7-2020

Search me in the dark: Privacy-preserving Boolean range query over encrypted spatial data

Xiangyu WANG
Xidian University

Jianfeng MA
Xidian University

Ximeng LIU
Fuzhou University

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Yinbin MIAO
Xidian University

See next page for additional authors

Citation

WANG, Xiangyu; MA, Jianfeng; LIU, Ximeng; DENG, Robert H.; MIAO, Yinbin; ZHU, Dan; and MA, Zhuoran. Search me in the dark: Privacy-preserving Boolean range query over encrypted spatial data. (2020). *2020 38th IEEE Conference on Computer Communications, INFOCOM: Toronto, Canada; July 6-9: Proceedings*. 2253-2262.

Available at: https://ink.library.smu.edu.sg/sis_research/5923

Author
Xiangyu WANG, Jianfeng MA, Ximeng LIU, Robert H. DENG, Yinbin MIAO, Dan ZHU, and Zhuoran MA

Search Me in the Dark: Privacy-preserving Boolean Range Query over Encrypted Spatial Data

Xiangyu Wang*†, Jianfeng Ma*†, Ximeng Liu‡, Robert H. Deng§, Yinbin Miao*†, Dan Zhu*, and Zhuoran Ma*†
*School of Cyber Engineering, Xidian University, P.R. China
†Shaanxi Key Laboratory of Network and System Security, P.R. China
‡College of Mathematics and Computer Science, Fuzhou University, P.R. China
§School of Information Systems, Singapore Management University, Singapore
Email:*{xy_wang, zhudan, zrma}@stu.xidian.edu.cn, †{jfma, ybmiao}@xidian.edu.cn ‡x@fzu.edu.cn §robertdeng@smu.edu.sg

***Abstract*—With the increasing popularity of geo-positioning technologies and mobile Internet, spatial keyword data services have attracted growing interest from both the industrial and academic communities in recent years. Meanwhile, a massive amount of data is increasingly being outsourced to cloud in the encrypted form for enjoying the advantages of cloud computing while without compromising data privacy. Most existing works primarily focus on the privacy-preserving schemes for either spatial or keyword queries, and they cannot be directly applied to solve the spatial keyword query problem over encrypted data. In this paper, we study the challenging problem of Privacy-preserving Boolean Range Query (PBRQ) over encrypted spatial databases. In particular, we propose two novel PBRQ schemes. Firstly, we present a scheme with linear search complexity based on the space-filling curve code and Symmetric-key Hidden Vector Encryption (SHVE). Then, we use tree structures to achieve faster-than-linear search complexity. Thorough security analysis shows that data security and query privacy can be guaranteed during the query process. Experimental results using real-world datasets show that the proposed schemes are efficient and feasible for practical applications, which is at least $70\times$ faster than existing techniques in the literature.**

***Index Terms*—privacy-preserving, Boolean range queries, encrypted spatial data.**

I. Introduction

Spatial keyword search has extensive applications in location-based services, social networks, epidemic prevention, geosciences, etc., and Boolean Range Query (PBR) are fundamental search functionalities over spatial datasets. The PBR problem is, given a set of *spatio-textual* objects, and a geometric query range R associated with a set of keywords W^*, a search user aims to retrieve all objects inside R each of which contains all keywords in the W^*. For example, a user can find friends with the same interests in an area; a medical researcher can predict whether a specific virus in a specific area is at risk of an outbreak by searching patients inside the area. Many companies outsource the spatial data to public clouds to manage their data and provide spatial keyword queries services to users. However, due to the potential threats of inside attackers and hackers, the privacy of spatial datasets in public clouds should be carefully taken care of, particularly in location-based and medical applications.

Unfortunately, existing studies primarily focus on the privacy-preserving schemes for either spatial or keyword queries, which cannot be directly applied to solve the PBRQ problem. Obviously, we can simply perform secure geometric range queries and keyword Boolean queries over the same spatial data, and take intersections of their results to obtain BRQ results. However, such an approach produces a lot of useless intermediate results, resulting in a large amount of additional communication and computational overhead. For example, a *facebook* user may find thousands of persons in his town but only three of them share the same interests with him. Therefore, we need to construct a tight structure, which can effectively reduce the search space, to support BRQ over encrypted spatial data.

A. Related Work and Challenges

Geometric range queries over encrypted spatial data. In textual keyword searchable encryption schemes [6]–[10], a server needs to perform equality checking for Boolean queries or comparisons for range queries. Different from the above schemes, a geometric range query over spatial databases requires *compute-then-compare* operations. For example, to decide whether a point is inside a circle, we calculate a distance from this point to the center of a circle, and then compare this distance with the radius of the circle. In principle, Homomorphic Encryption (HE) [11] could securely evaluate *compute-then-compare* operations. However, the evaluation with HE does not directly reveal search decisions (such as inside or outside) over encrypted data, which limits its usage in search. To achieve a secure circle range query, Zhu *et. al* [2] calculated the distance using BGN [12] scheme and evaluate the results range by hash tables, however, their solution is computationally expensive due to the homomorphic operation. To avoid expensive HE, Wang *et. al* proposed several schemes [1], [3], [4] based on the Shen-Shi-Waters (SSW) encryption [13]. The SSW encryption can evaluate whether the inner product of two vectors is zero without leaking privacy. In [1], Wang *et. al* generated a number of concentric circles covering all the possible points inside a queried circle range and evaluated the points in the encrypted databases using SSW. This scheme incurs huge communication costs and cannot support arbitrary geometric range queries. In [3], Wang *et. al* represented each data point and generated geometric range

TABLE I
COMPARISON WITH PRIOR WORKS.

Scheme	CRSE [1]	SRQC [2]	GRSE [3]	FastGeo [4]	EGRQ [5]	Our PBRQ-L	Our PBRQ-Q
Faster-than-linear	✗	✗	✓	✓	✓	✗	✓
Performance	Low	Very low	Low	High	Very High	High	Very High
Search Method	Circle	Circle	Geometric	Geometric	Geometric	Geometric and BRQ	Geometric and BRQ
Security	IND-SCPA	IND-CPA	IND-SCPA	IND-SCPA	KBA	IND-SCPA	IND-SCPA

query as a Bloom filter, thus the result of an inner product of these two Bloom filters correctly indicates whether a point is inside a geometric area. However, as data or searching scope increases, a huge Bloom filter should be generated to cover all possible points in a specific querying range, which increases storage and computational cost. Then, to improve the query efficiency and accuracy, they converted spatial data and geometric range queries to equality-vector form, and leveraged a two-level search scheme to verify whether a point is inside a geometric range [4].

Unfortunately, due to the large number of pairing operations used in SSW, the performance of [4] is still limited. In addition, Xu *et. al* [5] achieved efficient geometric range queries based on secure kNN [14] and polynomial fitting technique. However, their scheme is only secure to resist Known Background Attacks (KBA), which is too weak to cope with attacks in many real-world scenarios.

Challenge: How to achieve efficient secure geometric range queries over encrypted spatial data?

Spatial keyword queries over spatial data. Spatial keyword queries have been studied for several years with the increasing popularity of geo-positioning technologies and location-based services. For the first time, Zhou et al. [15] used a hybrid index structure that integrates inverted files and R-trees to support both textual keywords and location-aware queries. The IR^2-tree [16] is another hybrid index that combines an R-tree with superimposed textual keywords signatures to support spatial keyword queries. Each node of the IR-tree [17], [18] is augmented with inverted files to support the ranking of objects based on a weighted sum of the text relevancy and spatial distance. Rocha-Junior et al. [19] proposed a novel index to improve the performance of top-k spatial keyword queries named Spatial Inverted Index (S2I) which maps each keyword to a distinct aggregated R-tree. In addition, IL-Quadtree [20] is designed based on the inverted index and the linear Quadtree to exploit both spatial and keyword based pruning techniques to effectively reduce the search space. However, none of the above schemes takes into account the problem of search over encrypted data.

Challenge: How to support secure Boolean range queries over encrypted spatial data?

B. Our Contributions

In this paper, we aim to solve the above challenging problems. The main contributions of the paper are summarized below:

1) We explore the problem of Privacy-preserving Boolean Range Query over encrypted spatial data (PBRQ), and formulate its privacy requirements.
2) We propose a basic PBRQ scheme based on Symmetric-key Hiden Vector Encryption (SHVE) [21] and space-filling curve code. Specifically, we encode the *spatio-textual* objects by Gray code [22] and bitmap, and make secure vector matches over them using SHVE to obtain BRQ results over encrypted spatial data efficiently.
3) To further improve the performance of PBRQ over large-scale datasets, we design a novel index structure called Bitmap Quadtree (BQ-tree) and propose an efficient secure prune algorithm, such that search complexity of PBRQ achieves *faster-than-linear*. In addition, we further discuss the efficiency improvement of proposed PBRQ schemes.
4) We implement the proposed schemes in JAVA, and evaluate them using various real-world datasets. The experimental results show that our schemes are practical over large-scale datasets. Especially, our final scheme is at least 70× faster than other existing works.

To the best of our knowledge, our schemes are the first to support Boolean range queries over encrypted spatial data with strong security. A comparison of our PBRQ schemes with prior works is shown in TABLE I.

II. PRELIMINARIES

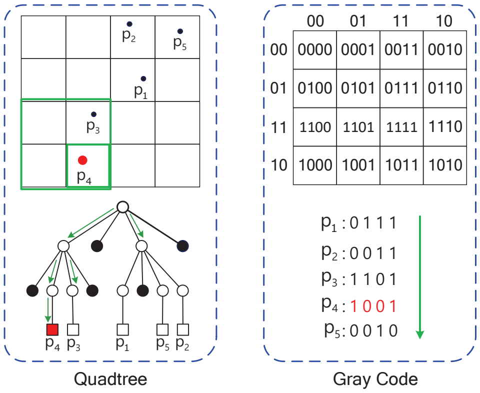
In this section, we review some background knowledge used in our work, including Quadtree [23], Gray code [22], and Symmetric-key Hidden Vector Encryption (SHVE) [21].

A. Quadtree and Gray Code

A Quadtree [23] is a space partitioning tree construct which can improve the search efficiency of spatial queries. In a Quadtree, the leaf nodes are data objects and each none-leaf node is recursively subdivided into 2^d regions, where d is the dimension of the space. Given a geometric range, the search process of Quadtree is described as follows.

1) Starting from the root node, for each child node of the current node, if the geometric range intersects with a non-leaf node, moving to search its children nodes.
2) When traverse to a leaf node (i.e., spatial data object), if the point of this leaf node insides the geometric range, add this object into the results set.

Each node of Quadtree can be encoded by Gray code [22]. A one-dimensional Gray code is denoted as follows, where '|'



	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

p1	0111
p2	0011
p3	1101
p4	1001
p5	0010

Quadtree

Gray Code

Fig. 1. An example of query over Quadtree and Grade code

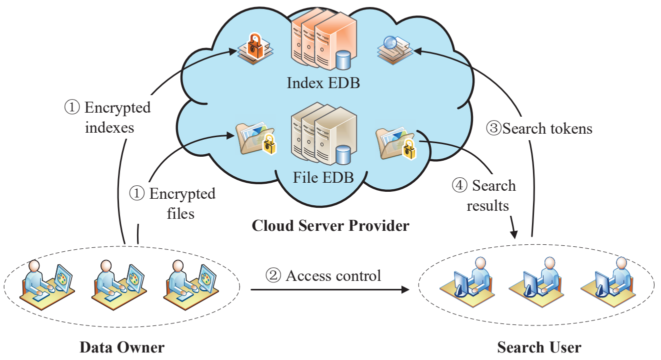


Fig. 2. System model of our schemes.

is the concatenation operator and G_μ is the vector of a Gray code instance at step μ.

$$G_1=(0,1), G_\mu=(g_1,g_2,...,g_{2^\mu}),$$
$$G_{\mu+1}=(0|g_1,0|g_2,...,0|g_{g_{2^\mu}},1|g_{g_{2^\mu}}...,1|g_{g_2},1|g_{g_1}).$$

For $\mu=3$, the Gray code is

$$G_1=(0,1), G_2=(00,01,11,10),$$
$$G_3=(000,001,011,010,110,111,101,100).$$

Given a $T\times T$ grid, the length of the required Gray code to represent all cells is $2^{\lceil \log_2 T\rceil}$. For a spatial point p, we denote $g_{\mathsf{p}} \leftarrow Gray(\mathsf{p})$ as the Gray code of p. As shown in Fig. 1, given a spatial dataset, we can build a Quadtree and encode all data points using Gray code. To find the point inside cell 1101 from the database, the server can traverse the Quadtree as described above or traverse all points using vector match.

B. Symmetric-key Hidden Vector Encryption

Symmetric-key Hidden Vector Encryption (SHVE) [21] is a predicate encryption scheme that supports conjunctive on encrypted data. Let $\Gamma\in\{0,1\}$ be a finite set of attributes and '$*$' be a wildcard symbol not in Γ, $\Gamma_*=\Gamma\cup\{*\}$. A SHVE is defined as the following four Probabilistic Polynomial-Time (PPT) algorithms:

- $\mathsf{SHVE.Setup}(1^\lambda)\rightarrow msk$: Input a security parameter λ, this algorithm returns a master secret key msk, and the massage space $\mathcal{M}$.
- $\mathsf{SHVE.KeyGen}(msk,\mathbf{v}\in\ \Gamma_*^d)\ \rightarrow\ \mathbf{s}$: Input a predicate vector $\mathbf{v}=\ \{v_1,...,v_d\}\ \in\ \Gamma_*^d$ and a master secret key msk, this algorithm returns a decryption key $\mathbf{s}$.
- $\mathsf{SHVE.Enc}(msk,\mu\in\mathcal{M},\mathbf{x}\in\Gamma^d)\rightarrow\mathbf{c}$: Input a message μ, an index vector $\mathbf{x}=\ \{x_1,...,x_d\}\in\Gamma^d$, and a master secret key msk, this algorithm returns the ciphertext $\mathbf{c}$ associated with $(\mathbf{x},\mu)$.
- $\mathsf{SHVE.Query}(\mathbf{c},\mathbf{s})\ \rightarrow\ \mu(\perp)$: Given a ciphertext $\mathbf{c}$ corresponding to the index vector $\mathbf{x}$ and a decryption key $\mathbf{s}$ corresponding to the predicate vector $\mathbf{v}$, this algorithm returns μ if $P_{\mathbf{v}}^{\mathsf{SHVE}}(\mathbf{x})=1$; otherwise returns $\perp$.

For each $\mathbf{v}\in\ \Gamma_*^d$, $\mathbf{x}\in\ \Gamma^d$, we have:

$$P_{\mathbf{v}}^{\mathsf{SHVE}}(\mathbf{x})=\begin{cases}1 & \forall\ 1\leq i\leq d,(v_i=x_i \text{ or } v_i=*)\\ 0 & \text{otherwise}\end{cases}$$

Correctness. A SHVE scheme is correct if for all security parameter λ, $(\mu,\mathbf{x})\in\ \ \mathcal{M}\ \times\ \Gamma^d$, and all predicate vectors $\mathbf{v}\in\ \Gamma_*^d$, $msk\ \leftarrow\ \mathsf{SHVE.Setup}(1^\lambda),\mathbf{s}\leftarrow \mathsf{SHVE.KeyGen}(msk,\mathbf{v}),\mathbf{c}\leftarrow\ \ \mathsf{SHVE.Enc}(msk,\mu,\mathbf{x})$, if $P_{\mathbf{v}}^{\mathsf{SHVE}}(\mathbf{x})=1$, then $\mathsf{SHVE.Query}(\mathbf{c},\mathbf{s})=\mu$; otherwise

$$\Pr[\mathsf{SHVE.Query}(\mathbf{c},\mathbf{s})=\perp]=\mathbf{1}-\mathsf{negl}(\lambda).$$

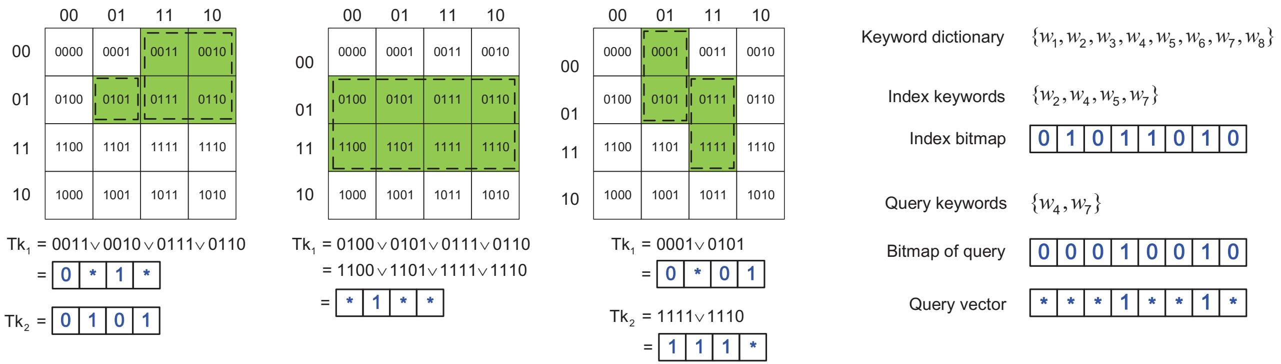
III. Problem Formulation

A. System Model and Threat Model

As shown in Fig. 2, our PBRQ consists of three main entities, namely Data Owners (**DO**), Cloud Service Provider (**CSP**) and Search Users (**SU**).

- DO encrypts privacy spatial data via symmetric encryption (i.e., AES) to protect data security and generates corresponding indexes for search according to system parameters, then he/she sends encrypted files and indexes to the CSP (Step ①).
- CSP has unlimited storage space and computation abilities that can provide data store, query, and computing services for DO or SU.
- SU receives system parameters from DO (Step ②), then he/she generates the trapdoor for his query using the parameters and sends the trapdoor to CSP (Step ③). After receiving the search results from CSP (Step ④), SU decrypts the results via the symmetric key.

Threat Model. In our schemes, DO and SU are considered to be fully trusted. CSP is a semi-honest (i.e., *honest-but-curious*) entity [1], [3]–[5], [13], [24]–[26] that provides reliable services, but it will try to learn the private information about data objects and Boolean range queries. To preserve private information leakage, DO only stores the encrypted spatial data on CSP, and a client (SU or DO) only submits the encrypted Boolean range queries to CSP.



(a) Examples of range query encoding

(b) An example of bitmap

Fig. 3. Example of spatial data object encoding

B. Definitions of PBRQ

Based on the above models, we now describe the definition of a Privacy-preserving Boolean Range Query (PBRQ) scheme. In a PBRQ scheme, the private files of DO are encrypted via symmetric encryption (i.e., AES) with secret key sk and the search process is performed using index encrypted by the master key msk. An authorized SU can generate search queries using msk to obtain search results and decrypt the search results by sk. In the rest of this paper, we no longer describe the encryption and decryption process of private files but focus on how to generate indexes and complete the query process securely.

Definition 1 (PBRQ). *A PBRQ scheme* Π *consists of four algorithms as follows:*

- $\mathsf{Setup}(1^{\lambda}) \rightarrow msk$: *Given a security parameter* λ*, this algorithm returns a master secret key* msk*.*
- $\mathsf{IndexBuild}(msk, \mathsf{DB}) \rightarrow \mathsf{EDB}$: *Given a master secret key* msk *and a spatial database* DB*, this algorithm returns the encrypted database* EDB*.*
- $\mathsf{TrapGen}(msk, Q) \rightarrow T_Q$: *Given a master key* msk*, a search query* Q*, this algorithm returns the search trapdoor* T_Q*.*
- $\mathsf{Query}(\mathsf{EDB}, T_Q) \rightarrow \mathcal{R}$: *Given a master key* msk*, an encrypted database* EDB*, and a search trapdoor* T_Q*, this algorithm returns the search results* $\mathcal{R}$*.*

Correctness. Let $\mathsf{BRQ}(\mathsf{DB}, Q) \rightarrow \mathcal{R}_q$ be the BRQ query results of a search query Q over a spatial database DB. We say that the above PBRQ scheme Π is correct if for all encrypted database EDB output by $\mathsf{IndexBuild}$, all search trapdoor T_Q output by $\mathsf{TrapGen}$, if $\mathsf{BRQ}(\mathsf{DB}, Q) \rightarrow \mathcal{R}_q$, then $\mathsf{Query}(\mathsf{EDB}, T_Q) \rightarrow \mathcal{R} = \mathcal{R}_q$.

C. Security of PBRQ

The main security requirements of PBRQ are to preserve both data privacy and query privacy from untrusted CSP, which can be informally explained as below:

- **Data Privacy.** Given the ciphertexts of two database DB_0 and DB_1, an adversary (e.g., an *honest-but-curious* cloud) cannot distinguish these two databases.
- **Query Privacy.** Given the search trapdoors of two queries Q_0 and Q_1, an adversary cannot distinguish these two queries.

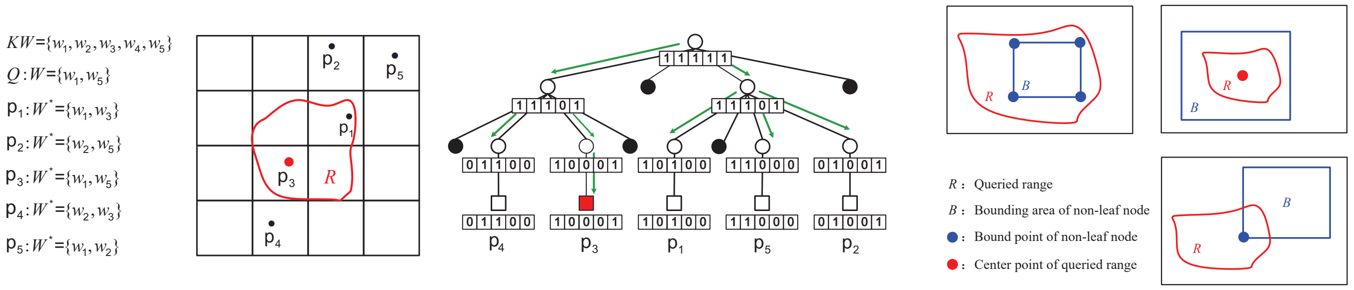
The rigorous definitions of our data privacy and query privacy with indistinguishability under Selective Chosen-Plaintext Attacks (IND-SCPA), and its corresponding leakage function are presented in Section V-A.

IV. Proposed PBRQ Schemes

In this section, we first present a PBRQ scheme with linear complexity based on Gray code and SHVE, namely PBRQ-L. Then, we design a novel index structure called Bitmap Quadtree (BQ-tree) and propose the corresponding secure prune algorithm to reduce the search complexity of PBRQ. Finally, we further discuss how to improve the efficiency of proposed PBRQ schemes using Bloom filter.

A. PBRQ-L: Linear PBRQ Scheme

Main Idea. The Boolean range queries require both geometric range queries for spatial data and Boolean keyword queries for textual keywords. The geometric range queries over plaintext spatial database requires to carry out *compute-then-compare* operations. As we described in Section I-A, the encryption primitives (i.e., HE) supporting this type of operation are too expensive. To achieve efficient geometric range queries over encrypted spatial databases, we encode the spatial point p as Gray code $g_{\mathsf{p}} \leftarrow Gray(\mathsf{p})$. Then, we design a new geometric range queries algorithm adapts to existing lightweight encryption primitives, where only SHVE is needed for the evaluation over encrypted databases. Our new geometric range queries algorithm is based on the Gray code of the queried spatial area: Given a geometric range query, we can generate all cells inside the range as search tokens and match all spatial points in the database. If the Gray code of a spatial point matches anyone of search tokens, it falls inside the geometric range; otherwise, it is not inside the geometric range. Particularly, if the code of two cells differs in just one bit, then a wildcard '$*$' can be used instead of that bit to minimize the number of search tokens. Fig. 3(a) shows some examples of token generation, for example, to query all objects inside cells (0011,



$KW=\{w_1,w_2,w_3,w_4,w_5\}$
$Q:W=\{w_1,w_5\}$
$\mathsf{p}_1:W^*=\{w_1,w_3\}$
$\mathsf{p}_2:W^*=\{w_2,w_5\}$
$\mathsf{p}_3:W^*=\{w_1,w_5\}$
$\mathsf{p}_4:W^*=\{w_2,w_3\}$
$\mathsf{p}_5:W^*=\{w_1,w_2\}$
p2
p5
p1
p3
R
p4
11111
11101
11101
01100
10001
10100
11000
01001
p4
p3
p1
p5
p2
R
B
R
B
B
R
R : Queried range
B : Bounding area of non-leaf node
● : Bound point of non-leaf node
● : Center point of queried range

(a) An example of BQ-tree

(b) All possible situations where a query intersects with a non-leaf node

Fig. 4. An example of BQ-tree and all possible situations where a query intersects with a non-leaf node

0010, 0101, 0111, 0110), the SU generate search tokens as $\mathsf{tk}_1 = 0011 \vee 0010 \vee 0101 \vee 0111 \vee 0110 = 0{*}1{*}, \mathsf{tk}_2 = 0101$, such that all cells are in consideration. In this way, the *compute-then-compare* operations are translated into vector match operations, which are available for SHVE. In addition, we encode keyword set W of each spatial object as a bitmap, such that Boolean keyword queries can also be achieved using vector match. Let KW be the textual keyword dictionary contained in the database and $|KW|$ be the size of KW. As shown in Fig. 3(b), the textual keywords contained in each object can be represented using a bitmap of size $|KW|$, with '1' indicating its existence in the objects and a '0' otherwise. For example, a bitmap $b = 01011010$ reveals that there are eight keywords in the database and the current object only contains the keywords in the second, fourth, fifth and seventh positions of the bitmap. To query keywords w_4, w_7, the SU generates the query as $b' = {*}{*}{*}1{*}{*}1{*}$.

Therefore, to construct an efficient PBRQ scheme, for each spatial objects $D_i = \{\mathsf{p}, W\} \in \mathsf{DB}$, we encode the spatial point p as Gray code $g_\mathsf{p} \leftarrow Gray(\mathsf{p})$ and the keywords W as bitmap b_W. Then, we generate the index vector of each spatial objects as $g_\mathsf{p}|b_W$ and encrypt it by $\mathsf{SHVE.Enc}$. To perform Boolean range queries over EDB, the SU first generates search tokens according to Gray code and bitmap, and then encrypts the tokens by $\mathsf{SHVE.KeyGen}$. The query process is the vector match of index and tokens, which performed by $\mathsf{SHVE.Query}$.

Scheme Details. Based on the above idea, we describe the details of our PBRQ-L scheme as follows.

- $\mathsf{Setup}(1^\lambda) \rightarrow msk$: Given a security parameter λ, DO computes and outputs a master key
$$msk \leftarrow \mathsf{SHVE.Setup}(1^\lambda).$$
- $\mathsf{IndexBuild}(msk, \mathsf{DB})$: Given a master key msk and a spatial database $\mathsf{DB} = \{D_1, ..., D_i, ..., D_n\}(1 \leq i \leq n)$, where $D_i = \{\mathsf{p}, W\}, \mathsf{p} \in \Delta_T^d, W \in KW$, T is the plaintext space of spatial data, and d is the dimension of spatial data. For each $D_i = \{\mathsf{p}, W\} \in \mathsf{DB}$, DO encodes the p as $g_\mathsf{p} \leftarrow Gray(\mathsf{p})$[1] and generates the bitmap of W as b_W. Then, DO calculates
$$\mathsf{ED} \leftarrow \mathsf{SHVE.Enc}(msk, \mathsf{ture}, g_\mathsf{p}|b_W).$$
- $\mathsf{TrapGen}(msk, Q)$: Given a master key msk and a search query $Q = \{R, W^*\}$, where R is a geometric range and W^* is a queried keywords set. SU first generates the bitmap of W^* as b_{W^*} and encodes the R into vector tokens as described above. Then, for each token tk_i, SU generates the search trapdoor as
$$\mathsf{TK}_i \leftarrow \mathsf{SHVE.KeyGen}(msk, \mathsf{tk}_i|b_{W^*}).$$
Finally, SU sends the set of search trapdoors $T_Q = \{\mathsf{TK}_1, ..., \mathsf{TK}_k\}$ to CSP, where k depends on the scope of the query range.
- $\mathsf{Query}(\mathsf{EDB}, T_Q)$: Upon receiving the trapdoors T_Q, for each spatial object $\mathsf{ED} \in \mathsf{EDB}$ and search token $\mathsf{TK}_i \in T_Q$, the CSP performs
$$\mathsf{Flag}_i \leftarrow \mathsf{SHVE.Query}(\mathsf{ED}, \mathsf{TK}_i).$$
If $\exists \mathsf{Flag}_i = \mathsf{ture}$, adds ED into results set $\mathcal{R}$. Finally, the CSP returns $\mathcal{R}$ to SU.

B. PBRQ-Q: BQ-tree based PBRQ Scheme

As described above, the PBRQ-L is a linear search scheme regarding the number of data objects in a database, which is difficult to cope with the requirement of large-scale databases. Here, we design a tree structure called Bitmap Quadtree (BQ-tree) and a corresponding pruning algorithm to reduce the search complexity to *faster-than-linear*. A BQ-tree is a hybrid index structure based on bitmap and Quadtree. In particular, each node of BQ-tree contains both spatial and keyword information. Each leaf node (i.e., spatial object) of BQ-tree contains a vector of Gray code for spatial point and bitmap for the keywords as described in PBRQ-L. Each non-leaf node of BQ-tree contains four bound points that denote its bounding area, and each bound point also contains a vector of Gray code and bitmap. It is worth noting that the bitmap of the bound point is the superimposition (OR-ing) of all the bitmaps of its children nodes. Thus a bitmap of a node is equivalent for all the objects in its subtree. Given a Boolean geometric range, the search process of BQ-ree is described as follows.

[1] In our scheme, we divide the d-dimensional plaintext space into T^d grid to cover all integer spatial points.

- $\mathsf{Setup}(1^\lambda) \rightarrow msk$: Given a security parameter λ, DO computes and outputs a master key

$$msk \leftarrow \mathsf{SHVE.Setup}(1^\lambda).$$

- $\mathsf{IndexBuild}(msk, \mathsf{DB})$: Given a master key msk and a spatial database DB, DO first generates a BQ-tree $\mathcal{T}$ according to DB. Let $N = \{\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4, b_N, B_N\}$ be a non-leaf node of $\mathcal{T}$, where $\mathsf{p}_j (1 \leq j \leq 4)$ is a bound point of bounding area B_N, b_N is the bitmap of the current node. Let tk be the tokens of bounding area B_N[a], the non-leaf node N is encrypted as

$$\begin{aligned} &\mathsf{ENp}_j \leftarrow \mathsf{SHVE.Enc}(msk, Gray(\mathsf{p}_j)),\\ &\mathsf{ENb}_N \leftarrow \mathsf{SHVE.Enc}(msk, b_N),\\ &\mathsf{ENtk} \leftarrow \mathsf{SHVE.KeyGen}(msk, \mathsf{tk}). \end{aligned}$$

For each leaf node $D_i = \{\mathsf{p}, W\} \in \mathsf{DB}$ of $\mathcal{T}$, DO encodes the p as $g_\mathsf{p} \leftarrow Gray(\mathsf{p})$ and generates the bitmap of W as b_W. Then, DO encrypts the node as

$$\begin{aligned} &\mathsf{EDp} \leftarrow \mathsf{SHVE.Enc}(msk, g_\mathsf{p}),\\ &\mathsf{EDb} \leftarrow \mathsf{SHVE.Enc}(msk, b_W). \end{aligned}$$

- $\mathsf{TrapGen}(msk, Q)$: Given a master key msk and a search query $Q = \{R, W^*\}$, where R is a geometric range and W^* is a query keywords set. SU first generates the bitmap of W^* as b_{W^*} and encodes the R into tokens $\{\mathsf{tk}_i\}_{i=1}^{k}$. Let c be the center point of R, SU generates the search trapdoors as

$$\begin{aligned} &\mathsf{TK}_i \leftarrow \mathsf{SHVE.KeyGen}(msk, \mathsf{tk}_i),\\ &\mathsf{Eb}_{W^*} \leftarrow \mathsf{SHVE.KeyGen}(msk, b_{W^*}),\\ &\mathsf{Ec} \leftarrow \mathsf{SHVE.Enc}(msk, \mathsf{c}). \end{aligned}$$

Finally, SU sends the set of search trapdoors $T_Q = \{\mathsf{TK}_1, ..., \mathsf{TK}_k, \mathsf{Eb}_{W^*}, \mathsf{Ec}\}$ to CSP.

- $\mathsf{Query}(\mathsf{EDB}, T_Q)$: Upon receiving T_Q, starting from the root node of $\mathcal{T}$, CSP performs the search process as:
  1) For each child node of the current node, performs

  $$\mathsf{Flag}_b \leftarrow \mathsf{SHVE.Query}(\mathsf{ENb}_N, \mathsf{Eb}_{W^*}).$$

  If $\mathsf{Flag}_b = \mathsf{true}$, then performs

  $$\begin{aligned} &\mathsf{Flag}_\mathsf{c} \leftarrow \mathsf{SHVE.Query}(\mathsf{Ec}, \mathsf{ENtk}),\\ &\mathsf{Flag}_{\mathsf{p}_{ij}} \leftarrow \mathsf{SHVE.Query}(\mathsf{ENp}_j, \mathsf{TK}_i), \end{aligned}$$

  If $\exists \mathsf{Flag}_\mathsf{c} = \mathsf{true}$ or $\exists \mathsf{Flag}_{\mathsf{p}_{ij}} = \mathsf{ture}$, moving to search its children nodes.
  2) When traverse to a leaf node (i.e., spatial object ED), CSP performs

  $$\begin{aligned} &\mathsf{Flag}_b \leftarrow \mathsf{SHVE.Query}(\mathsf{Eb}_{W^*}, \mathsf{EDb}),\\ &\mathsf{Flag}_{\mathsf{p}_i} \leftarrow \mathsf{SHVE.Query}(\mathsf{EDp}, \mathsf{TK}_i), \end{aligned}$$

  If $\mathsf{Flag}_b = \mathsf{ture}$ and anyone of $\mathsf{Flag}_{\mathsf{p}_i}$ is true, add ED into results set $\mathcal{R}$.

Finally, CSP returns results set $\mathcal{R}$ to SU.

[a] Due to the construction of Quadtree, the bounding area of each non-leaf node can be encoded into a single token.

Fig. 5. Details of PBRQ-Q

1) Starting from the root node, for each child node of the current node, if the geometric range intersects with a non-leaf node and the bitmap match, moving to search its children nodes.
2) When traversing to a leaf node (i.e., spatial object), if the point of this leaf node insides the geometric range and the bitmap match, add this object into the results set.

In order to achieve the above search algorithm over encrypted databases, the key part is to check whether a query intersects with a rectangle (i.e., bounding area of a non-leaf node) at a non-leaf node and also verify whether a point is inside a query at a leaf node in the ciphertext domain. It is obvious to see that we can still leverage the same method in our proposed PBRQ-L to check whether a point is inside a query over encrypted data and whether the bitmaps are matched. As for check whether a query intersects with a rectangle at a non-leaf node, we can also use a similar method. As shown in Fig. 4(b), if a bound point of the non-leaf node insides a query or the center point of the query insides the non-leaf node, the query intersects with the non-leaf node.

Therefore, each non-leaf node can be generated as bound points and a bounding area. The Gray code of bound points and the bounding area are encrypted by $\mathsf{SHVE.KeyGen}$, $\mathsf{SHVE.Enc}$, respectively. The query range is generated as a center point and queried area, where the center point is encrypted by $\mathsf{SHVE.Enc}$ and the query range is encrypted by $\mathsf{SHVE.KeyGen}$. In this way, CSP is able to check whether a query range intersects with a rectangle at a non-leaf node. The detail of PBRQ-Q is shown in Fig. 5

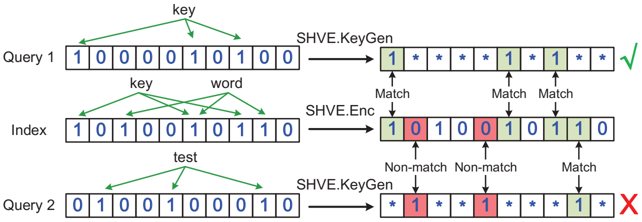


Fig. 6. An example of Bloom filter

Performance Improvement. In the above schemes, bitmaps are stored in each non-leaf node and spatial object for fast pruning and keywords match. However, it is a challenge in large-scale data applications that the length of the bitmap increases as the number of keywords contained in the data set increases, which incurs increasing storage, search, and communication cost. To address this issue, we further improve the proposed schemes using the Bloom Filter (BF). A BF is an efficient probabilistic (indexing) data structure to store information about the existence of an item in a dataset [27]. It is constructed from a set of t independent hash functions $h_i(\cdot)$. Each hash function produces a uniformly distributed bit location in the range $[0, m-1]$ for a given message, where m is the bit-length of hash function's output. To map an keyword set W to a Bloom filter $\mathcal{B}$, calculate $H_1 = h_1(W), H_2 = h_2(W), ..., H_t = h_t(W)$ and set $\mathcal{B}[H_i] = 1$, where $i \in [1, t]$. As shown in Fig. 6, to check whether "key" or "test" is in the dataset, a search user generates BF using $h_1, ..., h_t$. Each "1" bit of "key" matches the corresponding bit of dataset's BF, the "key" probably exists; two "1" bits of "test" does not match the corresponding bit of dataset's BF, it does not exist. The probability for false positives of a BF is computed as

$$f = (1 - (1 - \frac{1}{m})^{|W|t})^t \approx (1 - e^{-|W|t/m})^t,$$

where $|W|$ is the number of keywords in BF, m is the length of BF, and t is the number of hash functions. We can use Bloom filter instead of the bitmap in each non-leaf node and spatial object of BQ-tree. Thus, each spatial object stores a Gray code vector of spatial position and a Bloom filter $\mathcal{B}$ of textual keywords contained in the current object. As for each non-leaf node of the index tree, it stores the token vectors of its bounding area and a Bloom filter $\mathcal{B}$ of all textual keywords contained in its children, which can be calculated by the "OR" operation of all its children's Bloom filter.

V. Security Analysis

A. Security Definitions

Before presenting the formal security definition of PBRQ, we first define four leakage functions:

- *Size Pattern:* CSP knows both the total number of indexes stored on itself and the total number of range queries (i.e., trapdoor) has been submitted by SU.
- *Access Pattern:* CSP reveals the identifier of each encrypted data returned for specific range query (i.e., trapdoor).
- *Search Pattern:* CSP can learn whether an encrypted spatial data is queried by two different trapdoors.
- *Path Pattern:* CSP can learn how the search algorithm traverse in the tree structure[2].

These leaked information are default in most searchable encryption schemes [1], [3]–[5], [28]. Given a database DB and a query Q, we denote the above leakages as $\mathcal{L}(\mathsf{DB}, Q)$.

Definition 2 (IND-SCPA Data Privacy of PBRQ). *Let Π = (Setup, IndexBuining, TrapGen, Query) be a PBRQ scheme over security parameter λ. The security game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ is defined as:*

- ***Init**: The adversary $\mathcal{A}$ submits two databases $\mathsf{DB}_0 = \{D_{0,1}, ..., D_{0,n}\}$ and $\mathsf{DB}_1 = \{D_{1,1}, ..., D_{1,n}\}$ with the same number of data objects to the challenger $\mathcal{C}$, where $D_{0,i}, D_{1,i} \in \Delta_T^d (1 \leq i \leq n)$.*
- ***Setup**: The challenger $\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda)$ to generate a master secret key msk and it keeps msk private.*
- ***Phase 1**: The adversary $\mathcal{A}$ adaptively submits a number of requests, where each requests is one of the two following types:*
  ***Ciphertext**: On the j-th ciphertext request, $\mathcal{A}$ outputs a dataset DB'_j, where $\mathsf{DB}'_j = \{D'_{j,1}, ..., D'_{i,n}\}, D'_{j,i} \in \Delta_T^d (1 \leq i \leq n)$. Challenger $\mathcal{C}$ responses with an encrypted database $\mathsf{EDB}_j \leftarrow \mathsf{IndexBuild}(msk, \mathsf{DB}_j)$.*
  ***Trapdoor**: On the j-th trapdoor request, $\mathcal{A}$ outputs a Boolean geometric range query $Q_j = \{R_j, W_j^*\}$, where $R \in \Delta_T^d, W_j^* \in W$. Challenger $\mathcal{C}$ responses with a search trapdoor $T_{Q_j} \leftarrow \mathsf{TrapGen}(msk, Q_j)$, where Q_j is subject to $\mathcal{L}(\mathsf{EDB}_0, Q_j) = \mathcal{L}(\mathsf{EDB}_1, Q_j)$.*
- ***Challenge**: With $\mathsf{DB}_0, \mathsf{DB}_1$ selected in **Init**, challenger $\mathcal{C}$ flips a coin $b \in \{0, 1\}$, computes $\mathsf{EDB}_b \leftarrow \mathsf{IndexBuild}(msk, \mathsf{DB}_b)$ to adversary $\mathcal{A}$.*
- ***Phase 2**: The adversary $\mathcal{A}$ continues to adaptively submit a number of requests, which are still subjected to the same restrictions in **Phase 1**.*
- ***Guess**: The adversary takes a guess b' of b.*

We say that Π is secure against Selective Chosen-Plaintext Attacks on data privacy if for any polynomial-time adversary in the above game, it has at most a negligible advantage

$$\mathbf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{IND\text{-}SCPA\text{-}Data}}(1^\lambda) = |\Pr[b' == b] - \frac{1}{2}| \leq \mathsf{negl}(\lambda).$$

Definition 3 (IND-SCPA Query Privacy of PBRQ). *Let Π = (Setup, IndexBuild, TrapGen, Query) be a PBRQ scheme over security parameter λ. The security game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ is defined as:*

- ***Init**: The adversary $\mathcal{A}$ submits two distinct Boolean geometric range queries $Q_0 = \{R_0, W_0^*\}, Q_1 = \{R_1, W_1^*\}$, where $R_0, R_1 \in \Delta_T^d, W_0^*, W_1^* \in W$.*
- ***Setup**: The challenger $\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda)$ to generate a master secret key msk and it keeps msk private.*
- ***Phase 1**: The adversary $\mathcal{A}$ adaptively submits a number of requests, where each requests is one of the two following types:*
  ***Ciphertext**: On the j-th ciphertext request, the adversary $\mathcal{A}$ outputs a dataset DB'_j, where $\mathsf{DB}'_j = \{D'_{j,1}, ..., D'_{i,n}\}$, $D'_{j,i} \in \Delta_T^d (1 \leq i \leq n)$. Challenger $\mathcal{C}$ responses with an encrypted database $\mathsf{EDB}_j \leftarrow \mathsf{IndexBuild}(msk, \mathsf{DB}_j)$, where EDB_j is subject to $\mathcal{L}(\mathsf{EDB}_j, Q_0) == \mathcal{L}(\mathsf{EDB}_j, Q_1)$.*
  ***Trapdoor**: On the j-th trapdoor request, $\mathcal{A}$ outputs a Boolean geometric range query $Q_j = \{R_j, W_j^*\}$, where $R \in \Delta_T^d, W_j^* \in W$. Challenger $\mathcal{C}$ responses with a search trapdoor $T_{Q_j} \leftarrow \mathsf{TrapGen}(msk, Q_j)$.*
- ***Challenge**: With Q_0, Q_1 selected in **Init**, challenger $\mathcal{C}$ flips a coin $b \in \{0, 1\}$, computes $T_{Q_b} \leftarrow \mathsf{TrapGen}(msk, Q_b)$ to adversary $\mathcal{A}$.*
- ***Phase 2**: The adversary $\mathcal{A}$ continues to adaptively submit a number of requests, which are still subjected to the same restrictions in **Phase 1**.*
- ***Guess**: The adversary takes a guess b' of b.*

We say that Π is secure against Selective Chosen-Plaintext Attacks on query privacy if for any polynomial-time adversary

[2] This leakage function only works on the PBRQ-Q scheme because of PBRQ-L does not use tree structure.

TABLE II
COMPARISON OF COMPUTATIONAL COMPLEXITY.

Scheme	IndexBuild	TrapGen	Query
GRSE	$O(n)(6M+2)\mathsf{T_e}$	$8M\mathsf{T_e}$	$O(n)(2M+2)\mathsf{T_p}$
GRSE-tree	$O(n+n')(6M+2)\mathsf{T_e}$	$8M\mathsf{T_e}$	$O(\log n)(2M+2)\mathsf{T_p}$
FastGeo	$O(\tau)((6T+2)\mathsf{T_e}+\mathsf{T_{PRF}})$	$O(\tau')(8T\mathsf{T_e}+\mathsf{T_{PRF}})$	$O(\tau')(2T+2)\mathsf{T_p}$
PBRQ-L	$O(n)(2^{\lceil \log_2 T \rceil}+m)\mathsf{T_{PRF}}$	$O(k)((2^{\lceil \log_2 T \rceil}+m)\mathsf{T_{PRF}}+\mathsf{T_{Enc}})$	$O(kn)((2^{\lceil \log_2 T \rceil}+m)\mathsf{T_{XOR}}+\mathsf{T_{Dec}})$
PBRQ-Q	$O(n)(2^{\lceil \log_2 T \rceil}+m))\mathsf{T_{PRF}}+$ $O(n')(2^{\lceil \log_2 T \rceil}(5\mathsf{T_{PRF}}+\mathsf{T_{XOR}})+\mathsf{T_{Enc}}+m\mathsf{T_{PRF}})$	$(O(k)2^{\lceil \log_2 T \rceil}+m)(\mathsf{T_{PRF}}+\mathsf{T_{XOR}})+$ $(O(k)+1)\mathsf{T_{Enc}}+m\mathsf{T_{PRF}}$	$O(k\log n)((2^{\lceil \log_2 T \rceil}+m)\mathsf{T_{XOR}}+$ $\mathsf{T_{Dec}})$

in the above game, it has at most a negligible advantage

$$\mathbf{Adv}^{\mathsf{IND\text{-}SCPA\text{-}Query}}_{\Pi,\mathcal{A}}(1^\lambda) = |\Pr[b' == b] - \frac{1}{2}| \leq \mathsf{negl}(\lambda).$$

B. Security of Proposed Schemes

Theorem 1 (Data privacy of PBRQ)**.** *Our PBRQ schemes are IND-SCPA data secure if the SHVE is IND-SCPA secure.*

Proof. To prove the IND-SCPA data privacy of our PBRQ, we simulate the security game defined in Def. 2 with an adversary $\mathcal{A}'$ from the ideal security game of the SHVE. And then we demonstrate that compromising the IND-SCPA data privacy of our PBRQ is equivalent to compromise the IND-SCPA of SHVE, which contradicts the assumption that SHVE is IND-SCPA secure proved in [21]. Specifically, by following Def. 2, the security game of our PBRQ is essentially simulated by q instances of SHVE, where q is the total number of SHVE instances needed in the game. As a result, the adversary $\mathcal{A}'$ could distinguish the two databases DB_0 and DB_1 as long as it could distinguish any pair of two vectors in the security game of SHVE. Then, we have

$$\mathbf{Adv}^{\mathsf{IND\text{-}SCPA\text{-}Query}}_{\Pi,\mathcal{A}}(1^\lambda) \leq \mathbf{Adv}^{\mathsf{IND\text{-}SCPA}}_{\mathsf{SHVE},\mathcal{A}'}(1^\lambda) \leq q \cdot \mathsf{negl}(\lambda)$$

which proves the IND-SCPA data privacy of our PBRQ. □

Theorem 2 (Query privacy of PBRQ)**.** *Our PBRQ schemes are IND-SCPA query secure if the SHVE is IND-SCPA secure.*

Proof. The query privacy of our PBRQ can be proved in a similar way as the proof of the data privacy analyzed above. We skip further details due to space limitations. □

VI. PERFORMANCE ANALYSIS

In this section, we analyze the performance of the proposed schemes theoretically and experimentally.

A. Theoretical Analysis

The notations used in theoretical analysis are described in TABLE III. Let ω be the size of a vector, then the computational cost of $\mathsf{SHVE.Enc}$, $\mathsf{SHVE.KeyGen}$, and $\mathsf{SHVE.Query}$ are $\omega\mathsf{T_{PRF}}$, $\omega(\mathsf{T_{PRF}}+\mathsf{T_{XOR}})+\mathsf{T_{Enc}}$, and $\omega\mathsf{T_{XOR}}+\mathsf{T_{Dec}}$, respectively. In PBRQ-L, each data object is encrypted by $\mathsf{SHVE.Enc}$ as a vector $g_{\mathsf{p}}|b_W$ in $\mathsf{IndexBuild}$, which costs $(2^{\lceil \log_2 T \rceil}+m)\mathsf{T_{PRF}}$. Each query is generated into k tokens and encrypted by $\mathsf{SHVE.KeyGen}$ in $\mathsf{TrapGen}$, which costs $k((2^{\lceil \log_2 T \rceil}+m)\mathsf{T_{PRF}}+\mathsf{T_{Enc}})$. In Query, for each data object, the computational cost is $k((2^{\lceil \log_2 T \rceil}+m)\mathsf{T_{XOR}}+\mathsf{T_{Dec}})$. As for PBRQ-Q, each non-leaf node of BQ-tree is generated as four $2^{\lceil \log_2 T \rceil}$-bits bounding points (encrypted by $\mathsf{SHVE.Enc}$), a $2^{\lceil \log_2 T \rceil}$-bits bounding area token (encrypted by $\mathsf{SHVE.Enc}$), and a m-bits BF (encrypted by $\mathsf{SHVE.KeyGen}$); each leaf node of BQ-tree is generated as a $2^{\lceil \log_2 T \rceil}$-bits spatial point and a m-bits BF. Each search query in PBRQ-Q is generated as k $2^{\lceil \log_2 T \rceil}$-bits range tokens (encrypted by $\mathsf{SHVE.KeyGen}$), a $2^{\lceil \log_2 T \rceil}$-bits center point (encrypted by $\mathsf{SHVE.Enc}$), and a m-bits BF (encrypted by $\mathsf{SHVE.KeyGen}$). In Query, for each data object, the computational cost is $k((2^{\lceil \log_2 T \rceil}+m)\mathsf{T_{XOR}}+\mathsf{T_{Dec}})$.

TABLE III
NOTATIONS FOR COMPARISON ANALYSIS

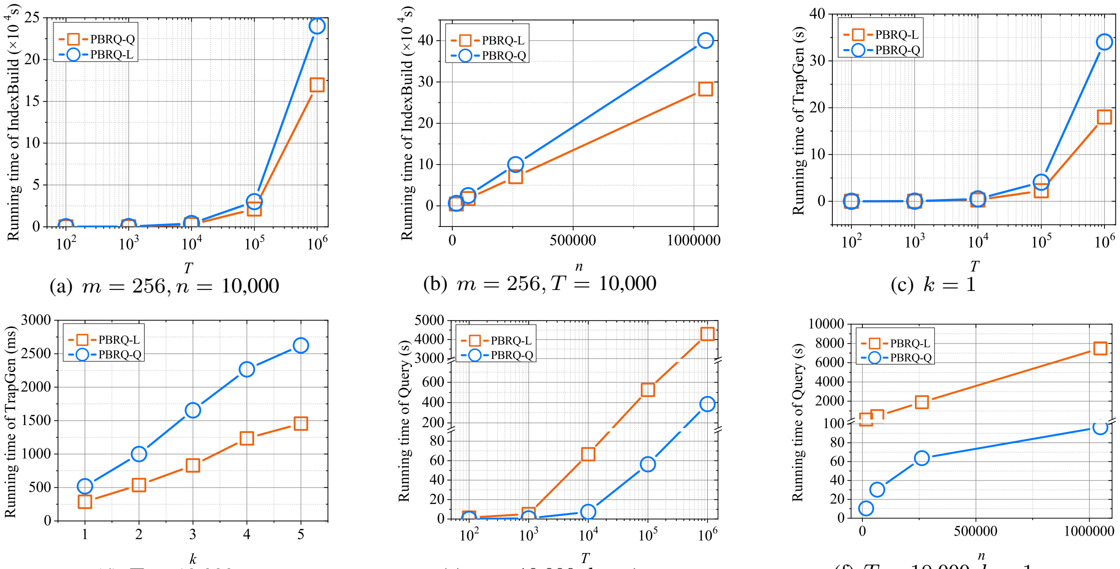
Notations	Meaning
n	size of dabab
n'	number of non-leaf nodes in index tree
m	size of bitmap in our schemes
M	size of BF in GRSE [3]
τ	total number of spatial vector instances in the database
τ'	number of index instances the server needs to evaluate
k	number of range tokens
$\mathsf{T_p}$	time taken to compute a pairing
$\mathsf{T_e}$	time taken to compute a exponentiation
$\mathsf{T_{PRF}}$	time taken to compute a PRF
$\mathsf{T_{XOR}}$	time taken to perform an exclusive-or operation over λ
$\mathsf{T_{Enc}}$	time taken to perform a symmetric encryption
$\mathsf{T_{Dec}}$	time taken to perform a symmetric decryption

In TABLE II, we compare the computational cost of our schemes with existing works. Note that the GRSE [3], GRSE-tree [3], and FastGeo [4] are presented based on pairing-based encryption SSW [13], while our PBRQ schemes are based on symmetric key encryption SHVE. Therefore, the computational cost of the above three schemes is significantly higher than that of our two PBRQ schemes because our schemes avoid expensive pairing and exponentiation operations. We will evaluate the performance of these schemes using the real-world datasets in Section VI-B.

B. Experimental Evaluations

Setup and Implementation. We implement our schemes in JAVA, where the code of SHVE is from [21]. Specifically, the security parameter is $\lambda = 128$. For comparison, we also implement the GRSE [3], GRSE-tree [3] and FastGeo [4] using the JAVA Pairing-Based Cryptography (JPBC) library [3], where the pairing operations are evaluated on super-singular curve $y^2 = x^3 + x$. In our PBRQ-Q, the maximum capacity of

[3] http://gas.dia.unisa.it/projects/jpbc/



(a) $m = 256, n =$ 10,000

(b) $m = 256, T =$ 10,000

(c) $k = 1$

(d) $T =$ 10,000

(e) $n =$ 10,000, $k = 1$

(f) $T =$ 10,000, $k = 1$

Fig. 7. Actual experimental results of proposed schemes. (a), (b) The computational cost of **IndexBuild**; (c), (d) The computational cost of **TrapGen**; (e), (f) The computational cost of **Query**.

deepest non-leaf nodes in BQ-tree is set to 16. In GRSE-tree, the maximum capacity of each node in R-tree is also set as 16. We leverage the Gowalla dataset[4], which contains 6,442,890 location check-in data objects collected from 196,591 users, to demonstrate the performance of our schemes. We preprocess the original dataset before we apply it to our experiments. Since the objects in Gowalla dataset do not contain textual keywords, we randomly select a document from the Enron dataset[5] to extract 16 textual keywords for each object. We set the size of BF in our PBRQ schemes as $m = 256, t = 16$, such that the false positive is $f \leq (1 - e^{(-16\times16)/256})^{16} \approx 4.2\times10^{-7}$. We test our schemes over this dataset with different data sizes. The experiments are conducted on a machine running Ubuntu 14.04 with 16 Intel Xeon v2 CPU, 30GB RAM.

Experimental Results. In our evaluation, each experiment result is the average running time of 10 tests. In Fig. 7(a), we plot the running time of IndexBuild in different size of plaintext space T, where dataset size n = 10,000. We can see that the running time of both schemes increases with the size of plaintext space. This is because a larger T leads to a longer Gray code of spatial point, which increases the computational cost of SHVE.Enc and SHVE.KeyGen. In addition, the running time of PBRQ-Q is longer than that of PBRQ-L, which is caused by additional tree nodes. Fig. 7(b) shows that the running time of IndexBuild in both schemes linearly increases with the dataset size, where T =10,000. It is worth noting that the running time only reflects the computational cost of spatial data and index tree encryption to focus on the core performance of the schemes, which ignores the spatial point encoding and index tree building process. Fig. 7(c) plots the running time of TrapGen in different size of T, where the number of range tokens $k = 1$. The same as IndexBuild, the running time of both schemes increases with the size of plaintext space because of the computational cost of SHVE.Enc and SHVE.KeyGen increases with the bit-length of the Gray code. We can also see from Fig. 7(d) that the running time of TrapGen is linear increases with k, since every range tokens need to be encrypted. The running time of Query in different situations is shown in Fig. 7(e) and Fig. 7(e). The search query is a 64×64 square associated with three keywords. We can observe from Fig. 7(e) that the running time of Query increases rapidly with T where n = 10,000, k = 1, because the computational cost of SHVE.Query is also increasing with the bit-length of the Gray code. Fig. 7(f) shows that the search complexity of PBRQ-Q is *fast-than-linear* and the running time of Query in PBRQ-Q over million data objects is 92.7s, which is about 86× faster than that of PBRQ-L.

TABLE IV
QUERY TIME COMPARISON AMONG SCHEMES

n	2×10^4	4×10^4	6×10^4	8×10^4	1×10^5
GRSE	4.36 h	9.12 h	13.67 h	17.32 h	20.96 h
GRSE-tree	0.73 h	1.45 h	2.32 h	2.84 h	3.13 h
FastGeo	76.33 s	132.62 s	218.16 s	286.28 s	320.21 s
PBRQ-L	10.64 s	22.51 s	32.33 s	43.21 s	52.93 s
PBRQ-Q	0.92 s	1.64 s	2.07 s	2.92 s	3.14 s

Notes. s: seconds;
h: hours;

Finally, we compare our PBRQ schemes with two previous schemes over our test dataset, where the plaintext space $T =$ 1,000 and the dataset size n from 2×10^4 to 1×10^5. The size of BF in two GRSE schemes is set as $M =$ 1,000. The range

[4] http://snap.stanford.edu/data/loc-Gowalla.html
[5] http://www.cs.cmu.edu/~enron/

query is a 64×64 square and the keyword set contains three keywords. As we can observe from TABLE IV, our schemes are extremely efficient. Even the query time of PBRQ-L is faster than the most efficient previous scheme FastGeo, and PBRQ-Q is at least $70\times$ faster compared to FastGeo.

VII. Conclusion

In this paper, we proposed two privacy-preserving Boolean range queries schemes over encrypted spatial databases for the first time. Firstly, we designed a basic PBRQ scheme using SHVE, bitmap, and Gray code. Then, we presented a novel index structure called Bitmap Quadtree (BQ-tree) to reduce the search complexity of the basic PBRQ scheme. In addition, we rigorously analyzed the security of our schemes under IND-SCPA. Finally, we implemented and evaluated our schemes using real-world datasets, and showed that our schemes are more efficient than existing schemes.

Acknowledgments

This work was supported by the The Key Program of NSFC Grant (U1405255), Shaanxi Science & Technology Coordination & Innovation Project (2016TZC-G-6-3), the Fundamental Research Funds for the Central Universities (SA-ZD161504, No.JB191506), the National Natural Science Foundation of China (No.61702404, No.U1804263, No.61702105), and the National Natural Science Foundation of Shaanxi Province (No.2019JQ-005). Xiangyu Wang was supported by the Doctoral Students' Short Term Study Abroad Scholarship, Xidian University.

References

[1] B. Wang, M. Li, H. Wang, and H. Li, "Circular range search on encrypted spatial data," in *Proc. CNS'15*, 2015, pp. 794–795.

[2] H. Zhu, R. Lu, C. Huang, L. Chen, and H. Li, "An efficient privacy-preserving location-based services query scheme in outsourced cloud," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 9, pp. 7729–7739, 2016.

[3] B. Wang, M. Li, and H. Wang, "Geometric range search on encrypted spatial data," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 4, pp. 704–719, 2016.

[4] B. Wang, M. Li, and L. Xiong, "Fastgeo: Efficient geometric range queries on encrypted spatial data," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 2, pp. 245–258, 2019.

[5] G. Xu, H. Li, Y. Dai, K. Yang, and X. Lin, "Enabling efficient and geometric range query with access control over encrypted spatial data," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 870–885, 2019.

[6] D. X. Song, D. A. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE S&P*, 2000, pp. 44–55.

[7] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. CCS'06*, 2006, pp. 79–88.

[8] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Proc. CRYPTO'13*, 2013, pp. 353–373.

[9] S. Kamara and T. Moataz, "Boolean searchable symmetric encryption with worst-case sub-linear complexity," in *Proc. EUROCRYPT'17*, 2017, pp. 94–124.

[10] Y. Miao, X. Liu, R. H. Deng, H. Wu, H. Li, J. Li, and D. Wu, "Hybrid keyword-field search with efficient key management for industrial internet of things," *IEEE Trans. Industrial Informatics*, vol. 15, no. 6, pp. 3206–3217, 2019.

[11] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput*, 2009, pp. 169–178.

[12] N. K. Boneh D, Goh E J, "Revisiting security risks of asymmetric scalar product preserving encryption and its variants," in *Proc. Theory of Cryptography Conference*, 2005, pp. 223–238.

[13] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Proc. TCC'09*, 2009, pp. 457–473.

[14] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proc. SIGMOD'09*, 2009, pp. 139–152.

[15] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma, "Hybrid index structures for location-based web search," in *Proc. CIKM '05*, 2005, pp. 155–162.

[16] I. De Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proc. ICDE'08*, 2008, pp. 656–665.

[17] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 337–348, 2009.

[18] Z. Li, K. C. K. Lee, B. Zheng, W. Lee, D. Lee, and X. Wang, "Ir-tree: An efficient index for geographic document search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 4, pp. 585–599, 2011.

[19] J. a. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *Proc. SSTD'11*, 2011, pp. 205–222.

[20] C. Zhang, Y. Zhang, W. Zhang, and X. Lin, "Inverted linear quadtree: Efficient top k spatial keyword search," in *Proc. ICDE'13*, 2013, pp. 901–912.

[21] S. Lai, S. Patranabis, A. Sakzad, J. K. Liu, D. Mukhopadhyay, R. Steinfeld, S. Sun, D. Liu, and C. Zuo, "Result pattern hiding searchable encryption for conjunctive queries," in *Proc. of ACM CCS'18*, 2018, pp. 745–762.

[22] J. R. Bitner, G. Ehrlich, and E. M. Reingold, "Efficient generation of the binary reflected gray code and its applications," *Commun. ACM*, vol. 19, no. 9, pp. 517–521, 1976.

[23] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, no. 1, pp. 1–9, 1974.

[24] X. Wang, J. Ma, Y. Miao, R. Yang, and Y. Chang, "EPSMD: an efficient privacy-preserving sensor data monitoring and online diagnosis system," in *Proc. INFOCOM'18*, 2018, pp. 819–827.

[25] X. Wang, J. Ma, X. Liu, and Y. Miao, "Search in my way: Practical outsourced image retrieval framework supporting unshared key," in *Proc. INFOCOM'19*, 2019, pp. 2485–2493.

[26] X. Wang, J. Ma, Y. Miao, X. Liu, and R. Yang, "Privacy-preserving diverse keyword search and online pre-diagnosis in cloud computing," *IEEE Transactions on Services Computing*, 2019.

[27] B. H. BLOOM, "Space/time trade-offs in hash coding with allowable errors," *Communications of ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[28] J. Ning, J. Xu, K. Liang, F. Zhang, and E. Chang, "Passive attacks against searchable encryption," *IEEE Trans. Information Forensics and Security*, vol. 14, no. 3, pp. 789–802, 2019.