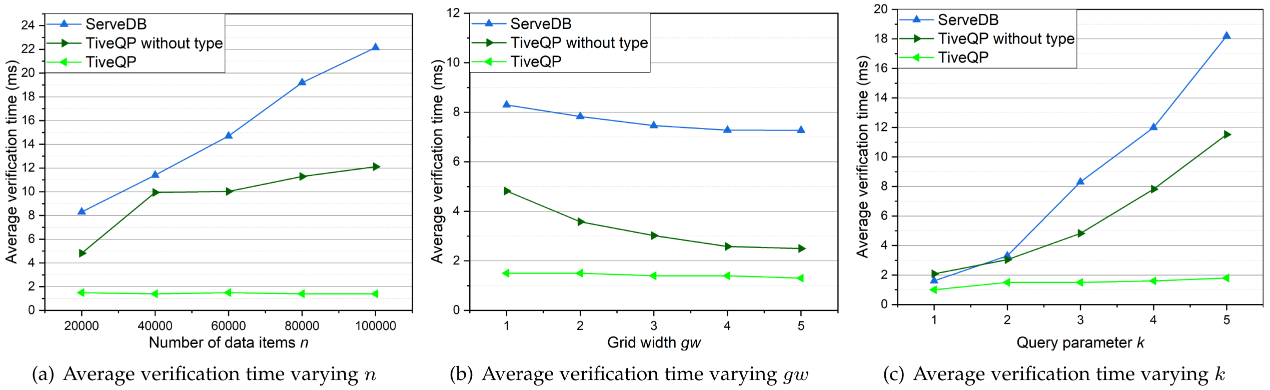
(c) Average query time varying k

(d) Proof size varying n

(e) Proof size varying gw

(f) Proof size varying k

Fig. 7. Performance of query processing.



(a) Average verification time varying n

(b) Average verification time varying gw

(c) Average verification time varying k

Fig. 8. Performance of result verification.

n and gw, to the method that TiveQP builds the index tree after careful structure organization. The time grows slightly with k because the user has to check more proofs from more search paths.

E. Comparison

In **index construction**, PBtree has a lower cost because it 1) uses a Bloom filter as index leading to less hash operations (only on leaf nodes) and half index size, 2) avoids logical OR operations when building the tree upwards, and 3) does not compute any verification information. The costs of IBtree and SecEQP are lower for not generating verification information. IBtree and SecEQP have an index tree of the same size for using the same IBF. ServeDB costs more than TiveQP because it uses multiple levels to encode locations, inserts data items into a Bloom filter for each node, and computes a HMAC for each Bloom filter segment of each node, thus involving more hashes. ServeDB's index size is larger than TiveQP's (after $gw = 1$) because its verification information contains many HMACs, even using Bloom filter as index. For example, when $n = 20,000$, the size of HMACs sums to $(20000 * 2 - 1) * N/200 * 256\text{bits} = 1.192$ GB. The construction cost of TiveQP without type is higher than TiveQP because it computes and contains more pairs of $bits$ and S for non-leaf nodes above the subtrees. The construction cost of R-Tree is the lowest because in only uses five-dimension points to create node indexes when inserting data items. However, it reveals data privacy. We did not introduce the update and delete property which the existing work (PBTree, IBTree, SecEQP, and ServeDB) did not mention either. All the trees support the two properties by recomputing a new leaf node and its father nodes and deleting a leaf node and updating its father nodes. Compared with other trees, TiveTree is built in a bottom-up fashion as IBtree and SecEQP. Each node in the TiveTree has an IBF index. All nodes in TiveTree contain verification information in each node for result verification as in ServeDB. In **query processing**, PBtree, IBtree, SecEQP, and ServeDB spend more time for two reasons. First, they process three types of data: type, location, and time. Second, they randomly partition the data items without considering the spatial attributes of data items when building the index. This in turn causes more search paths than TiveQP, thus resulting in more hash operations and more query processing time. Specifically, PBTree's query time exceeds other schemes for using one HMAC and one modular operation in processing each prefix. SecEQP's query time is higher than the one of IBtree for using multiple coordinate systems. ServeDB has extra query time for using multiple levels to encode locations that involves more hash checks on Bloom filters. It has a higher communication cost than TiveQP for transmitting more proofs that include the Bloom filter and HMACs of each key node. The query time of R*-Tree is higher than TiveQP because the search on multi-dimension points spends more time than IBF hashes.

In **result verification**, ServeDB consumes more time because its data user 1) verifies the correctness by using Merkle tree, 2) verifies the correctness of Bloom filter for each key node, 3) re-maps the trapdoors for each key node to verify the correctness of matched trapdoor set UMT and matched trapdoor set MT, and 4) verifies the query process in each search path. Specifically, the second step involves many Bloom filter hashes and checking HMACs.

The cost of TiveQP without type is always higher than TiveQP for computing extra LCS in construction and performing additional checks on location and time above subtrees in querying. Comparison shows that TiveQP provides an advantage in query processing and result verification. The construction is an one-time and offline process, which will not impact the efficiency greatly. The accuracy of TiveQP is the same as IBTree and SecEQP for using IBF as index. Since it is well studied and the page numbers are limited, we do not provide accuracy comparison. We do not compare with SVkNN because 1) We are different in index, system model, and techniques; 2) SVkNN requires two servers to execute secure grid computation and Paillier encryptions/decryptions, resulting in 560 seconds ($n = 2000, k = 20$) while ours is 10 ms ($n = 20000, k = 20$).

VIII. Discussions

In this section, we discuss two relevant issues, namely local processing and time period.

A. Complementary Set

The complementary sets of location, time, and type are proposed to verify query results without violating the privacy of other data items. After the data owner uses space encoding and prefix encoding techniques to calculate the complementary sets of location, time, and type, the complementary sets are inserted into the IBF for data users to verify during the validation phase. Therefore, the complementary information will not be stored, but the inserted IBF will be sent to an untrusted cloud server and stored for verification.

The differences in applying the complementary set to location, time, and type is preprocessing. In specific, they have different format that leads to difference preprocessing operations to unify them into an appropriate format for prefix encoding.

B. Local Processing

It is feasible for the CS not to process the time query but the data user does it locally. However, this results in increased search time, inaccurate results, and poor service experience. Further, the time feature is already adopted in current services like OpenTable (https://www.opentable.ca).

C. Time Period

Regarding splitting time period, we can process opening hours and access time separately by transforming them into two sets, and then match them via private set intersection. Specifically, the opening hours of stores are relatively fixed while the access time of users are dynamic. Therefore, the treatment of the two time is somehow different. We can apply the splitting technique to a wider range of application scenarios. For example, a user needs k nearest cafes that are rated over three stars and a rider hails a ride with a specific time to meet the driver.

The current design considers a general accessing hour (e.g., a store opens every day from "8:00" to "12:00"). However, in real life, the accessing time of locations can be different for dates. Therefore, we can revise the coding of opening time and access time according to the specific scenarios.

D. Result Deletion

If an untrusted cloud server intentionally deletes some correct results, these correct results will be marked as UMN rather than MLN. When users verify the completeness, they will reconstruct the search path for each matched result and verify each evidence node include UMN in the search path. Such UMNs will not pass verification.

IX. Conclusion

We have proposed a time-restricted, verifiable, and efficient SkNN query processing scheme TiveQP. Two key novelties are fusing the feature of time-restricted access into SkNN query processing by integrating spatial attribute with time attribute, and designing a privacy-preserving verification method by leveraging membership checking in complementary sets. We design a space encoding technique and a prunning strategy to improve query efficiency. We formally state and prove the security of TiveQP. Experimental results show that TiveQP is highly efficient in query processing and result verification.

References

[1] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: Anonymizers are not necessary," in *Proc. 34th ACM SIGMOD Int. Conf. Manage. Data*, Vancouver, Canada, 2008, pp. 121–132.

[2] 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 Secure Comput.*, vol. 19, no. 2, pp. 1133–1148, Mar./Apr. 2022, doi: 10.1109/TDSC.2020.3017534.

[3] M. Li, Y. Chen, C. Lal, M. Conti, M. Alazab, and D. Hu, "Eunomia: Anonymous and secure vehicular digital forensics based on blockchain," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 225–241, Jan./Feb. 2023, doi: 10.1109/TDSC.2021.3130583.

[4] 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. Netw.*, vol. 30, no. 5, pp. 2060–2073, Oct. 2022, doi: 10.1109/TNET.2022.3157654.

[5] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks onsecure outsourced databases," in *Proc. 23rd ACM Conf. Comput. Commun. Secur.*, Vienna, Austria, 2016, pp. 1329–1340.

[6] B. Wang, Y. Hou, and M. Li, "Practical and secure nearest neighbor search on encrypted large-scale data," in *Proc. IEEE 35th Annu. Int. Conf. Comput. Commun.*, San Francisco, USA, 2016, pp. 1–9.

[7] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia, "Data recovery on encrypted databases with k-nearest neighbor query leakage," in *Proc. IEEE 40th Symp. Secur. Privacy*, San Francisco, USA, 2019, pp. 1033–1050.

[8] L. Ou, Z. Qin, S. Liao, J. Weng, and X. Jia, "An optimal noise mechanism for cross-correlated IoT data releasing," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 4, pp. 1528–1540, Jul./Aug. 2021.

[9] L. Ou, Z. Qin, S. Liao, Y. Hong, and X. Jia, "Releasing correlated trajectories: Towards high utility and optimal differential privacy," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 5, pp. 1109–1123, Sep./Oct. 2020.

[10] L. Ou, Z. Qin, S. Liao, T. Li, and D. Zhang, "Singular spectrum analysis for local differential privacy of classifications in the smart grid," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5246–5255, Jun. 2020.

[11] J. Wakefield, "Location data collection firm admits privacy breach," BBC, 2021. [Online]. Available: https://www.bbc.com/news/technology--59063766

[12] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proc. 35th ACM SIGMOD Int. Conf. Manage. Data*, Providence, USA, 2009, pp. 139–152.

[13] H. Hu, J. Xu, C. Ren, and B. Choi, "Processing private queries over untrusted data cloud through privacy homomorphism," in *Proc. IEEE 27th Int. Conf. Data Eng.*, Hannover, Germany, 2011, pp. 601–612.

[14] B. Yao, F. Li, and X. Xiao, "Secure nearest neighbor revisited," in *Proc. 29th IEEE Int. Conf. Data Eng.*, Brisbane, Australia, 2013, pp. 733–744.

[15] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environment," in *Proc. IEEE 30th Int. Conf. Data Eng.*, Chicago, USA, 2014, pp. 664–675.

[16] X. Yi, R. Paulet, E. Bertino, and V. Varadharajan, "Practical k nearest neighbor queries with location privacy," in *Proc. IEEE 30th Int. Conf. Data Eng.*, Chicago, USA, 2014, pp. 640–651.

[17] 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," in *Proc. IEEE 35th Int. Conf. Data Eng.*, Macao, China, 2019, pp. 662–673.

[18] N. Cui, X. Yang, B. Wang, J. Li, and G. Wang, "SVkNN: Efficient secure and verifiable k-nearest neighbor query on the cloud platform," in *Proc. IEEE 36th Int. Conf. Data Eng.*, Dallas, USA, 2020, pp. 253–264.

[19] R. Li, A. Liu, A. L. Wang, and B. Bruhadeshwar, "Fast range query processing with strong privacy protection for cloud computing," in *Proc. 40th Int. Conf. Very Large Data Bases*, Hangzhou, China, 2014, pp. 1953–1964.

[20] R. Li and A. X. Liu, "Adaptively secure conjunctive query processing over encrypted data for cloud computing," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, San Diego, USA, 2017, pp. 697–708.

[21] S. Wu, Q. Li, G. Li, D. Yuan, X. Yuan, and C. Wang, "ServeDB: Secure, verifiable, and efficient range queries on outsourced database," in *Proc. IEEE 35th Int. Conf. Data Eng.*, Macao, China, 2019, pp. 626–637.

[22] C. Guo, W. Li, X. Tang, K.-K. R. Choo, and Y. Liu, "Forward private verifiable dynamic searchable symmetric encryption with efficient conjunctive query," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2023.3262060.

[23] C. Guo, W. Liu, X. Liu, and Y. Zhang, "Secure similarity search over encrypted non-uniform datasets," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 2102–2117, Third Quarter 2022.

[24] C. Guo, X. Chen, Y. Jie, Z. Fu, M. Li, and B. Feng, "Dynamic multi-phrase ranked search over encrypted data with symmetric searchable encryption," *IEEE Trans. Serv. Comput.*, vol. 13, no. 6, pp. 1034–1044, Nov./Dec. 2020.

[25] Y. Zhang et al., "Anonymous multi-hop payment for payment channel networks," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2023.3262681.

[26] X. Jia, Z. Yu, J. Shao, R. Lu, G. Wei, and Z. Liu, "Cross-chain virtual payment channels," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 3401–3413, 2023.

[27] M. Li, Y. Chen, C. Lal, M. Conti, F. Martinelli, and M. Alazab, "Nereus: Anonymous and secure ride-hailing service based on private smart contracts," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 4, pp. 2849–2866, Jul./Aug. 2023.

[28] A. X. Liu and F. Chen, "Collaborative enforcement of firewall policies in virtual private networks," in *Proc. 27th ACM Symp. Princ. Distrib. Comput.*, Toronto, Canada, 2008, pp. 95–104.

[29] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, Virginia, USA, 2006, pp. 79–88.

[30] M. Szydlo, "Merkle tree traversal in log space and time," in *Proc. 10th Int. Conf. Theory Appl. Cryptographic Techn.*, Interlaken, Switzerland, 2004, pp. 541–554.

[31] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," in *Proc. 38th Annu. Symp. Foundations Comput. Sci.*, Miami Beach, USA, 1997, pp. 458–467.

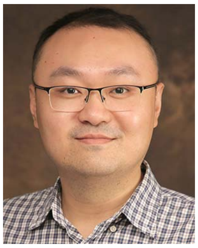
[32] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 2nd ed. Boca Raton, FL, USA: Chapman & Hall/CRC, 2015.

[33] "Yelp open dataset," 2023. [Online]. Available: https://www.yelp.com/dataset

[34] "Yelp dataset," 2023. [Online]. Available: https://www.kaggle.com/yelp-dataset/yelp-dataset

[35] B. H. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Commun. ACM*, 1970, vol. 13, no. 7, pp. 422–426.

[36] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Atlantic City, USA, 1990, pp. 322–331.



Meng Li (Senior Member, IEE) received the PhD degree in computer science and technology from the School of Computer Science and Technology, Beijing Institute of Technology (BIT), China, in 2019. He is an associate professor and dean assistant with the School of Computer Science and Information Engineering, Hefei University of Technology (HFUT), China. He is also a postdoc researcher with the Department of Mathematics and HIT Center, University of Padua, Italy, where he is with the Security and PRIvacy Through Zeal (SPRITZ) research group led by Prof. Mauro Conti (IEEE fellow). He was sponsored by ERCIM 'Alain Bensoussan' Fellowship Programme (from 2020 to 2021) to conduct postdoc research supervised by Prof. Fabio Martinelli with CNR, Italy. He was sponsored by China Scholarship Council (CSC) (from 2017 to 2018) for joint Ph.D. study supervised by Prof. Xiaodong Lin in the Broadband Communications Research (BBCR) Lab, University of Waterloo and Wilfrid Laurier University, Canada. His research interests include security, privacy, fairness, applied cryptography, cloud computing, edge computing, blockchain, and vehicular networks. In this area, he has published more than 60 papers in international peer-reviewed transactions, journals, magazines, and conferences, including *IEEE Transactions on Dependable and Secure Computing*, *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Vehicular Technology*, *IEEE Transactions on Smart Grid*, *IEEE Transactions on Industrial Informatics*, *IEEE Transactions on Network and Service Management*, *IEEE Transactions on Network Science and Engineering*, *IEEE Transactions on Green Communications and Networking*, *IoT Journal*, *Information science*, *Future Generation Computer Systems*, *IEEE Communications Magazine*, *IEEE Wireless Communications*, MobiCom, ICICS, SecureComm, TrustCom, and IPCCC. He is an associate editor of *IEEE Transactions on Information Forensics and Security* and *IEEE Transactions on Network and Service Management*.

Jianbo Gao (Student Member, IEEE) received the BE degree from Anhui Medical University, in 2020, and the MS degree from the School of Computer Science and Information Engineering, Hefei University of Technology, in 2023. He is currently working toward the PhD degree with the School of Cyberspace Science and Technology, Beijing Institute of Technology. His research interests include security, privacy, applied cryptography, and vehicular networks.

Liehuang Zhu (Senior Member, IEEE) is currently a professor, secretary with the School of Cyberspace Science and Technology, Beijing Institute of Technology. He has published more than 100 peer-reviewed journal or conference papers, including more than 50 IEEE/ACM Transactions papers. He has been granted a number of IEEE Best Paper Awards, including IWQoS 17', TrustCom 18'. His research interests include security protocol analysis and design, wireless sensor networks, and cloud computing.

Zijian Zhang received the PhD degree from the School of Computer Science and Technology, Beijing Institute of Technology. He is now an associate professor with the School of Cyberspace Science and Technology, Beijing Institute of Technology. He was a visiting scholar with the Computer Science and Engineering Department, State University of New York at Buffalo in 2015. His research interests include design of authentication and key agreement protocol and analysis of entity behavior and preference.

Chhagan Lal received the PhD degree in computer science and engineering from the Malaviya National Institute of Technology, Jaipur, India, in 2014. He is currently working as a postdoctoral research fellow with the Delft University of Technology, The Netherland. Previously, he was a postdoctoral fellow with the Department of Mathematics, University of Padua, Italy, where he was part of the SPRITZ research group. He was a postdoctoral research fellow with the Simula Research Laboratory, Norway. His current research interests include applications of blockchain technologies, security in software-defined networking, and Internet of Things networks.

Mauro Conti (Fellow, IEEE) received the PhD degree from the Sapienza University of Rome, Italy, in 2009. He is full professor with the University of Padua, Italy. He is also affiliated with TU Delft and University of Washington, Seattle. After his PhD, he was a postdoc researcher with Vrije Universiteit Amsterdam, The Netherlands. In 2011 he joined as assistant professor with the University of Padua, where he became associate professor in 2015, and full professor in 2018. He has been visiting researcher with GMU, UCLA, UCI, TU Darmstadt, UF, and FIU. He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a Fellowship by the German DAAD (2013). His research is also funded by companies, including Cisco, Intel, and Huawei. His main research interest include the area of Security and Privacy. In this area, he published more than 400 papers in topmost international peer-reviewed journals and conferences. He is editor-in-chief for *IEEE Transactions on Information Forensics and Security*, area editor-in-chief for *IEEE Communications Surveys & Tutorials*, and has been associate editor for several journals, including *IEEE Communications Surveys & Tutorials*, *IEEE Transactions on Dependable and Secure Computing*, and *IEEE Transactions on Network and Service Management*. He was program chair for TRUST 2015, ICISS 2016, WiSec 2017, ACNS 2020, CANS 2021, and general chair for SecureComm 2012, SACMAT 2013, NSS 2021, and ACNS 2022. He is senior member of the ACM, and fellow of the Young Academy of Europe.