Pystin: Enabling Secure LBS in Smart Cities With Privacy-Preserving Top-*k* Spatial–Textual Query

Divya Negi, Suprio Ray, *Member, IEEE*, and Rongxing Lu, *Senior Member, IEEE*

***Abstract*—The convergence of technologies like Cloud computing, mobile, and smart phone technologies has led to the rapid development of location-based services (LBS) in smart cities. For flexibility and cost savings, there is a recent trend to migrate LBS to the Cloud, however it poses a serious threat to the user privacy. In this paper, we present a new privacy preserving top-*k* spatio-textual keyword (T*k*SK) query scheme, called privacy-preserving spatio-textual index (Pystin), which is performed over outsourced Cloud and can enable secure LBS in smart cities. In Pystin, a query user's accurate location is protected by the combination of Boneh–Goh–Nissim homomorphic encryption and hash bucket techniques, and the privacy of textual information are persevered by a one-way hash function. In addition, a quad-tree-based spatio-textual indexing is integrated into Pystin to further reduce the query latency. Detailed security analyzes show that the proposed Pystin scheme is indeed a privacy-preserving T*k*SK query scheme. Furthermore, extensive experiments are conducted, and results confirm the scalability, efficiency properties of our proposed Pystin scheme.**

***Index Terms*—Location-based services (LBS), privacy-preserving, smart city, top-*k* spatial–textual query.**

I. Introduction

As THE popularity and affordability of smart-phones, and the advances in mobile technologies and user friendly applications have led to the rapid growth in location-based services (LBS) in smart cities. Applications, such as Apple Maps, Google Maps, social networks, location-based gaming, targeted advertising, and points of interest search, produce huge volumes of data each day. For example, as of 2017, there are 1.32 billion daily active users (DAUs) on Facebook that post 4.3 billion messages everyday on average. Similarly, 328 million DAUs on Twitter send 500 million tweets a day [1], [2]. The rapidly rising data volume has the potential to help generate deep insights through analytics to drive critical decision making for businesses and other organizations.

Big data holds great promise for the betterment of our smart cities. However, big data also exposes users to great privacy and security risks. Due to its "pay for what you use" model, the Cloud is very attractive to enterprises financially, and more and more companies are outsourcing their data onto the Cloud. This, however, is fraught with serious privacy concerns. Any data that is stored as plaintext can be misused by either a malicious third party, or by a Cloud service provider which is considered as *honest-but-curious*. Here, *honest-but-curious* means that a Cloud server stores and processes data honestly, but would like to know more about the data. In addition, the privacy of users who submit queries to a Cloud hosted system could be compromised by their search keywords. Even de-identification technique, through anonymizing data for the purpose of query processing, may still not be sufficient to retain the privacy of users. For instance, the anonymized search logs released by AOL for academic purposes were used to easily identify users by their searches [3]. Therefore, the impetus to developing novel privacy-preserving query processing techniques becomes stronger than ever before.

A top-*k* spatio-textual keyword (T*k*SK) query is one important type of LBS queries, which retrieves a set of *k* objects ranked by a ranking function according to their spatial and textual relevance [4]. Consider for example, on a Friday night you are looking for a "specialty restaurant serving Indian style spicy curry close to your current location in a city that you first visit." In this case, the ranking function takes into account both the textual relevance and the spatial proximity of the spatio-textual objects from the query point. However, when users submit this type of query, some sensitive information like location and query patterns of users could be leaked, if not well protected, these sensitive information could be further misused by criminals to analyze users' behavior and attack them. Further, as more data owners tend to outsource their data to the Cloud, it becomes crucial to consider privacy-preserving T*k*SK (PT*k*SK) search in the context of outsourced Cloud, which can help protect against possible privacy breaches and unsolicited access.

Preserving privacy in Cloud comes at a price, and adding a privacy feature will lead to the search latency considerably. Given the importance of privacy-preserving LBS (PP-LBS), a number of schemes have been proposed in past years. However, most of them are concerned with designing privacy-preserving schemes for either spatial or keyword queries, but not both. For example, Yiu *et al.* [5] proposed the use of symmetric encryption AES and complete transformation of location space to secure the components of a T*k*SK query.

Manuscript received September 11, 2018; revised January 3, 2019; accepted February 18, 2019. Date of publication March 1, 2019; date of current version October 8, 2019. The work of S. Ray and R. Lu was supported in part by the Natural Sciences and Engineering Research Council of Canada Discovery Grants and in part by New Brunswick Innovation Foundation (NBIF) Start-Up Grants. *(Corresponding author: Suprio Ray.)*

D. Negri was with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada. She is now with Morgan Stanley Canada Limited, Montreal, QC H3C 3S4, Canada (e-mail: dnegi@unb.ca).

S. Ray and R. Lu are with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada (e-mail: sray@unb.ca; rlu1@unb.ca).

Digital Object Identifier 10.1109/JIOT.2019.2902483

Hu *et al.* [6] used R-tree-based index in conjunction with a homomorphic encryption scheme called asymmetric scalar product-preserving encryption with noise (ASPEN). However, these approaches fail to address the challenges in secure spatio-textual query processing. Therefore, how to design a privacy-preserving top-*k* spatial–textual query scheme over outsourced Cloud is of particular interest to enable secure LBS in smart cities.

Aiming at addressing the above challenges, in this paper, we propose a new scheme, called privacy-preserving spatio-textual index (Pystin), for PT*k*SK queries over outsourced Cloud. The proposed Pystin scheme employs Boneh–Goh–Nissim (BGN) homomorphic encryption [7] and novel hash bucket techniques to enable users to launch top-*k* LBS queries in smart cities, while preserving users' accurate location and keywords privacy against the Cloud server. In addition, to improve the performance, Pystin extends I^3 [8], a quad-tree-based spatio-textual indexing approach, which can provide efficient spatial pruning and make Pystin update efficient for spatial data [8], [9]. Specifically, the contributions of this paper are threefold.

1) We propose Pystin, a privacy-preserving top-*k* spatial–textual query over outsourced Cloud to enable secure LBS in smart cities, where a query user's accurate location is protected by the combination of BGN homomorphic encryption [7] and hash bucket techniques, and the privacy of textual information are persevered by a one-way hash function.
2) We integrate the quad-tree in I^3 as the spatio-textual indexing [8] into Pystin, which improves the query performance in Pystin.
3) We conduct extensive experiments against a Baseline secure approach (without an index), and a nonsecure spatio-textual index, to demonstrate the scalability, efficiency, and security of our proposed Pystin scheme.

The remainder of this paper is organized as follows. In Section II, we introduce our system model, security model, and design goal. In Section III, we recall some preliminaries. Then, in Section IV, we present our proposed Pystin scheme, followed by security analysis and performance evaluation in Sections V and VI, respectively. In Section VII, we discuss the related works. Finally, we draw our conclusions in Section VIII.

II. Models and Design Goal

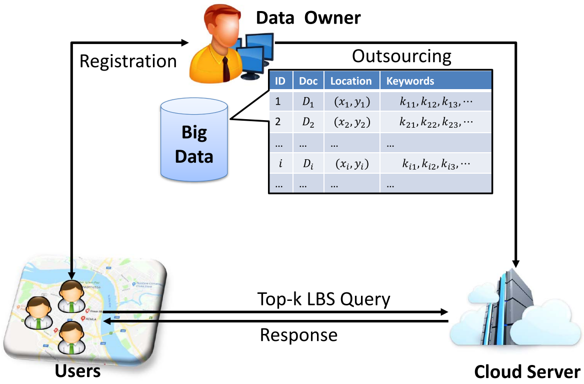
In this section, we formalize our system model, security model, and identify our design goal.

A. System Model

In our system model, we consider a typical Cloud-based top-*k* LBS query model, which includes a data owner, a Cloud server, and end users, as shown in Fig. 1.

1) Data Owner: Data owner is an LBS provider who owns a big data set $\mathbb{D} = \{D_1, D_2, \ldots\}$, where each document $D_i \in \mathbb{D}$ has the following format.

ID	Document content	Location	Keywords
i	$D_i = (D_i.loc, D_i.doc)$	(x_i, y_i)	$k_{i1}, k_{i2}, k_{i3}, \cdots$



ID	Doc	Location	Keywords
1	D_1	(x_1, y_1)	$k_{11}, k_{12}, k_{13}, \cdots$
2	D_2	(x_2, y_2)	$k_{21}, k_{22}, k_{23}, \cdots$
…	…	…	…
i	D_i	(x_i, y_i)	$k_{i1}, k_{i2}, k_{i3}, \cdots$
…	…	…	…

Fig. 1. System model of a top-*k* LBS query under consideration.

Since a Cloud server can provide powerful capabilities in storing and processing data, we assume the data owner is willing to outsource the big data set $\mathbb{D} = \{D_1, D_2, \ldots\}$ to the Cloud server for offering better LBS to end users.

2) Cloud Server: Cloud server is a powerful entity, which stores the outsourced data $\mathbb{D} = \{D_1, D_2, \ldots\}$ from the data owner, and also processes LBS queries from the authorized users. In order to improve the query efficiency, the Cloud server builds the index on the data, and also utilizes some data structures to organize the outsourced big data set $\mathbb{D} = \{D_1, D_2, \ldots\}$.

3) End Users: End users are a set of authorized users. After registering him/herself with the data owner, each end user obtains a query key from the data owner. Later, he/she can use the query key to send LBS queries such as "find the top three spicy curry restaurants near me," to the Cloud server.

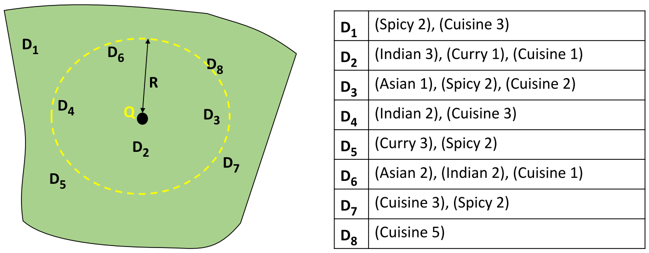
B. Security Model

In our security model, we consider the data owner to be trustable, while the Cloud server is semi-trusted, and follows *honest-but-curious* model. That is, the Cloud server will follow the protocol, but may be curious about the data owner's big data set $\mathbb{D} = \{D_1, D_2, \ldots\}$ and the end user's query privacy, including query interests and accurate location information of user. The end users will faithfully follow the protocol to launch the top-*k* LBS query, and there is no collusion between any end user and the Cloud server. In addition, we consider the following two assumptions: 1) we assume that there is a secure channel of communication between the data owner and the Cloud server over which the data owner outsources its data to the Cloud server and 2) we assume that the data owner provides the authorized keys to the Cloud server and the end users after performing authorization and access control measures.

Note that it is possible for an external attacker to launch other active attacks on data integrity and availability in more realistic scenarios. However, since we focus on efficient and PP-LBS query, those active attacks are beyond the scope of this paper and will be discussed in our future work.

C. Design Goal

Based on the above system model and security model, our design goal is to propose a new efficient and PT*k*SK query to



D_1	(Spicy 2), (Cuisine 3)
D_2	(Indian 3), (Curry 1), (Cuisine 1)
D_3	(Asian 1), (Spicy 2), (Cuisine 2)
D_4	(Indian 2), (Cuisine 3)
D_5	(Curry 3), (Spicy 2)
D_6	(Asian 2), (Indian 2), (Cuisine 1)
D_7	(Cuisine 3), (Spicy 2)
D_8	(Cuisine 5)

Spatial Dataset: D={D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8}
Query Point: Q with range R.

Keywords and their occurrences in the Dataset.

Fig. 2. Example of spatial database.

enable secure LBS in smart cities. In particular, the following two objectives should be achieved.

1) Our Proposed Scheme Should Be Privacy-Preserving: In order to adapt to the *honesty-but-curious* model, the Cloud server should not be able to read the contents and keywords in each document $D_i \in \mathbb{D}$. In addition, the Cloud server should not know the end user's query interest and the *accurate* location information. Note that, in order to enable the Cloud server to dynamically organize the big data set $\mathbb{D}$ according to each document D_i's location (x_i, y_i), we do not need to hide the location (x_i, y_i) to the Cloud server. Nevertheless, we still need to guarantee the Cloud server cannot know the accurate location of end user.

2) Our Proposed Scheme Should Be Efficient: In order to achieve the above privacy-preservation requirements, the Cloud server has to pay for additional computational cost to deal with the user's LBS query. In our proposed scheme, we aim to make the Cloud server's query response much efficient.

III. Preliminaries

In this section, we recall some preliminaries, including top-k spatio-textual queries, some existing spatio-textual indexing schemes, and BGN homomorphic encryption technique [7], which will serve as the basis for our proposed Pystin scheme.

A. Top-k Spatio-Textual Keyword Query

Let $\mathbb{D}$ be a spatial database such that $\mathbb{D} = \{D_1, D_2, \ldots\}$, as shown in Fig. 2. Each object D_i in $\mathbb{D}$ is defined as a tuple $(D_i.loc, D_i.doc)$. Here, $D_i.loc = (x_i, y_i)$ is the spatial information, i.e., $D_i.loc.latitude = x_i$, $D_i.loc.longitude = y_i$, and $D_i.doc$ is the textual description of the object containing keywords $k_{i1}, k_{i2}, k_{i3}, \ldots$, i.e.,

$$D_i.doc = \{k_{i1}, k_{i2}, k_{i3}, \ldots\}. \tag{1}$$

A query Q is also defined as a tuple $(Q.loc, Q.doc, k)$. $Q.loc$ is the query location (x_j, y_j), and $Q.doc$ is the set of keywords k_j in the query Q, where

$$Q.doc = \{k_1, k_2, k_3, \ldots\} \tag{2}$$

and k is the number of documents to be returned.

In the case of AND semantics, a document D_i is a candidate only when it contains all the keywords in the query; i.e., $\forall k_j \in Q.doc, k_j \in D_i.doc$. In the case of OR semantics, D_i is a candidate if it contains at least one keyword; i.e., $\exists k_j \in Q.doc, k_j \in D_i.doc$. A ranking function [8] that computes the relevance score of a spatial object D_i and query Q is defined as

$$D_i.Score_{total} = \alpha Score_{sp} + (1 - \alpha)Score_{tx} \tag{3}$$

where $Score_{sp}$ is the spatial relevance score of a document D_i, $Score_{tx}$ is the textual relevance score of D_i with respect to query Q, and the value of $\alpha \in [0, 1]$ determines the importance of spatial or textual relevance score in the overall score. Concretely, $Score_{sp}(D_i.doc, Q.doc)$ is defined as follows [8]:

$$Score_{sp}(D_i.doc, Q.doc) = 1 - \frac{Euc_{dist}(D_i.loc, Q.loc)}{dist_{MAX}} \tag{4}$$

where $Euc_{dist}(D_i.loc, Q.loc)$ is the Euclidean distance between $D_i.loc$ and $Q.loc$, and $dist_{MAX}$ is the maximum possible Euclidean distance between any two points in the space under consideration.

For $Score_{tx}(D_i.doc, Q.doc)$, we use a term frequency-inverse document frequency (TF-IDF) [10] scheme to model the documents and query in a vector-space model [11]. Further, to measure the similarity between D_i and Q, we use cosine similarity between the vectors representing the document $D_i.doc$ and query $Q.doc$. As a result, $Score_{tx}(D_i.doc, Q.doc)$ is computed as follows:

$$\begin{aligned} &Score_{tx}(D_i.doc, Q.doc) \\ &= \frac{\sum_{t \in D_i.doc \cap Q.doc} tfidf(D_i.doc.t, D_i.doc) tfidf(Q.doc.t, Q.doc)}{\sqrt{\sum_{t=1}^{|D_i.doc|} tfidf(D_i.doc.t, D_i.doc)^2} \sqrt{\sum_{t=1}^{|Q.doc|} tfidf(Q.doc.t, Q.doc)^2}} \end{aligned} \tag{5}$$

where t refers to a term (keyword) either in the document $D_i.doc$ or the query $Q.doc$. $D_i.doc.t$ refers to a term in $D_i.doc$ and $Q.doc.t$ refers to a term in $Q.doc$. For a given term τ and document doc, the TF-IDF weight, $tfidf(\tau, doc)$ is calculated as a product of term-frequency (tf) and inverse document frequency (idf) as

$$tfidf(\tau, doc) = tf(\tau, doc) idf(\tau, doc) \tag{6}$$

where

$$tf(\tau, doc) = \frac{f(\tau, doc)}{|doc|} \tag{7}$$

$$idf(\tau, doc) = \log \frac{|\mathbb{D}|}{|\{doc' \in \mathbb{D} | \tau \in doc'\}|}. \tag{8}$$

Further, $|\mathbb{D}|$ is the number of documents in $\mathbb{D}$. $f(\tau, doc)$ is the number of times term τ appears in a document doc and $|doc|$ is the number of terms document doc contains. $|\{doc' \in \mathbb{D} | t \in doc'\}|$ is the number of documents in $\mathbb{D}$ in which the term τ appears. Note, other variations of tf and idf formulation [12] can be used as well.

B. Spatio-Textual Index

Inverted files are the most popular one for textual indexing, and R-tree and its variations are also commonly used for spatial indexing. Therefore, for spatio-textual indices, it is natural for us to combine these two approaches. Most of the current state-of-the-art hybrid indices follow this pattern. For example, IR-tree [4] index combines R-trees and inverted files, and each node in R-tree contains pointer to an inverted file containing details of the objects. I^3 index [8] incorporates inverted files with quadtrees for efficient spatial pruning. We use I^3 as basis for our proposed index structure in our proposed scheme.

C. BGN Homomorphic Encryption

The BGN homomorphic encryption technique has been widely studied in privacy preserving scenarios [7], and mainly consists of three algorithms: 1) key generation; 2) encryption; and 3) decryption. Since BGN is built upon the bilinear pairing with composite order, we first recall the properties of bilinear pairing with composite order. Let p and q be two large primes of the same length, i.e., the bit length $|p| = |q|$, and $N = pq$. Two groups $(\mathbb{G}, \mathbb{G}_T)$ of composite order N are called *bilinear map with composite order* if there exists a computable mapping $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ with the following three properties [7].

1) *Bilinearity:* $e(g^a, h^b) = e(g, h)^{ab}$ for any $(g, h) \in \mathbb{G}^2$ and $a, b \in \mathbb{Z}_N$ where $\mathbb{Z}_N \in \{0, 1, 2, \ldots, N-1\}$.
2) *Nondegeneracy:* There exists $g \in \mathbb{G}$ such that $e(g, g)$ is with the order N in $\mathbb{G}_T$; and
3) *Computability:* There exists an efficient algorithm to compute $e(g, h) \in \mathbb{G}_T$ for all $(g, h) \in \mathbb{G}$.

Let $\mathbf{g}$ be a generator of $\mathbb{G}$, then $g = \mathbf{g}^q \in \mathbb{G}$ can generate the subgroup $\mathbb{G}_p = \{g^0, g^1, \ldots, g^{p-1}\}$ of order p, and $g' = \mathbf{g}^p \in \mathbb{G}$ can generate the subgroup $\mathbb{G}_q = \{g'^0, g'^1, \cdots, g'^{q-1}\}$ of order q in $\mathbb{G}$. The subgroup decision problem is assumed hard, and we can use it to build the BGN homomorphic encryption [7]. Next, we briefly explain the three algorithms.

1) Key Generation: Given the security parameter κ, composite bilinear parameters $(N, g, \mathbb{G}, \mathbb{G}_T, e)$ are generated by $\mathcal{CG}en(\kappa)$, where $N = pq$ and p, q are two κ-bit prime numbers, and $g \in \mathbb{G}$ is a generator of order n. Set $h = g^q$, then h is a random generator of the subgroup of $\mathbb{G}$ of order p. The public key is $pk = (N, \mathbb{G}, \mathbb{G}_T, e, g, h)$, and the corresponding private key is $sk = p$.

2) Encryption: We assume the message space consists of integers in the set $\mathbb{S} = \{0, 1, \ldots, \Delta\}$ with $\Delta \ll q$. To encrypt a message $m \in \mathbb{S}$, we choose a random number $r \in \mathbb{Z}_N$, and compute the ciphertext $c = E(m, r) = g^m h^r \in \mathbb{G}$.

3) Decryption: Given the ciphertext $c = E(m, r) = g^m h^r \in \mathbb{G}$, the corresponding message can be recovered by the private key p. Observe that $c^p = (g^m h^r)^p = (g^p)^m$. Let $\hat{g} = g^p$. To recover m, it suffices to compute the discrete log of c^p base $\hat{g}$. Since $0 \leq m \leq \Delta$, the expected time is around $O(\sqrt{\Delta})$ when using the Pollard's lambda method [13].

The BGN encryption has the following addition and multiplication homomorphic properties.

1) *Addition in $\mathbb{G}$:* Given $E(m_1, r_1) \in \mathbb{G}$ and $E(m_2, r_2) \in \mathbb{G}$, we have $E(m_1, r_1) \cdot E(m_2, r_2) = E(m_1 + m_2, r_1 + r_2) \in \mathbb{G}$. For simplicity, we omit the random items, and we have $E(m_1) \cdot E(m_2) = E(m_1 + m_2)$.
2) *Multiplication in $\mathbb{G}$:* Given $E(m_1) \in \mathbb{G}$ and $m_2 \in \mathbb{S}$, we have $E(m_1)^{m_2} = E(m_1 \cdot m_2) \in G$.
3) *Multiplication From $\mathbb{G}$ to $\mathbb{G}_T$:* Given $E(m_1), E(m_2) \in \mathbb{G}$, we have $e(E(m_1), E(m_2)) = E_T(m_1 \cdot m_2) \in \mathbb{G}_T$, where $E_T(\cdot)$ denotes a ciphertext in $\mathbb{G}_T$.

IV. Our Proposed Pystin Scheme

In this section, we present our Pystin scheme, which mainly consists of five parts: 1) system initialization; 2) big data outsourcing; 3) index construction; 4) end user top-*k* LBS query (PT*k*SKQ); and 5) Cloud server query response.

A. System Initialization

As the data owner is a trustable entity in our system model, she bootstraps the whole system in the system initialization phase. Specifically, given the security parameter κ, the data owner uses $\mathcal{CG}en(\kappa)$ to generate the bilinear parameters $(N, g, \mathbb{G}, \mathbb{G}_T, e)$, where $N = pq$. Then, the data owner sets the BGN public key $pk = (N, \mathbb{G}, \mathbb{G}_T, e, g, h)$ and the private key $sk = p$, where $h = g^q$. Further, the data owner chooses a secure symmetric encryption algorithm $Enc(.)$, e.g., AES, and a cryptographic hash function $H(.)$, e.g., SHA-1, and also chooses random numbers $s, s_1, s_2, t \in \mathbb{Z}_N$ as secret keys. Finally, the data owner keeps (p, s, s_1, s_2, t) secret, and publishes $pk = (N, \mathbb{G}, \mathbb{G}_T, e, g, h)$, $Enc(.)$, and $H(.)$.

User Registration: When an end user registers him/herself to the LBS services provided by the data owner, the data owner will authenticate the user and authorize the access key $AK = (s, s_1, g^{s_2}, g^{s_2^2})$ to the user, so that the latter can use the access key to launch the LBS query to the Cloud server.

B. Big Data Outsourcing

As the Cloud server is *honest-but-curious*, before outsourcing $\mathbb{D} = \{D_1, D_2, \ldots\}$ to the Cloud server, the data owner secures the documents and the interest keywords in $\mathbb{D}$ so that the Cloud server can process the PP-LBS query. Concretely, big data outsourcing includes two parts: 1) data sourcing and 2) hash bucket construction.

1) Data Outsourcing: The data owner first runs the following steps for each document $D_i \in \mathbb{D}$ before outsourcing.

Step 1: The data owner uses the secret key s_1 to compute $ED_i = Enc_{s_1}(D_i)$.

Step 2: The data owner then chooses a random number $r_{4i} \in \mathbb{Z}_N$ and uses the secret key s_2 to compute $C_{4i} = g^{s_2(s_2+x_i^2+y_i^2)} \cdot h^{r_{4i}}$, where (x_i, y_i) is the location of D_i.

Step 3: For each keyword k_{ij}, the data owner uses the secret key s to compute $hk_{ij} = H(k_{ij}||s)$.

Step 4: The data owner formats the new form of D_i as follows.

ID	Doc. Content	Location	Keywords
i	ED_i	$(x_i, y_i), C_{4i}$	$hk_{i1}, hk_{i2}, hk_{i3}, \cdots$

After processing all documents $\mathbb{D} = \{D_1, D_2, \ldots\}$ into the encrypted dataset $\mathbb{ED} = \{ED_1, ED_2, \ldots\}$, the data owner outsources $\mathbb{ED}$ to the Cloud server.

2) Hash Bucket Construction: In order to enable the Cloud server to process LBS query over encrypted $\mathbb{ED}$, the data

Algorithm 1: Hash Bucket Builder

```
Input: a private function $F(x) = e(g, g)^{pt(2s_2+x)}$ defined by the
       data owner, and an integer $u \geq 1$
Output: hash buckets $\mathcal{HB}_{u\cdot 1000}$
for $i = 0; i <= u \cdot 1000; i++$ do
    for $j = i; j <= u \cdot 1000; j++$ do
        compute $R = i^2 + j^2$;
        if $(u - 1) \cdot 1000 \leq \sqrt{R} < u \cdot 1000$ then
            store $H(F(x))$ in $\mathcal{HB}_{u\cdot 1000}$
        else if $\sqrt{R} > u \cdot 1000$ then
            continue;
sort all items in $\mathcal{HB}_{u\cdot 1000}$ and remove the duplicated items
return $\mathcal{HB}_{u\cdot 1000}$
```

owner also builds and outsources a set of hash buckets to the Cloud server. Hash buckets are used by the Cloud server to find the spatial proximity between a query and a document. The data owner authorizes a processing key $g^{s_2^{-1}tp}$ to the Cloud server, and also generates and outsources a set of hash buckets, e.g., $(\mathcal{HB}_{1000}, \mathcal{HB}_{2000}, \ldots, \mathcal{HB}_{10\,000})$, to the Cloud server. Each hash bucket $\mathcal{HB}_{u\cdot 1000}$ is generated by running Algorithm 1 with the input of u. With the hash bucket $\mathcal{HB}_{i\cdot 1000}$, $i = 1, 2, \ldots, 10$, it is possible for the Cloud server to determine whether the Euclidean distance $l_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ of two points (x_1, y_1) and (x_2, y_2) is within the range $[(i-1)\cdot 1000, i\cdot 1000)$ while without disclosing the accurate Euclidean distance l_{12}. The correctness is as follows.

If the Euclidean distance of two points (x_1, y_1) and (x_2, y_2) is $l_{12} = w$, which is really within the range $[(i - 1) \cdot 1000, i \cdot 1000)$, from the construction of $\mathcal{HB}_{i\cdot 1000}$, the hash value $H(e(g, g)^{pt(2s_2+w)})$ should be in $\mathcal{HB}_{i\cdot 1000}$. At the same time, if we only know the value $e(g, g)^{pt(2s_2+w)}$, due to the hardness of discrete logarithm problem and without knowing the secret keys (p, t, s_2), we of course cannot get the value of w. Therefore, the correctness is satisfied. The hash bucket technique can help the Cloud server roughly determine the current location of a query user is within a range, but cannot know the accurate location, thus the query user's accurate location can be protected.

C. Index Construction

The Cloud server builds the Pystin secure index, as shown in Fig. 3, and our implementation is based on I^3 indexing scheme [8]. The secure index is comprised of an inverted list of encrypted keywords. These keywords are categorized as dense and sparse. Given a threshold λ, a keyword is dense in a cell if it is frequency exceeds λ, else the keyword is said to be sparse or nondense in the keyword cell. The entry in the wordmap for a sparse keyword points directly to the page in disk containing the tuple. However, for dense keywords, the documents are arranged in a quad-tree, as seen in Fig. 3. This index maintains a head file for dense keywords. For each encrypted keyword dense in a keyword cell, a summary node with summary information $\mathscr{E} = <\mathscr{E}.sig, \mathscr{E}.max_s>$ is created. $\mathscr{E}.sig$ contains information about the documents that contain the encrypted keyword, and $\mathscr{E}.max_s$ is the upper bound textual score for the keyword.

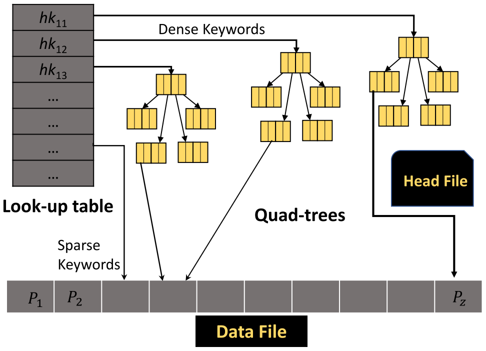


Fig. 3. Various components of the Pystin secure index.

The signature file *sig* is a bitmap of length n that aggregates all the documents containing encrypted keyword hk in cell C. A cell can be pruned in the following cases.

1) If there is no intersection between the signatures of different keywords in a cell, i.e., the cell does not contain any document that contains all the query keywords. In such a case, the cell can be simply pruned.
2) If there is an intersection between the signatures of different keywords in a cell, the result is stored in *sig* and an intersection of $\mathscr{C}.docs$ and *sig* is performed. In case there is no intersection, the search space is pruned.
3) It is straightforward to calculate the spatial relevance scores from the cell boundary points with respect to query point using Algorithm 5. Since, the maximum spatial relevance score can then be calculated by comparing these scores, we can calculate the upper bound score of keyword hk for cell C. If this upper bound aggregate score is smaller than the kth-score in the top-k results, we simply prune this cell as well.

D. End User Top-k LBS Query

With the access key $AK = (s, s_1, g^{s_2}, g^{s_2^2})$, an end user can launch some top-k LBS query, i.e., PTkSKQ, to the Cloud server. Suppose the user is located at location (x_0, y_0) and has interest keywords $(k_{01}, k_{02}, k_{03}, \ldots)$. Then, the following steps will be performed by the end user for the top-k LBS query.

Step 1: The end user chooses three random numbers $r_1, r_2, r_3 \in \mathbb{Z}_N$, and computes

$$C_1 = h^{r_1} \cdot g^{s_2^2} g^{s_2(x_0^2+y_0^2)}, C_2 = h^{r_2}/g^{s_2\cdot 2x_0}, C_3 = h^{r_3}/g^{s_2\cdot 2y_0}.$$

Step 2: For each interest keyword $k_{0j} \in (k_{01}, k_{02}, k_{03}, \ldots)$, the end user utilizes the access key s to compute $hk_{0j} = H(k_{0j}||s)$.

Step 3: After that, the end user sends the query to the Cloud server, where the query is formed by $(k, C_1, C_2, C_3, hk_{01}, hk_{02}, hk_{03}, \ldots)$.

Note that, upon receiving the secured top-k results from the Cloud server, the end user can decrypt the results by using the access key s_1. In the following, we take a close look on how the Cloud server responds the end user's query.

```
Algorithm 2: Query Processing [8]
   Input: Secure top-$k$ LBS query
          $(k, C_1, C_2, C_3, hk_{01}, hk_{02}, hk_{03}, \cdots)$, dataset
          $\mathbb{ED} = \{ED_1, ED_2, \cdots\}$, Pystin secure index
   Output: vector $R$ =[$< doc_1, s_1 >, < doc_2, s_2 >, \cdots$ ] of top-$k$
          documents sorted according to their relevance scores
1  initialize a root candidate and push it into a priority Queue $PQ$
2  set $\delta \leftarrow 0$, $s \leftarrow 0$, $k \leftarrow 5$, $count \leftarrow 0$, $\mathscr{C}.Score_{ub} \leftarrow 0$
3  while $PQ$ is not empty do
4      pop the first candidate $\mathscr{C}$ from the $PQ$
5      if $\mathscr{C}.Score_{ub} <= \delta$ and $\delta != 0$ then
6          break
7      if $\mathscr{C}.denseKwds$ is empty then
8          processSparse($\mathscr{C}.docs$)
9      else
10         processDense($\mathscr{C}.denseKwds$,$PQ$)
11 return $R$
```

```
Algorithm 3: ProcessSparse
   Input: $\mathscr{C}.docs$
   Output: vector $R$
1  for $doc \in \mathscr{C}.docs$ do
2      Calculate the relevance score $s$ for $doc$
3      if $count < k$ then
4          insert $doc$ into sorted vector $R$
5          increment $count$
6          if $count = k$ then
7              $\delta \leftarrow s$
8      else
9          if $s > \delta$ then
10             $\delta \leftarrow s$
11             delete the last element in $R$
12             insert doc into sorted vector $R$
```

```
Algorithm 4: ProcessDense
   Input: $\mathscr{C}.denseKwds$,$PQ$
   Output: $PQ$
1  for child cell $C'_i$ in $\mathscr{C}.C$ do
2      create a new candidate $\mathscr{C}'$
3  for keyword $hk_{ij}$ in $\mathscr{C}.denseKwds$ do
4      if $hk_{ij}$ is dense in $C'_i$ then
5          insert $hk_{ij}$ into $\mathscr{C}'.denseKwds$
6      else
7          retrieve tuples $\{\mathscr{T}\}$ in keyword cell $< hk_{ij}, C'_i >$
8          for $\mathscr{T} \in \{\mathscr{T}\}$ do
9              update $\mathscr{C}'.docs$
10 if prune( $\mathscr{C}'$=TRUE) then
11     continue
12 updateupperscore( $\mathscr{C}$)
13 $PQ$.add( $\mathscr{C}'$)
```

E. Cloud Server LBS Query Response

In order to efficiently process the end users' queries, the Cloud server first builds the secure index as shown in Fig. 3. Then, based on the secure index and the precomputed hash-buckets, the server ranks the candidate documents according to their relevance scores and returns the top-k results to the end user.

1) Query Processing: The query processing algorithm (Algorithm 2) follows a top-down approach starting from a root cell $\mathbb{C}$. A sparse keyword can be processed as explained in Algorithm 3. The tuples are loaded from the datafile. As the document location is known, using Algorithm 5, the spatial score for the documents can be computed. For the dense keywords, we perform the query processing as described in Algorithm 4.

A candidate cell is defined as

$$\mathscr{C} = <\mathscr{C}.C, \mathscr{C}.denseKwds, \mathscr{C}.docs, \mathscr{C}.Score_{ub}>$$

where $\mathscr{C}.C$ represents cell C, the current search region, where $\mathscr{C}.denseKwds$ is a list of dense keywords in cell C. $\mathscr{C}.Score_{ub}$ is the upper bound score in the current cell. Let δ be the kth score of the current top-k results. The result of the algorithm is a vector R of top-k documents ranked according to their relevance scores.

A priority queue PQ of candidates is maintained (Algorithm 2) in descending order of their textual relevance scores. Initially, a candidate for the root cell $\mathbb{C}$ is pushed into PQ. Subsequently, a candidate $\mathscr{C}$ with the maximum upper bound score is popped from the priority queue PQ. If $\mathscr{C}.Score_{ub} <= \delta$, we stop the search as the rest of the candidates are pruned (lines 5 and 6 in Algorithm 2). Otherwise, we check if $\mathscr{C}.denseKwds$ is empty. If the current cell does not contain any dense keywords, then all the related tuples have been loaded from the disk and stored in $\mathscr{C}.docs$. The final relevance score s of these documents can be calculated directly and δ is updated accordingly (lines 1–12 in Algorithm 3). But, in case there exist keywords those are dense in $\mathscr{C}.C$, we zoom into the child cells and create a new candidate $\mathscr{C}'$ for each child cell. $\mathscr{C}'.C$ is set to the child cell C'_i (lines 1 and 2 in Algorithm 4). For each dense keyword in $\mathscr{C}'.denseKwds$, if it is no longer dense in the child cell C'_i, we load the related tuples from the disk and remove the keyword from $\mathscr{C}'.denseKwds$. For each tuple $\mathscr{T}$, we update ED_i's textual relevance score to include $\mathscr{T}.s$ and the corresponding document ED_i is added to $\mathscr{C}.docs$ (lines 4 and 5 in Algorithm 4). Otherwise, if the query keyword hk_{ij} is dense in C'_i, we insert hk_{ij} into $\mathscr{C}'.denseKwds$ (lines 7–9 in Algorithm 4). After that, as explained in Section IV-C, we see if the new candidate $\mathscr{C}'$ can be pruned. If not, its upper bound score is calculated and updated. It is then pushed into the priority queue (lines 10–13 in Algorithm 4). The algorithm continues until all the candidate cells are exhausted, at which point we also have our top-k results.

2) Calculating the Spatial Relevance Score: The Cloud server first computes C from the secure location information received from the data owner, i.e., C_{4i}, and from the end user, i.e., (C_1, C_2, C_3), and then computes f using $g^{s_2^{-1}tp}$ provided to it by the data owner, as described in Algorithm 5.

We know that if the document D_i is within the range, then the precomputed hashbuckets will contain the value f. The Cloud server checks each hashbucket in

Algorithm 5: Rough Range Finder

```
Input: the processing key $g^{s_2^{-1}tp}$,
the hashbuckets $(\mathcal{HB}_{1000}, \mathcal{HB}_{2000}, \cdots, \mathcal{HB}_{10,000})$,
secure top-$k$ LBS query $(k, C_1, C_2, C_3, hk_{01}, hk_{02}, hk_{03}, \cdots)$,
and $(x_i, y_i), C_{4i}$ in one document $ED_i$ in Cloud
Output: distance range $dr$
1 compute
      $C = C_1 \cdot C_2^{x_i} \cdot C_3^{y_i} \cdot C_{4i}$
      $= g^{s_2 \cdot (2s_2 + (x_0 - x_i)^2 + (y_0 - y_i)^2)} \cdot h^r$ for some random $r$
2 compute
      $f = e(C, g^{s_2^{-1}tp}) = e(g, g)^{pt \cdot (2s_2 + (x_0 - x_i)^2 + (y_0 - y_i)^2)}$
  for $i = 1; i <= 10; i++$ do
3     if $H(f) \in \mathcal{HB}_{i \cdot 1000}$ then
4         return $dr = i$ /* if $i = 1$, $dr = 1$ indicates the range
          $[0, 1000)$; else if $i > 1$, $dr = i$ indicates the range
          $[(i-1) * 1000, i * 1000)$ */
5 return $dr = -1$ /*showing the range is out of 10,000 */
```

$\{\mathcal{HB}_{1000}, \ldots, \mathcal{HB}_{10\,000}\}$ for the presence of f (lines 3 and 4). Depending upon which bucket contains f, the rough range of the document can be known which is used to calculate the spatial relevance score.

The Cloud server then combines the spatial and textual relevance scores to calculate the total relevance scores and return the top-k results as described in Algorithm 2.

V. Security Analysis

In this section, we analyze the security of our proposed scheme. Concretely, under the assumptions made before, i.e., the honest-but-curious model, no collusion, etc., we check whether our scheme can achieve privacy-preserving spatial keyword query against the Cloud server.

The Data Owner's Documents $\mathbb{D} = \{D_1, D_2, \ldots\}$ Are Privacy-Preserving for Keywords in Cloud: In Pystin, before outsourcing $\mathbb{D} = \{D_1, D_2, \ldots\}$ to Cloud, each document $D_i \in \mathbb{D}$ will be encrypted into the following format.

ID	Doc. Content	Location	Keywords
i	$ED_i = Enc_{s_1}(D_i)$	$(x_i, y_i), C_{4i}$	$hk_{i1}, hk_{i2}, \cdots$

For the document content $ED_i = Enc_{s_1}(D_i)$, without knowing the secret key s_1, the Cloud server cannot know D_i directly from ED_i. Each keyword $hk_{ij} = H(k_{ij}||s)$ is the hash value of $k_{ij}||s$, where k_{ij} is the real keyword, and s is a secret key. Because of the oneway-ness of hash function H, the real keyword k_{ij} will not be revealed from hk_{ij}. The Cloud server may find the keywords by other attacks such as frequency attacks. Note that, in order to facilitate the calculation of the textual relevance score in (5), i.e.,

$$Score_{tx}(D_i.doc, Q.doc) = \frac{\sum_{t \in D_i.doc \cap Q.doc} tfidf(D_i.doc.t, D_i.doc)\, tfidf(Q.doc.t, Q.doc)}{\sqrt{\sum_{t=1}^{|D_i.doc|} tfidf(D_i.doc.t, D_i.doc)^2}\sqrt{\sum_{t=1}^{|Q.doc|} tfidf(Q.doc.t, Q.doc)^2}}$$

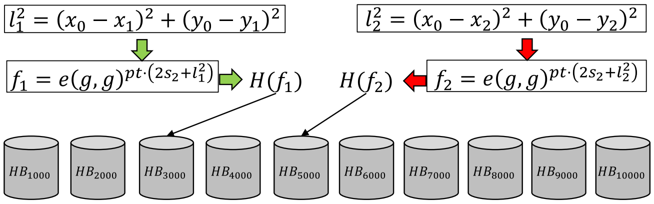

$l_1^2 = (x_0 - x_1)^2 + (y_0 - y_1)^2$
$l_2^2 = (x_0 - x_2)^2 + (y_0 - y_2)^2$
$f_1 = e(g,g)^{pt \cdot (2s_2 + l_1^2)}$
$H(f_1)$
$H(f_2)$
$f_2 = e(g,g)^{pt \cdot (2s_2 + l_2^2)}$
HB_{1000}
HB_{2000}
HB_{3000}
HB_{4000}
HB_{5000}
HB_{6000}
HB_{7000}
HB_{8000}
HB_{9000}
HB_{10000}

Fig. 4. Preserving the end user's accurate location (x_0, y_0) with the rough range finder algorithm. Since $H(f_1)$, $H(f_2)$ lie in HB_{3000} and HB_{5000}, respectively, the Cloud server only knows $l_1 \in [2000, 3000)$ and $l_2 \in [4000, 5000)$.

it is not a good strategy to consider the indistinguishability security on hk_{ij}, as it will make the textual relevance score calculation very slow. Therefore, we only consider the real keyword k_{ij} to be one-way secure in Pystin for achieving a tradeoff between security and efficiency. Similarly, in order to facilitate the calculation of spatial proximity in (3), Pystin does not hide the location (x_i, y_i) for each document D_i, because the disclosure of (x_i, y_i), as we will discuss below, will not help the Cloud server to gain the end user's accurate location. Therefore, the data owner's documents $\mathbb{D} = \{D_1, D_2, \ldots\}$ are privacy-preserving for keywords in the Cloud.

The End User's Query Keywords Are Privacy-Preserving, and the End User's Accurate Location Will Also Be Protected: In the end user's query $(C_1, C_2, C_3, hk_{01}, hk_{02}, hk_{03}, \ldots)$, each keyword $hk_{0i} = H(k_{0i}||s)$ is a hash value of $k_{0i}||s$. As we discussed above, each real keyword k_{0i} can be one-way secure. Since the end user's location (x_0, y_0) is encrypted with BGN encryption [7] into $C_1 = h^{r_1} \cdot g^{s_2^2} g^{s_2(x_0^2 + y_0^2)}$, $C_2 = h^{r_2} / g^{s_2 \cdot 2x_0}$, $C_3 = h^{r_3} / g^{s_2 \cdot 2y_0}$, without knowing the private key in BGN encryption, the Cloud server cannot get the location (x_0, y_0) from (C_1, C_2, C_3). As the Cloud server knows each document D_i's location (x_i, y_i), if the Cloud server knows the distance l_1 between the location (x_0, y_0) and the location (x_1, y_1) of D_1, and the distance between the location l_2 between the location (x_0, y_0) and the location (x_2, y_2) of D_2, the Cloud server can possibly compute (x_0, y_0) from $(x_0 - x_1)^2 + (y_0 - y_1)^2 = l_1^2$ and $(x_0 - x_2)^2 + (y_0 - y_2)^2 = l_2^2$. However, the Cloud server cannot get the accurate distances l_1 and l_2 by using Algorithm 5.

As shown in Fig. 4, if we consider the ranges of l_1 and l_2 to be 1000 in Algorithm 5, the Cloud server can guess the correct l_1 and l_2 both only with probability $(1/10^6)$. As a result, the Cloud server cannot know the accurate values of the location (x_0, y_0), and thus the end user's accurate location is also protected. Obviously, there is a tradeoff between the query accuracy and user's accurate location privacy. Note that, we do not protect the access pattern in this paper, as it will lower the performance greatly.

VI. Performance Evaluation

In this section, we conduct extensive experiments to evaluate the performance, efficiency, and scalability of our proposed Pystin scheme.

A. Experimental Setup

1) Algorithms: In order to do a fair comparison for our proposed Pystin scheme, we choose a Baseline secure

TABLE I
APPROACHES COMPARED IN THE EXPERIMENTS

Approaches	Description
I^3	Scalable plaintext index for spatial top-k query
Pystin	Our approach for privacy preserving spatial top-k query
Baseline	Implements the security model of Pystin without any index.

TABLE II
DATASETS USED IN THE EXPERIMENTS

Dataset name	Num.of tuples	Average num. of keywords	Max num. of keywords	Average doc. length
TW200k	200,000	5.08	30	25.58
TW2mi	2,000,000	5.06	70	25.55
GN	2,275,002	7	25	43

approach (without an index) called Baseline, and a nonsecure spatio-textual index I^3 that has been regarded as one of the fastest indices for T*k*SK queries. Note that, I^3 is highly scalable and has been shown to be more efficient than the other state-of-the-art solutions such as IR-tree [4]. Our implementation of Baseline is a linear scan of spatial objects in document space. It uses the same security model as Pystin, to process privacy preserving top-*k* query. The details of the three algorithms (approaches) are shown in Table I.

2) Data and Queries: For experimentation, we choose two variations of Twitter dataset and geographic names (GNs) dataset, which are real-world spatio-textual datasets. In specific, they were chosen to study the behavior of Pystin with varying the size and nature of the data. A summary of the datasets is presented in Table II. It is worth noting that Twitter dataset is widely distributed across the whole geographic region. Since tweets are constrained by a size of 140 characters, most of the tweets have all unique words. Therefore, we use two variations of Twitter dataset containing 200 000 and 2 million records, which were geo-tagged by Chen *et al.* [14] using a real road network dataset [15]. In addition, GN dataset [16] is the United States standard for geographic nomenclature, which is mostly confined to the U.S. and contains highly frequent keywords.

For spatio-textual query sets, we generate queries based on a given dataset. Each query set for a dataset consists of 100 queries. An important parameter used during the query generation is selectivity. For example, a 5% selectivity implies that there is a 5% chance that the current query Q contains all keywords from a known document object D_i in the dataset when, $|Q.doc| \leq |D_i.doc|$, or w keywords when, $w = |D_i.doc|$ and $|Q.doc| > |D_i.doc|$. Otherwise, there is a 95% chance that the query location will be random and the query keywords are selected randomly from the dictionary for that given dataset.

3) Environmental Setup: A server with 3.00-GHz GenuineIntel (8 core), 16-GB RAM was used to conduct all the experiments. All index structures reside on disks. The code for I^3 was generously made available by the authors. All indices and algorithms are implemented in Java and the JVM heap is set to 12 GB.

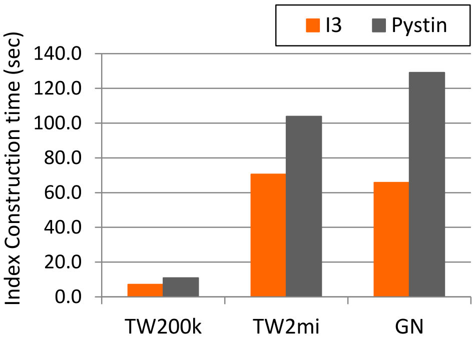


Fig. 5. Index construction time.

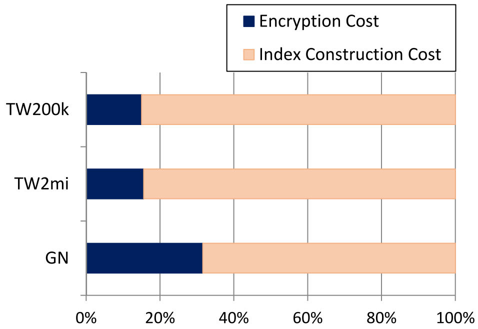


Fig. 6. Cost of secured index construction.

B. Index Construction Performance

In this section, we present the time and storage requirements of Pystin and how they compare against I^3. As explained earlier, we choose I^3 as the benchmark for this set of experiments as it is proven to be very efficient and faster than most of the spatio-textual indices for big data available today. Since Baseline (as in Table I) is not an index-based approach, it is omitted from the experiments in this section.

1) Index Construction Time: Fig. 5 shows the index construction time for each dataset. For the smaller dataset TW200k, the performance of Pystin is comparable to that of I^3. For the larger datasets TW2mi and GN, Pystin takes up to 2.6× longer than I^3. This is expected, as the index construction time of Pystin includes the cost of security related operations. Su *et al.* [17] also noted the same in terms of the index construction time of their secure index and the spatio-textual index IR-tree. On comparing index construction time of TW2mi and GN dataset, TW2mi takes more time to construct the index as the number of unique keywords are more in TW2mi dataset.

2) Cost of Security Operations: The index construction time for Pystin shown in Fig. 6 includes the encryption time. We can see that the encryption cost is approximately around 30% of the total index construction time. The cost of index construction depends on the size of the data, i.e., the number of objects in the dataset and the length of the keywords. Since Pystin encrypts all the data and query keywords to a fixed size code, index construction time depends solely on the number of objects in the data space.

3) Storage Cost: In Fig. 7, we report the sizes of the different indices along with the data-file size. Index size greatly

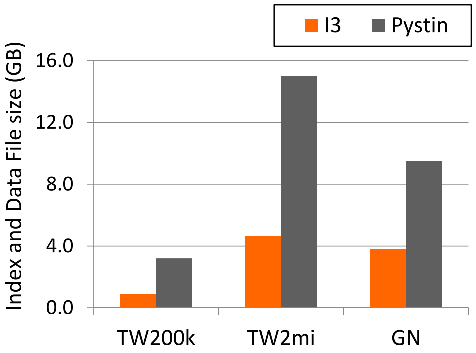


Fig. 7. Storage cost.

TABLE III
PARAMETER SETTINGS

Parameter	Settings
k	1, 2, 3, 4, 5
α	0.1, 0.3, 0.5, 0.7, 0.9
Radius	1, 10, 20, 100
Selectivity	5, 10, 20,30

varies with the number of objects which is evident from the huge difference in the index sizes of TW200k and TW2mi. Given the fact that the index is stored in Cloud, the size of the index is not a concern.

C. Query Performance Comparison

We evaluate and report the query performance of the three approaches Pystin, I^3, and Baseline in this section. Experiments are performed under different parameter settings as described in Table III on the three datasets TW200k, TW2mi, and GN to evaluate the efficiency of the approaches.

In general, of the three algorithms that were evaluated, the query performance of Pystin was significantly better than that of the Baseline, and was worse than that of I^3. This trend is observed with all the three datasets, as shown in Fig. 8(a)–(c). This is completely in line with our expectation, as Pystin index does much better than the secure approach Baseline because it does not use an index. Also, Pystin's query performance is worse than that of I^3 index, because I^3 is not a secure index and hence does not pay for the cost of security operations.

1) Varying k in Top k: In Fig. 8, we examine the query performance while varying k as 1, 2, 3, 4, and 5. In all the three approaches, the same ranking function calculates the total score for all the objects in the document space. Hence, varying k does not have an impact on the query performance in our experiments.

2) Varying Selectivity: We experimented with different values of *selectivity* in the query sets, such as 5%, 10%, 20%, and 30%. Fig. 9 presents the query performance in the three approaches. We can see that I^3 and Pystin are both sensitive to the change in *selectivity*. As the value of *selectivity* increases, there are more queries for which the spatial objects are scanned and the score is calculated. However, this increase in the cost of query processing is not significant for I^3. For Pystin, though the cost increases up to 2.16× when *selectivity* increases from 5% to 30%, it is not very significant in terms of the overall query performance over Baseline as seen in Fig. 9. It can be observed that Pystin takes a higher amount of time for query processing compared to I^3. This is due to the fact that Pystin involves use of BGN encryption in the Cloud to calculate the spatial relevancy score. However, it is also noticeable that the query performance time remains under 1 s for Pystin. Our scheme is 100 times faster than the Baseline algorithm for all the three datasets.

3) Varying α: Recall, α is a weight to specify the importance of spatial and textual relevance scores in the overall score in (3). For the Twitter datasets where the average number of keywords in a spatial object is relatively low (as shown in Table II), α does not have an impact on the query performance. The average number of keywords per spatial object in slightly more in the GN dataset compared to the Twitter datasets, but not significantly larger. In Fig. 10, we report that the query performance remains unaffected by the value of α in our experiments.

D. Scalability of Pystin

In this section, we further evaluate the scalability of our proposed Pystin scheme.

1) Varying Query Radius: As mentioned earlier, Pystin uses bucketization to find whether or not an object is in the range. The cost of creating hashbuckets is quadratic to the maximum radius. However, this is a one-time cost incurred in the preprocessing step and it does not contribute to the query processing cost. The query processing times while varying the radius from 5 to 100 km, are shown in Fig. 11. As the increase in radius does not require any additional computation and the fact that Pystin calculates the score for all the objects in the document space, there is no additional cost of query processing associated with the increase in the radius, which makes Pystin scalable. The difference in query execution times are primarily due to the variation in dataset sizes.

2) Effect of Increase in Number of Objects: Figs. 8–10 show the results of varying the size of the datasets. These experiments confirm that Pystin is scalable with respect to the dataset size. Increasing the number of spatial objects leads to an increase in number of nodes in the Quadtrees of Pystin. However, this contributes only to a sublinear increase in the query processing time.

3) Effect of Keyword Length: Our experimental evaluation has demonstrated that the performance of Pystin is significantly better than a privacy-preserving Baseline approach (i.e., Baseline) that does not use an index. Pystin encrypts all the query keywords and dataset to fixed length codes. Therefore, an increase in the keyword length does not have any effect on the query performance.

VII. RELATED WORK

In this section, we discuss some related works, including spatial–textual index, secure spatial query, secure textual query, and secure spatio-textual query.

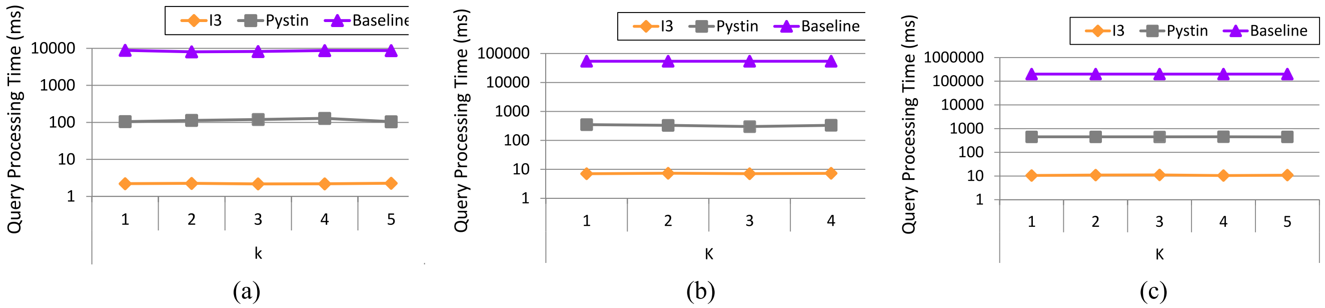


Fig. 8. Query performance with different datasets: vary k. (a) TW200k. (b) TW2m. (c) GN.

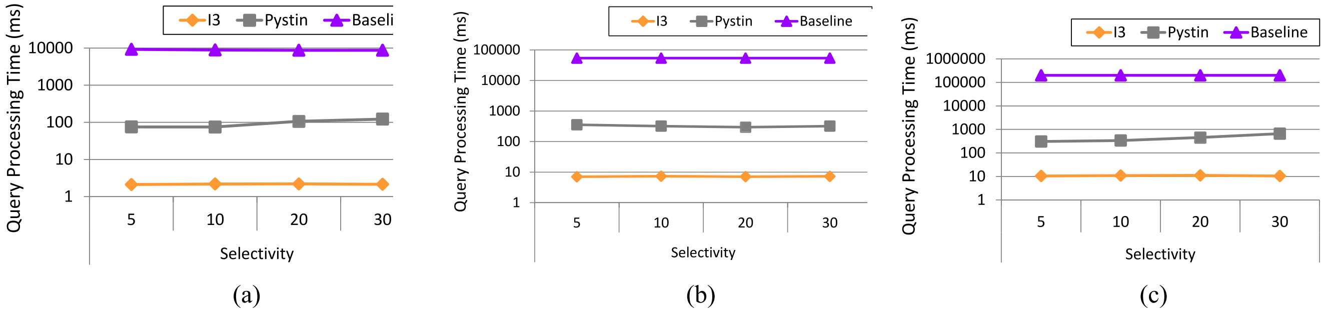


Fig. 9. Query performance with different datasets: vary selectivity. (a) TW200k. (b) TW2m. (c) GN.

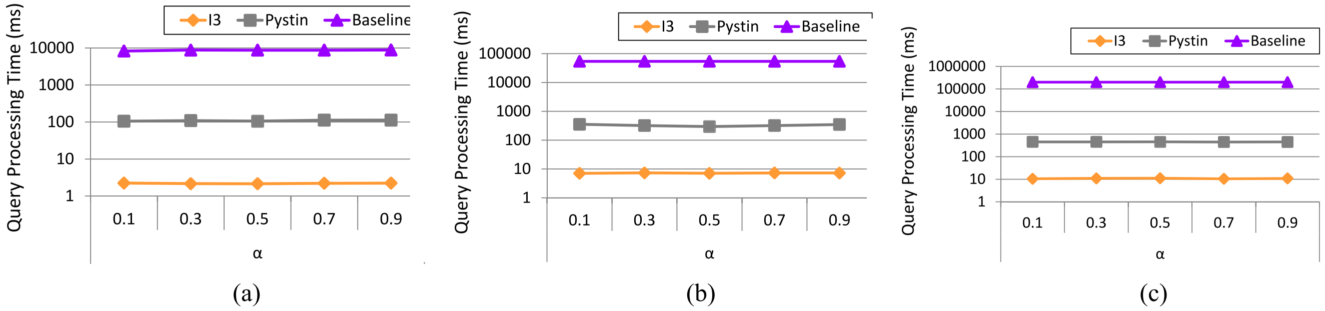


Fig. 10. Query performance with different datasets: vary α. (a) TW200k. (b) TW2m. (c) GN.

A. Spatio-Textual Index

Spatio-textual index is one hybrid indexing, which combines the textual and the spatial indices. Currently, inverted files and R-trees are commonly chosen as the textual and spatial indices, respectively. Cong *et al.* [4] proposed IR-tree, which augments each node of R-tree with a bitmap signature file that represents the textual information in the child nodes. Rocha-Junior *et al.* [18] proposed an index structure, called S2I, which uses a variation of R-tree called aggregate R-tree or a R-tree to map each frequent keyword to a tree. Zhang *et al.* [8] proposed I^3 technique, which combines the inverted files and quad-tree. I^3 stores the information in the hierarchy of keyword cells. In addition, it also stores the summary information of each keyword cell for efficient pruning, which can improve the efficiency and scalability of the index. These index structures are aimed at top-*k* spatial keyword search, but they do not support privacy-preserving query processing. Our index, Pystin, supports privacy-preserving spatio-textual queries and utilizes I^3 as a basis to provide secure query processing over the Cloud.

B. Secure Spatial Query

In a Cloud-based model, it is imperative to have a mechanism for blind processing as the Cloud is semi-trusted and *honest-but-curious*. Gruteser and Grunwald [19] first addressed secure spatial query by introducing the location k-anonymity model, in which an adversary could not identify the user location with a probability of $1/k$. In another effort, Gedik and Liu [20] introduced the concept of trusted third party (TTP) to achieve location cloaking. Khoshgozaran and Shahabi [21] proposed a TTP-based scheme to convert the spatial information of the object and query in a space to a different space. The TTP is responsible for maintaining the relation between the two spaces for accuracy. In a similar attempt, [22] proposed location cloaking using TTP. User location is transformed into an area with at least $k-1$ users. The concept of dummy locations is applied by Kido *et al.* [23] where a user hides her location by introducing many random points in her query. However, the TTP-based schemes, such as [21] and [22], suffer from the fact that TTP houses the sensitive information, which is a huge security risk.

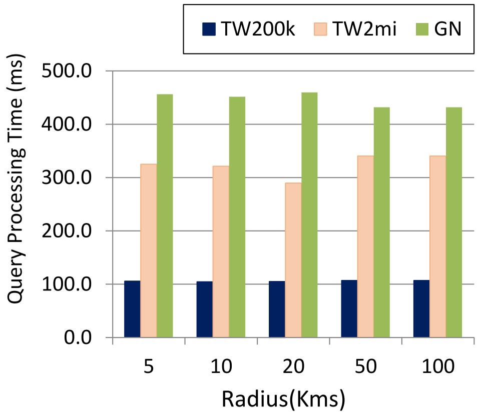


Fig. 11. Query performance of Pystin: vary radius.

Moreover, these schemes focus on location privacy only and are not a best fit for spatio-textual queries on big data that leverage textual relevance greatly. To address the issues in big data query, an efficient mechanism to access, store and manage data is required. For this reason, Hu *et al.* [6] used R-tree-based index. Their approach, called ASM-PH, implements privacy homomorphism to map operations on plaintext to operations on ciphertext and provides a method to support secure spatial queries on Cloud. In another approach, Yiu *et al.* [5] proposed an index traversal technique based on location transformation by using AES. This approach supports the range queries without errors over the transformed location space. Elmehdwi *et al.* [24] used the Paillier cryptosystem to secure query information. However, these schemes are limited by their applications to location privacy only. On the other hand, our approach achieves both location and textual privacy and confidentiality.

C. Secure Textual Query

For secure textual query, searchable encryption (SE) is central to the idea of secure query processing. Curtmola *et al.* [25] gave the formal definition of SE. In this context, the Cloud servers can offer either Boolean or ranked search approaches to process user provided encrypted queries [26]. Furthermore, each of these two approaches could either support single or multikeyword search. A single keyword Boolean search returns documents that contain the given keyword. Boolean operators, such as AND and OR can be used in the case of multikeyword search. Boolean searches look for exact match and do not use a ranking method. Ranked search techniques are able to return documents that are ordered based on a relevance score in relation to the keyword (or keywords) in the user queries. The relevance score can be calculated by extending the searchable index to utilize a keyword ranking functions, such as TF-IDF [10], cosine similarity, or the language model [27].

Inverted list is a popular technique for searching textual data. Curtmola *et al.* [25] proposed a textual index based on the inverted list. Wang *et al.* [28] addressed the problem of order preserving encryption to rank the objects in a textual index. Boneh and Waters [29] proposed for the first time asymmetric encryption-based SE. In another work [30], moving from single keyword search to multikeyword search, Cao *et al.* employed symmetric encryption for multiword ranked search scheme. Sun *et al.* [31] proposed an efficient ranked privacy preserving keyword search using cosine similarity. However, these schemes secure the textual queries only and fail to secure a spatio-textual query. In contrast, our index is able to secure spatio-textual queries.

D. Secure Spatio-Textual Query

Su *et al.* [17] proposed an IR-tree-based [4] index scheme, called P*k*SKQ, which is focused on privacy preserving top-*k* query processing over the Cloud. Specifically, the scheme combines the spatial and textual information into a single vector to provide a unifying approach. The vector is then secured against the chosen-plaintext and known-plaintext attacks by using ASPEN encryption. To search with the secure index, P*k*SKQ employs two techniques: 1) anchor-based position determination and 2) position distinguished trap door generation, which allows for the similarity computations between the query and the documents without divulging any information.

Different from the above, our proposed Pystin scheme is based on I^3 index, which is significantly faster than IR-tree, as was demonstrated in [8]. Pystin maintains summary information for efficient secure pruning, where space is pruned based on the relevance score as well as the signature file. Pystin supports Ranked multikeyword search over encrypted spatio-textual data, and achieves most security requirements identified in [26]. We have demonstrated the efficiency and scalability of Pystin through extensive experimentation.

VIII. Conclusion

In this paper, we have proposed a new privacy-preserving top-*k* spatio-textual query scheme, called Pystin, to enable secure LBS in smart cities. The prposed Pystin is performed over outsourced Cloud, which combines BGN homomorphic encryption and hash bucket techniques to allow a registered query user to obtain PT*k*SK query results, without divulging the accurate location information. In addition, the privacy of textual information is also persevered by a one-way hash function. In order to further reduce the query latency, an efficient quad-tree-based spatio-textual indexing is integrated into Pystin. Detailed security analyzes show that Pystin is indeed a PT*k*SK query scheme. Furthermore, extensive experiments are conducted to confirm the scalability, efficiency properties of Pystin. In future work, we will exploit security and efficiency issues of other LBS queries in smart cities.

References

[1] (2017). *Stats*. [Online]. Available: https://newsroom.fb.com/company-info/

[2] (2017). *Twitter Users*. [Online]. Available: https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

[3] M. Barbard and T. Zeller, *A Face Is Exposed for AOL Searcher No. 4417749*, New York Times, New York, NY, USA, Aug. 2006.

[4] 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, Aug. 2009.

[5] M. L. Yiu, G. Ghinita, C. S. Jensen, and P. Kalnis, "Outsourcing search services on private spatial data," in *Proc. ICDE*, 2009, pp. 1140–1143.

[6] H. Hu, J. Xu, C. Ren, and B. Choi, "Processing private queries over untrusted data cloud through privacy homomorphism," in *Proc. ICDE*, 2011, pp. 601–612.

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

[8] D. Zhang, K.-L. Tan, and A. K. H. Tung, "Scalable top-K spatial keyword search," in *Proc. EDBT*, 2013, pp. 359–370.

[9] 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.

[10] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manag.*, vol. 24, no. 5, pp. 513–523, 1988.

[11] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[12] (2018). *TF-IDF*. [Online]. Available: https://en.wikipedia.org/wiki/Tf-idf

[13] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1997.

[14] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *Proc. VLDB Endowment*, vol. 6, no. 3, pp. 217–228, 2013.

[15] (2018). *DIMACS Implementation Challenge—Challenge Benchmarks*. [Online]. Available: http://www.dis.uniroma1.it/challenge9/download.shtml

[16] (2018). *United States Board on Geographic Names*. [Online]. Available: https://geonames.usgs.gov/domestic/download_data.htm

[17] S. Su *et al.*, "Privacy-preserving top-K spatial keyword queries in untrusted cloud environments," *IEEE Trans. Services Comput.*, vol. 11, no. 5, pp. 796–809, Sep./Oct. 2015.

[18] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-K spatial keyword queries," in *Proc. Int. Symp. Spat. Temporal Databases*, 2011, pp. 205–222.

[19] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *Proc. ACM 1st Int. Conf. Mobile Syst. Appl. Services*, 2003, pp. 31–42.

[20] B. Gedik and L. Liu, "Location privacy in mobile systems: A personalized anonymization model," in *Proc. ICDCS*, 2005, pp. 620–629.

[21] A. Khoshgozaran and C. Shahabi, "Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy," in *Proc. SSTD*, 2007, pp. 239–257.

[22] C.-Y. Chow, M. F. Mokbel, and W. G. Aref, "Casper*: Query processing for location services without compromising privacy," *ACM Trans. Database Syst.*, vol. 34, no. 4, p. 24, 2009.

[23] H. Kido, Y. Yanagisawa, and T. Satoh, "Protection of location privacy using dummies for location-based services," in *Proc. ICDE Workshops*, 2005, p. 1248.

[24] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environments," in *Proc. ICDE*, 2014, pp. 664–675.

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

[26] D. V. N. S. Kumar and P. S. Thilagam, "Approaches and challenges of privacy preserving search over encrypted data," *Inf. Syst.*, vol. 81, pp. 63–81, Mar. 2018.

[27] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *Proc. SIGIR*, 1998, pp. 275–281.

[28] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proc. ICDCS*, 2010, pp. 253–262.

[29] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. TCC*, Amsterdam, The Netherlands, 2007, pp. 535–554.

[30] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222–233, Jan. 2014.

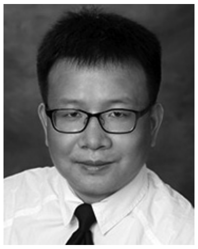
[31] W. Sun *et al.*, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proc. ACM SIGSAC*, 2013, pp. 71–82.

Divya Negi received the master's degree in computer science from the Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada.

After graduating from the University of New Brunswick, she joined Morgan Stanley Canada Limited, Montreal, QC, Canada, as a Software Engineer. Her current research interests include big data systems, query processing and security, and privacy issues in big data.

Suprio Ray (M'17) received the Ph.D. degree from the Department of Computer Science, University of Toronto, Toronto, ON, Canada.

He is an Assistant Professor with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada. His current research interests include big data and database management systems, run-time systems for scalable data science, provenance and privacy issues in big data, and data management for the Internet of Things.



Rongxing Lu (S'09–M'10–SM'15) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada.

He has been an Assistant Professor with the Faculty of Computer Science, University of New Brunswick, NB, Canada, since 2016. He was an Assistant Professor with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, from 2013 to 2016. His current research interests include applied cryptography, privacy enhancing technologies, and Internet of Things big data security and privacy.

Dr. Lu was a recipient of the Governor Generals Gold Medal and the IEEE Communications Society (ComSoc) Asia–Pacific Outstanding Young Researcher Award. He currently serves as the Secretary of the IEEE ComSocCIS-TC.