LS-RQ: A Lightweight and Forward-Secure Range Query on Geographically Encrypted Data

Yanguo Peng, *Member, IEEE*, Long Wang, Jiangtao Cui, *Member, IEEE*, Ximeng Liu, *Member, IEEE*, Hui Li, *Member, IEEE*, and Jianfeng Ma, *Member, IEEE*

Abstract—In the era of cloud computing, to achieve convenient location-based service (LBS), consumers such as users, companies, and organizations prefer subcontracting massive geographical data to public clouds after encryption for privacy and security. However, numerous harmful cyber-attacks happen on those public clouds in an unpredicted and hourly manner. To alleviate those concerns, various secure query schemes on the encrypted data have been proposed in the literature. As a fundamental query of LBSs, forward-secure range query has not been well investigated. To address this issue, we propose a lightweight and forward-secure range query (LS-RQ) on geographically encrypted data, which soundly balances between security and efficiency. Promisingly, we design an index mechanism to manage geographical data on the public clouds, while not compromising the privacy of data. Moreover, our LS-RQ schemes provide a convenient approach to range query on geographically encrypted data on-the-fly. We also rigorously prove that LS-RQ is forward-secure. Finally, extensive experimental studies are performed on both real and synthetic datasets. By observation, our LS-RQ schemes are highly efficient in realistic environments. Particularly, on encrypted datasets with about 1 000 000 geographical data, our solution to secure range query takes strictly less than a second.

Index Terms—Dynamic range query, locality sensitive hashing, proxy re-encryption, forward-security, location-based service

1 Introduction

In the era of cloud computing, various types of data from both academic and industrial communities are constantly subcontracted to public clouds by consumers (i.e., users, companies, organizations, etc.). Within location-based service (LBS) applications, a representative field among these, geographically tagged data is subcontracted to public clouds for data-driven services, such as query, mining, analysis, etc. With the help of public clouds, any company can provide advertising recommendations and spatial-temporal correlated services on-the-fly without worrying about hardware investment and maintenance overhead. Notably, Twitter[1] and Netflix[2] have outsourced all trajectory data and messages (i.e., photos, videos, and texts) with geographical tags to public clouds.

Unfortunately, unpredictably and hourly harmful cyber attacks by hacker groups have threatened privacy and security of data terribly in public clouds, such as Equifax data breach,[3] Verizon cloud leak,[4] etc. Those hacker groups leverage on the leaked trajectory data to infer consumers' private information (i.e., home address, private property, etc.), which are further sold for benefits. An ideal solution towards this problem is the encryption-before-outsourcing mechanism, which avoids the potential leakage of private information. However, as traditional symmetrical encryptions completely break the semantic of original data and thus degenerate the availability of data, the LBS cannot work properly with such encryptions.

To overcome the problem, extensive research efforts have been conducted with respect to secure LBS models, especially the fundamental building block, namely secure range query over geographical data with 2-dimensionality [1], [2], [3], [4], [5], [6], [7]. Compared with 1-dimensional data (i.e., keyword [8], [9], integer/float values [10], etc.), geographical data is inherently out-of-order, and thus the encrypted one is harder to be securely indexed. Hence, a secure range query on geographically encrypted data is more challenging. Definitely, a secure range query on geographical data can efficiently retrieve points of interest that are bounded in a certain 2-dimensional range without decrypting the data on-the-fly.

However, in the above schemes a token, which is submitted to the public cloud for secure range query on-the-fly, will inherently be long-term effective. Since the public cloud can easily capture a legal token, it means that the public cloud can reconstruct all original data by leveraging the attacks that are defined in literatures [11], [12], [13], [14]. Similarly, an adversary, who secretly captures a legal token, can ubiquitously do the same attack until consumers are aware of the leakage of the token. Fortunately, forward-security [15], [16] is a promising security model against such attacks. In a secure range query with forward-security, a legal query token generated with a particular timestamp is only allowed to issue queries on existing records but will be invalid on those records updated afterwards.

1. https://blog.twitter.com/engineering/en_us/topics/infrastructure/2018/a-new-collaboration-with-google-cloud.html
2. https://media.netflix.com/en/company-blog/completing-the-netflix-cloud-migration
3. https://www.ftc.gov/equifax-data-breach
4. https://money.cnn.com/2017/07/12/technology/verizon-data-leaked-online/index.html

- *Y. Peng, L. Wang, and J. Cui are with the School of Computer Science and Technology, Xidian University, Xi'an 710071, China. E-mail: {ygpeng, cuijt}@xidian.edu.cn, wldklyx@gmail.com.*
- *X. Liu is with the College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China. E-mail: snbnix@gmail.com.*
- *H. Li and J. Ma are with the School of Cyber Engineering, Xidian University, Xi'an 710071, China. E-mail: {hli, jfma}@xidian.edu.cn.*

Manuscript received 11 July 2019; revised 18 Dec. 2019; accepted 11 Feb. 2020. Date of publication 17 Feb. 2020; date of current version 17 Jan. 2022. (Corresponding author: Ximeng Liu.)
Digital Object Identifier no. 10.1109/TDSC.2020.2974218

TABLE 1
Summary of Primary Notations

Notation	Meaning
$\mathbb{D}$, $\boldsymbol{p}$	A set of points, and a point.
$\mathbb{B}$, B	A bucket, and a bucket ID.
Q	A query range.
τ	A token for query, insertion, updating or deletion.
$\#(X)$	Number of elements in X.
ID	An identifier of a point.
$(ID_i, \boldsymbol{p}_i)$	An ID-point pair, in which ID_i is the identifier of $\boldsymbol{p}_i$.
$\mathtt{Dic}$, $\varphi\mathtt{Dic}$	Single and forwardly inverted dictionary.
$\mathtt{h}$, $\mathtt{h}(\boldsymbol{p})$	A locality sensitive hashing (LSH) function.
H, $H(\cdot)$	A keyed hash function.
$\mathtt{PRE}$	A concrete proxy re-encryption algorithm.
$< pk, sk >$	A pair of public and secret keys.
$rk_{A\rightarrow B}$	A re-encryption key.
$\mathcal{C}$, $\mathcal{A}$	Challenger and adversary.
$Game_{\mathcal{R},\mathcal{A}(1^\lambda)}$	A real game between $\mathcal{C}$ and $\mathcal{A}$.
$Game_{\mathcal{S},\mathcal{A}(1^\lambda)}$	A simulated game between $\mathcal{C}$ and $\mathcal{A}$.
$\mathcal{L}$	A leakage function.
$\tilde{\mathtt{h}}$	A set that contains all points with $\mathtt{h}$ in dataset.
Δ	A label for fast locating candidate points.
$\|\|\boldsymbol{\alpha}\|\|_2$	The ℓ_2 norm of $\boldsymbol{\alpha}$.
$\&(X)$	The address of X.
m	The number of the adopting LSH functions.
ℓ	The LSH value length for a single LSH function.

For example, Twitter leverages on the result of remote secure range query to mine hot spots in a city on-the-fly. If a legal query token is secretly captured by an adversary (even a public cloud), who can ubiquitously issue legal requests of the same range query, he/she is able to re-construct the whole dataset and further infer corresponding users' private information until Twitter is aware of the leakage of the token. In contrast, in a forward-secure range query, even if a previous legal token is leaked, an adversary cannot execute the range query on the newly updated records. This scheme limits the leakage risk of both data and query. Hence, forward-security is especially practical for dynamic application scenarios, in which the geographical tags will be frequently updated due to address (i.e., geographical coordinates) migration and facility addition.

In particular, cloud computing aims at providing a lightweight framework for consumers, where most computing and storage burdens are shifted to public clouds. Following this principle, we propose a **L**ightweight and forward-**S**ecure **R**ange **Q**uery (LS-RQ) scheme on geographically encrypted data, and demonstrate that it has a good balance between security and efficiency. The main contributions are threefold,

1) *Secure data storage:* LS-RQ allows users to outsource geographical datasets to public clouds for secure storage without compromising the privacy of each record.
2) *Secure range query on-the-fly:* LS-RQ enables fast range query directly on geographically encrypted data without decrypting on public clouds. Moreover, it prevents the potential leakages of both data and query.
3) *Forward-security and high efficiency:* We present a rigorous proof of LS-RQ's forward-security. Moreover, extensive experimental studies are carried out on both real and synthetic datasets to demonstrate the superiority of LS-RQ.

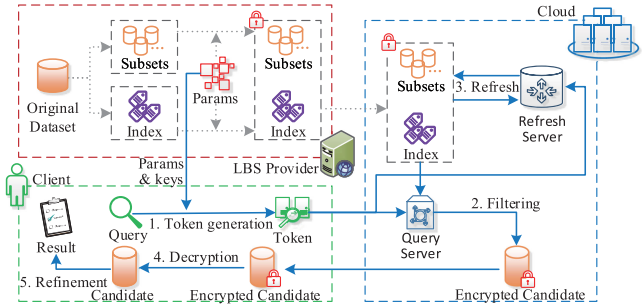


Fig. 1. A forward-secure framework for range query in LBS. (The dotted lines with arrows exhibit the off-line preprocessing. The solid numbered lines with arrows exhibit the on-line processing sequentially.)

The rest of this paper is organized as follows. In Section 2, we present the framework of LS-RQ and the formal model of security, followed by the generic idea to achieve LS-RQ. In Section 3, a new index mechanism for LS-RQ is designed based on several newly designed kernel components. Then, we present the formal construction of LS-RQ in Section 4 and theoretically analyze it in Section 5. In Section 6, experimental studies are elaborated and analyzed in detail. Finally, we summarize related works in Section 7 and conclude this paper in Section 8. Primary notations are summarized in Table 1.

2 Problem Formalization

In this part, we present the system framework and formalize the security model for LS-RQ. Afterwards, several generic building blocks for designing a secure range query framework are introduced. Finally, we give a brief introduction of proxy re-encryption (PRE), which will be employed in the construction of LS-RQ.

2.1 System Framework

A dynamic range query framework on geographically encrypted data consists of three entities (namely, a client, a public cloud, and an LBS provider) in Fig. 1. During the off-line phase, the LBS provider divides the original dataset into subsets and generates an index. It selects a set of security-related parameters and encrypts both them afterwards. Following that, the encrypted ones are subcontracted to the public cloud. During the on-line phase, given a range query, the client generates a token and submits it to the public cloud with the help of the LBS provider. The public cloud will perform a filtering algorithm to obtain encrypted candidates and re-encrypt the accessed encrypted points and index. Then, the public cloud sends the encrypted candidates back to the client, who will further decrypt and refine the partial answer to obtain the final result.

2.2 Security Model

Forward-secure range query is a promising branch of dynamic searchable symmetric encryption. Note that, public cloud is assumed to be semi-honest, which means that it strictly follows the protocol but is curious about the encrypted data and index. Client and LBS provider are both assumed to be honest. The forward-security model for LS-RQ is parameterized by a

collection of leakage functions

$$\mathcal{L} = (\mathcal{L}_S, \mathcal{L}_Q, \mathcal{L}_I, \mathcal{L}_D, \mathcal{L}_U).$$

Here, $\mathcal{L}_S$, $\mathcal{L}_Q$, $\mathcal{L}_I$, $\mathcal{L}_D$ and $\mathcal{L}_U$ are the leakage functions during system setup, range query, data insertion, deletion and updating, respectively. In fact, the first 4 primitives are consistent with those in existing forward-security models [15], [16], [17], [18], [19], [20], [21].

More precisely, a real game $Game_{\mathcal{R},\mathcal{A}(1^\lambda)}$ and a simulated (ideal) game $Game_{\mathcal{S},\mathcal{A}(1^\lambda)}$ are defined. We carefully define a sequence of games from the real game $Game_{\mathcal{R},\mathcal{A}(1^\lambda)}$ to approximate the simulated game $Game_{\mathcal{S},\mathcal{A}(1^\lambda)}$. The task of adversary $\mathcal{A}$ is to distinguish between every two contiguous games. Following on the definitions of leakage functions and games, $\mathcal{L}$-forward-adaptive-security for LS-RQ can be formalized as follows.

Definition 1 ($\mathcal{L}$-forward-adaptive-security for LS-RQ). *An LS-RQ scheme is $\mathcal{L}$-forward-adaptive-secure if two constraint conditions are satisfied.*

1) For any adversary $\mathcal{A}$ in probabilistic polynomial-time, there exists an efficient simulator $\mathcal{S}$ such that the following equation holds:

$$|\Pr[Game_{\mathcal{R},\mathcal{A}(1^\lambda)} = 1] - \Pr[Game_{\mathcal{S},\mathcal{A}(1^\lambda)} = 1]| \leq negl(1^\lambda).$$

Herein $negl(1^\lambda)$ is a negligible function bounded by security parameter 1^λ. Both the real and simulated games are defined as follows,

- $Game_{\mathcal{R},\mathcal{A}(1^\lambda)}$. *$\mathcal{A}$ initially chooses a dataset $\mathbb{D}$, and gets back the output of standard* Setup. *Then, $\mathcal{A}$ adaptively performs* Insertion, Query, Updating *and* Deletion *and gets the real transcripts generated by these primitives. Finally, $\mathcal{A}$ observes the real transcripts and outputs a bit $b \in \{0, 1\}$.*
- $Game_{\mathcal{S},\mathcal{A}(1^\lambda)}$. *$\mathcal{A}$ initially chooses a dataset $\mathbb{D}$, and gets back the output of standard $\mathcal{S}(\mathcal{L}_S(1^\lambda))$. Then, $\mathcal{A}$ adaptively performs $\mathcal{S}(\mathcal{L}_I(1^\lambda))$, $\mathcal{S}(\mathcal{L}_Q(1^\lambda))$, $\mathcal{S}(\mathcal{L}_U(1^\lambda))$ and $\mathcal{S}(\mathcal{L}_D(1^\lambda))$, and gets the ideal transcripts generated by these primitives. Finally, $\mathcal{A}$ observes the ideal transcripts and outputs a bit $b \in \{0, 1\}$.*

Additionally, a game returning 1 means that the adversary accepts all transcripts and outputs all primitives. Otherwise, a game returns 0.

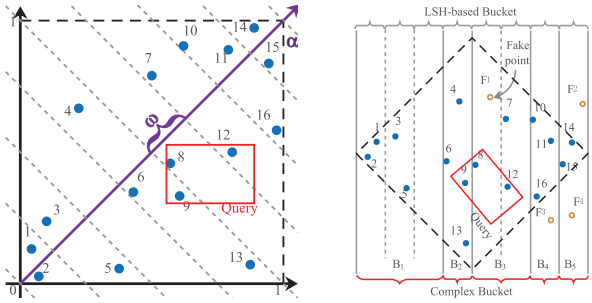
2) There exists a leakage function $\bar{\mathcal{L}}$ such that,

$$\mathcal{L}_I(\{(ID_i, \boldsymbol{p}_i)\}_{i=1}^n) = \bar{\mathcal{L}}(\{ID_i\}_{i=1}^n).$$

Herein $\{(ID_i, \boldsymbol{q}_i)\}_{i=1}^n$ is the inserted dataset, and $\{ID_i\}_{i=1}^n$ denotes the corresponding set of IDs. $(ID_i, \boldsymbol{p}_i)$ is an ID-point pair, where ID_i is the identifier of $\boldsymbol{p}_i$. It means that the insertion primitive of any $\mathcal{L}$-forward-adaptive-secure LS-RQ scheme reveals only the identifiers of the inserted points but nothing else.

2.3 System Overview

Efficiency and security are the paramount concerns in range query on a public cloud. LS-RQ is systematically reviewed from the following aspects.



(a) Locality sensitive hashing (b) Buckets

Fig. 2. A generic idea for fast range query on geometric data. (Generally, a range is mapped into a set of complex buckets which completely covers the range itself. The given range contains $\boldsymbol{p}_8$, $\boldsymbol{p}_9$ and $\boldsymbol{p}_{12}$. For avoiding a cloud knowing the range itself, both B_2 and B_3 are sent to the cloud from the client. The cloud returns all real points (i.e., $\boldsymbol{p}_4$, $\boldsymbol{p}_6$, $\boldsymbol{p}_7$, $\boldsymbol{p}_8$, $\boldsymbol{p}_9$, $\boldsymbol{p}_{12}$, $\boldsymbol{p}_{13}$) and fake ones (i.e., F_1) are sent back to the client, who refines it with the real range and obtains the accurate results in plain domain.)

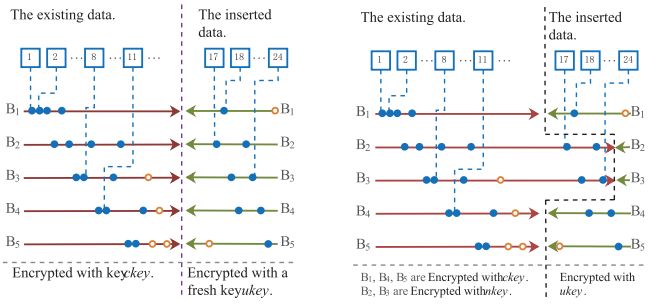
Dataset Division. Index, which is a dominant component for fast locating candidate points, can only be efficiently constructed on ordered data. However, geographically data is out-of-order from a cursory observation and is difficult to be indexed. Additionally, if all subsets contain different numbers of points, an adversary can easily distinguish between any two encrypted subsets. Hence, to provide indistinguishability between encrypted subsets, all subsets should be of regularity, which means all subsets contain a consistent number of points.

To resolve the first concern, LSH is revised (formally defined in Section 3.1) to be compatible with geographic data and employed in a novel manner to construct complex buckets (i.e., the outsourcing subsets). LSH [22] is a widely-adopted dimension reduction tool that can map high-dimensional data into a single ordered code (i.e., integer) by equation $\mathtt{h}(\boldsymbol{p}) = \lfloor \frac{\boldsymbol{\alpha} \cdot \boldsymbol{p} + r}{w} \rfloor$ as shown in Fig. 2a.[5] Furthermore, in order to retrieve candidate points efficiently, aggregating close points into a unique bucket is widely adopted in various applications [23], [24], [25]. Once a query is issued, the client can quickly locate the bucket, in which the points are near to the query one. In LS-RQ as shown in Fig. 2b, all points in each LSH-based bucket corresponds to a unique LSH value by $B_i = \{\boldsymbol{p}_j | \mathtt{h}(\boldsymbol{p}_j) = i \wedge \boldsymbol{p}_j \in \mathbb{D}\}$. Following that, to eliminate small-scale buckets, we reasonably select a volume n_V (i.e., 4 in Fig. 2b) to construct complex buckets, by adopting new-designed greedy merge method (formally defined in Section 3.2). All LSH-based buckets are merged into several complex buckets such that the size for each complex bucket is closest to but does not exceed n_V as shown in Fig. 2b.

Additionally, to eliminate the irregularity of complex buckets, fake points are added such that the size of each complex bucket equals to n_V. Specifically, a single fake point is added into B_3 and B_4 respectively, and two fake points are added into B_5. All fake points are hollow and can be easily distinguished from the true ones in solid.

Index and Query. Note that, both rectangle-range and circular-range queries are compatible with the mechanism to

5. Here, $\boldsymbol{\alpha}$ is a random vector (i.e., point), r is a segment offset, and w is a unique width of segments.



(a) Before query. (b) After query.

Fig. 3. A generic idea for forward-security in range query. (Assume that the query result falls in complex buckets B_2 and B_3.)

be described below, hereby we adopt rectangle-range query to illustrate the general idea as shown in Fig. 2. Specifically, a rectangle-range query is mapped into a set of identifiers of complex buckets, such that the complex buckets' corresponding spatial region can completely cover the region of range query. Specifically, in a range query as illustrated in Fig. 3, by employing a new-designed forwardly inverted dictionary, both complex buckets and deleting points can be quickly located. Specifically, in the forwardly inverted dictionary (formally defined in Section 3.3), all complex buckets are inversely indexed for fast range query and all points are forwardly indexed for quickly deleting points.

Moreover, for forward-security, the encrypted dataset stored on the cloud is divided into two parts. Points in the first and second parts are encrypted with $ckey$ and $ukey$, respectively. When issuing a query, a token for retrieving all points in B_2 and B_3 is submitted to the cloud, who returns all encrypted points in B_2 and B_3 as candidate points. After performing the query, the points that have been visited (i.e., all points in B_2 and B_3) are re-encrypted with $nkey$. In this way, forward-security is satisfied, which will be analyzed in Section 5.2. The formal construction of LS-RQ is in Section 4.

2.4 Proxy Re-Encryption Algorithm

Blaze *et al.* first introduced the concept of PRE [26]. Since then, PRE has undergone extensive research [27], [28], and has been deployed into various practical applications [8], [29]. We introduce PRE into LS-RQ to re-encrypt the visited points on public clouds while answering a range query. A concrete proxy re-encryption is a tuple of 6 primitives, $\texttt{Setup}(1^\lambda) \rightarrow prm$, $\texttt{KeyGen}() \rightarrow (< pk_A, sk_A >)$, $\texttt{ReKeyGen}(sk_A, sk_B) \rightarrow rk_{A\rightarrow B}$, $\texttt{Enc}(pk_A, m) \rightarrow c_A$, $\texttt{ReEnc}(rk_{A\rightarrow B}, c_A) \rightarrow c_B$ and $\texttt{Dec}(sk_A, c_A) \rightarrow m$. Note that prm is necessary for the last five primitives but is omitted in the rest of this paper, sk_A and pk_A are secret and public keys, $rk_{A\rightarrow B}$ is a re-encryption key, $m \in \mathcal{R}_{NTRU}/q$ is a plaintext and $c \in \mathcal{R}_{NTRU}/q$ is a ciphertext. Additionally, PRE satisfies the following properties.

- *Correctness*. First, given sk_A and $c_A = \texttt{Enc}(pk_A, m)$, $\texttt{Dec}(sk_A, c_A) = m$ holds. Second, given sk_B and $c_B = \texttt{ReEnc}(rk_{A\rightarrow B}, c_A)$, $\texttt{Dec}(sk_B, c_B) = m$ holds.
- *Transitivity*. If $rk_{A\rightarrow B} = sk_A^{-1} \cdot sk_B$ and $rk_{B\rightarrow C} = sk_B^{-1} \cdot sk_C$, then $rk_{A\rightarrow C} = rk_{A\rightarrow B} \cdot rk_{B\rightarrow C}$.

In fact, we only list part of the properties that PRE supports. The other properties (such as directionality, interactivity, etc.), which are not necessary for our work, are omitted here. Also, the detailed definition of PRE is referred to in the literature [27].

3 Index Mechanism for LS-RQ

In this section, we design a new-defined even LSH function and a new-defined greedy merge method to generate complex buckets with regularity. Following that, to speed up the process of query, a forwardly inverted dictionary is designed to manage geographical data.

3.1 Even LSH Function

In order to generate regularity complex buckets, even LSH function is carefully designed by leveraging normalization and density distribution of a dataset.

Theorem 1 (Even Division). *Give a code-length $\ell \in \mathbb{N}^*$, a random vector $\boldsymbol{\alpha} = (\alpha_x, \alpha_y) \in \mathbb{R}^d \times \mathbb{R}^d$ such that $||\boldsymbol{\alpha}||_2 = 1$, and a density function $f(x, y)$ that dataset $\mathbb{D}$ follows. The critical value $\{c_i | 1 \le i \le 2^\ell\}$ (as shown in Fig. 10 in supplementary 1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2020.2974218) are calculated by solving the following equation, such that each area contains consistent number of points.*

$$\begin{aligned} A \cdot i/2^\ell = {} & \Phi_1(c_{ix}, u(c_{ix})) - \Phi_1(0, u(0)) \\ & - \Phi_2(c_{ix}, v(c_{ix})) + \Phi_2(0, v(0)). \end{aligned} \tag{1}$$

Here, $u(x)$ and $v(x)$ are the upper and lower bound of integral, $A =_{\mathbb{D}} f(x, y)d_\sigma$, $\Phi_1 = \int F(x, u(x))dx$, $\Phi_2 = \int F(x, v(x))dx$ and $F(x, y) = \int f(x\alpha_x + y\alpha_y, y\alpha_x - x\alpha_y)dx$.

(The proof is in supplementary 1, available online.)

Definition 2. *An even LSH function $\mathtt{h} : \mathbb{R}^d \rightarrow \{0, 1\}^\ell$ maps a d-dimensional object into a single integer. An even LSH function is piecewise defined as*

$$\mathtt{h}(\boldsymbol{p}) = \{i | c_{i-1} < \boldsymbol{p} \cdot \boldsymbol{\alpha} \le c_i\}.$$

Here, $\boldsymbol{\alpha}$ is a random vector such that $||\boldsymbol{\alpha}||_2 = 1$, $c_0 = 0$, and $\{c_i | 1 \le i \le 2^\ell\}$ are determined by Theorem 1.

By Definition 2, all points in a dataset are divided into complex buckets such that each one contains an almost unique number of points.

3.2 Greedy Merge Method

By only introducing even LSH function, it is not sufficient to guarantee the regularity for all complex buckets due to the following reasons: 1) the distribution of data is not known previously; or 2) even if the distribution is known, all complex buckets are not regular with a non-negligible probability since the loss of precision during the construction. In order to make the numbers of points in complex buckets equal, a greedy merge method is proposed to resolve the first problem. Besides, fake addition is additionally introduced to resolve the second problem. Here, we define $\boldsymbol{p}_i \sim \boldsymbol{p}_j$ iff $\mathtt{h}_i = \mathtt{h}_j$, in which $\mathtt{h}_i$ is the LSH value of $\boldsymbol{p}_i$. Specially, we denote $\tilde{\mathtt{h}}_i = \{\boldsymbol{p}_j | \boldsymbol{p}_i \sim \boldsymbol{p}_j\}$.

For Unknown Distribution of Dataset. Equal-length LSH function defined in [30] is employed to map original points to LSH-based codes. LSH-mapping dictionary $\texttt{Dic}_\mathtt{H}$, which is generated by a greedy merge method, maintains a set $\{(\mathtt{h}_i, B_j) | \tilde{\mathtt{h}}_i \subseteq \mathbb{B}_j\}$ that maps LSH values into complex buckets.

The greedy merge method is formally described in Algorithm 1. $\mathbb{D}$ is sorted at the beginning and $\mathtt{Dic_H}$ is initialized to be empty (Line 2-3). Maximum bucket size w is the greater of a threshold θ and the maximum volume of buckets (Line 4). A predefined threshold can avoid obtaining a locally optimal solution, which means that w is too small to result in an excessive number of complex buckets. The merge starts from the first element in a dataset (Line 5). Then, a loop is carried out to continuously construct complex buckets (Lines 6-19). Also, an LSH-mapping dictionary is generated. Following that, fake addition (shown in Algorithm 2) is applied to guarantee the regularity for all complex buckets (Line 20). Finally, the regular complex buckets $\mathbb{B}$ and LSH-mapping dictionary $\mathtt{Dic_H}$ are returned (Line 21).

Algorithm 1. Greedy Merge Method

1: **function** GREEDYMERGE ($\mathbb{D}$, θ)
2: Sort $\mathbb{D}$ according to their LSH values;
3: Let $\mathtt{Dic_H} \leftarrow \emptyset$, $i \leftarrow 1$, $j \leftarrow 1$; /* i and j are subscripts of complex buckets and points, respectively. */
4: $w \leftarrow \max\{\theta, \max_{1\le k\le n} \#(\tilde{\mathtt{h}}_k)\}$;
5: **repeat**
6: $\mathbb{B}_i \leftarrow \emptyset$;
7: **repeat**
8: **if** $\#(\mathbb{B}_i)$ is greater than w **then**
9: $\mathtt{Dic_H.Ins}(\mathtt{h}, B_i)$, where $\mathtt{h} \in [\mathtt{h}_{j-1}, \mathtt{h}_j - 1]$ and B_i is the ID of complex bucket $\mathbb{B}_i$. The loop is terminated;
10: **else**
11: Add all points in $\tilde{\mathtt{h}}_j$ into $\mathbb{B}_i$, and $(\mathtt{h}_j, B_i)$ into $\mathtt{Dic_H}$;
12: Increase j by $\#(\tilde{\mathtt{h}}_j)$;
13: **end if**
14: **until** no left points that are not inserted;
15: Increase i by 1;
16: **until** $j > n$;
17: FAKEADDITION ($\mathbb{B}$) in Algorithm 2;
18: **return** $\mathbb{B}$, $\mathtt{Dic_H}$;
19: **end function**

For Known Distribution of Dataset. By adopting even LSH function, original points are mapped into different LSH values. All points that correspond to an LSH value constitutes a complex bucket. That means $\mathbb{B}_i = \tilde{\mathtt{h}_i}$. However, all complex buckets still have different numbers of points since the possible loss of precision during construction. Here, we first adopt Algorithm 1 to merge buckets. Then, a fake addition algorithm, as shown in Algorithm 2, is applied to guarantee the regularity for all complex buckets. In the algorithm, the fake points added can be attached with a special label in practice for easy recognition. Additionally, according to the proposed even LSH function, LSH-mapping dictionary $\mathtt{Dic_H}$ maintains a set $\{(\mathtt{h}_i, B_j) | \tilde{\mathtt{h}_i} \in \mathbb{B}_j\}$.

3.3 Forwardly Inverted Dictionary

To accelerate the insertion/deletion of points into/from a dataset, a forwardly inverted dictionary is introduced [16] to index all points. However, as [16] is based on the assumption that the Cloud is completely trusted, it cannot be applied in our scenario. Therefore, we present a novel forwardly inverted dictionary scheme. Different from [16], which requires either keyword or document counting, our scheme is based on a single dictionary without counting operation.

Algorithm 2. Fake Addition Algorithm

1: **function** FAKEADDITION ($\mathbb{B}$)
2: Pick up the maximal size $\#_{\max}$ of irregular complex buckets as the unique size for regular complex buckets;
3: **for** $i = 1$ to $\#(\mathbb{B})$ **do**
4: Add $\#_{\max} - \#(\mathbb{B}_i)$ fake points into $\mathbb{B}_i$;
5: **end for**
6: **return** Regular complex buckets $\mathbb{B}$;
7: **end function**

Definition 3 (Single Dictionary). *A single dictionary* $\mathtt{Dic}$ *is a data structure that maintains a set of* $(\Delta, data) \in \{0,1\}^* \times \{0,1\}^*$ *pairs. It allows the following operations (i.e., create, insert, get, update and delete respectively) to manage tuples in the set.*

- $\mathtt{Dic.Crt}(\{(\Delta_i, data_i) | 1 \le i \le n\}) \rightarrow \mathtt{Dic}$;
- $\mathtt{Dic.Ins}(\Delta, data) \rightarrow \mathtt{Dic}'$;
- $\mathtt{Dic.Get}(\Delta_i) \rightarrow data_i$ *or* $\perp$;
- $\mathtt{Dic.Upd}(\Delta, data) \rightarrow \mathtt{Dic}'$;
- $\mathtt{Dic.Del}(\Delta) \rightarrow \mathtt{Dic}'$ *or* $\perp$;

Here, Δ is the unique access entry for each tuple in $\mathtt{Dic}$. The access entry can be efficiently located by adopting perfect hash function [31], which has been widely adopted in almost all mainstream programming languages (such as C/C++, Java, Python, etc.), to implement creation, insertion, getting, updating and deletion.

Definition 4 (Forwardly Inverted Dictionary). *A forwardly inverted dictionary* $\varphi\mathtt{Dic}$ *is a dual dictionary that maintains a set* $\{e_i = (\Delta_i^{(B)}, \Delta_i^{(ID)}, data_i)\}$. *It allows the following operations to manage tuples in the set.*

- $\varphi\mathtt{Dic.Crt}(\{e_i | 1 \le i \le n\})$:
  - $\mathtt{Dic_I.Crt}(\{(\Delta_i^{(B)}, data_i) | 1 \le i \le n\})$.
  - $\mathtt{Dic_F.Crt}(\{(\Delta_i^{(ID)}, \Delta_i^{(B)}, \&(data_i)) | 1 \le i \le n\})$.
- $\varphi\mathtt{Dic.Ins}(\{e_i | 1 \le i \le n\})$:
  - $\mathtt{Dic_I.Ins}(\{(\Delta_i^{(B)}, data_i) | 1 \le i \le n\})$.
  - $\mathtt{Dic_F.Ins}(\{(\Delta_i^{(ID)}, \Delta_i^{(B)}, \&(data_i)) | 1 \le i \le n\})$.
- $\varphi\mathtt{Dic.Get}(\Delta^{(B)}) \rightarrow \{(\Delta^{(ID)}, data)\}$ *or* $\perp$:
  - *If* $\mathtt{Dic_I.Get}(\Delta^{(B)}) = \perp$ *then return* $\perp$.
  - $\{(\Delta^{(ID)}, data)\} \leftarrow \mathtt{Dic_I.Get}(\Delta^{(B)})$.
- $\varphi\mathtt{Dic.Get}(\Delta^{(ID)}) \rightarrow data$ *or* $\perp$:
  - *If* $\mathtt{Dic_F.Get}(\Delta^{(ID)}) = \perp$ *then return* $\perp$.
  - $(\Delta^{(B)}, \&(data)) \leftarrow \mathtt{Dic_F.Get}(\Delta^{(ID)})$.
  - *Return* $(\Delta^{(B)}, data)$ *by address resolution of data.*
- $\varphi\mathtt{Dic.Upd}(\Delta^{(ID)}, data')$:
  - *If* $\mathtt{Dic_F.Get}(\Delta^{(ID)}) = \perp$ *then return* $\varphi\mathtt{Dic}$.
  - *Otherwise* $\mathtt{Dic_F.Upd}(\Delta^{(ID)}, data')$.

Here, $\mathtt{Dic_F}$ is a one-to-one dictionary, and $\mathtt{Dic_I}$ is a one-to-many dictionary. That means $\mathtt{Dic_I.Crt}()$ will append multiple tuples $< \Delta^{(ID)}, data >$ to the access entry $\Delta^{(B)}$ and $\mathtt{Dic_I.Ins}()$ will append extra tuples $< \Delta^{(ID)}, data >$ to the tails. Other operations work in the same way as single dictionary.

4 CONSTRUCTION OF LS-RQ

In this section, the detailed construction of a novel light and forward-secure range query on geometric encrypted data is presented. We shall discuss the framework in single setting

and multiple setting in sequence, where single and multiple indicate the number of even LSH functions employed.

4.1 LS-RQ in Single Setting

An LS-RQ scheme based on a single LSH function is named LS-RQ-S, which is also illustrated in supplementary 2, available online.

Setup. This primitive takes as input a dataset $\mathbb{D}$ and data encryption keys. It generates a forwardly inverted dictionary φDic as the output.

Specifically, a security parameter 1^λ is selected according to application requirements. A keyed hash function $H : \{0,1\}^* \times \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$, a proxy re-encryption algorithm PRE and a pseudo-random permutation (PRP) $\pi : \{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$ are generated according to the security parameter 1^λ. The public parameter for such a range query system is $SP = (\lambda, H, prm)$, where prm is the public parameter in PRE and generated by PRE.Setup(). Both complex buckets $\mathbb{B}$ and LSH-mapping dictionary $\mathtt{Dic}_\mathtt{H}$ are derived as described in Section 3.2.

To provide secure guarantee, two pairs $< cpk_i, csk_i >$ and $< upk_i, usk_i >$ of public and secret keys are generated by PRE.KenGen() for each complex bucket $\mathbb{B}_i$. Additionally, a key dictionary $\mathtt{Dic}_\mathtt{key}$ containing a set $\{(B_i, (< cpk_i, csk_i >, < upk_i, usk_i >, < npk_i = \mathtt{null}, nsk_i = \mathtt{null} >))|1 \leq i \leq \#(\mathbb{B})\}$ is maintained at the LBS provider side. A forwardly inverted dictionary φDic is created based on $\mathtt{Dic}_\mathtt{key}$. In φDic, $\Delta^{(ID)} = \pi(ID), \Delta^{(B)} = H(B, csk_i)$ and $data = \mathtt{PRE.Enc}(cpk_i, \boldsymbol{p})$.

Insertion. This primitive takes as input the newly added dataset AddSet and φDic, and outputs a newly generated φDic. Specifically, at the client side, AddSet containing all newly inserted points is generated. Concretely, all the new points are scattered into brand-new complex buckets $\mathbb{B}'$, both that and $\mathbb{B}$ share the same LSH-mapping dictionary $\mathtt{Dic}_\mathtt{h}$. All the newly added points are encrypted with a new key pair $< upk, usk >$ to provide forward-security. That means $\Delta^{(ID)} = \pi(ID)$, $\Delta^{(B)} = H(B, usk_i)$ and $data = \mathtt{PRE.Enc}(upk_i, \boldsymbol{p})$. $\Delta^{(ID)}$, $\Delta^{(B)}$ and $data$ are further added into AddSet. Then, AddSet is sent to the cloud, who will insert each element in AddSet into φDic by φDic.Ins() in Definition 4.

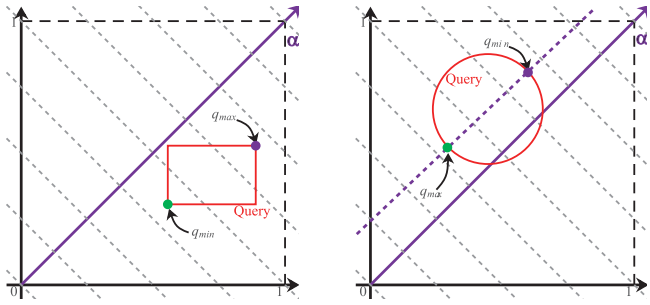
Query. This primitive takes as input a range query Q and φDic, and outputs the query result by a single interaction between a client and a public cloud. Generally, executing a query is divided into three relatively independent processes.

First, the query itself is translated into a set of identifiers for complex buckets which completely cover areas that the query falls in. For a rectangle query, $Q = (\boldsymbol{q}_{\min}, \boldsymbol{q}_{\max})$. For circular query, $Q = (\boldsymbol{o}, r)$ where $\boldsymbol{o} = (o_x, o_y)$ is the center and r is the radius. Explicitly, $\boldsymbol{q}_{\min}$ and $\boldsymbol{q}_{\max}$ can be calculated by resolving the following system of equations.

$$\begin{cases} (x - o_x)^2 + (y - o_y)^2 = r^2 \\ (y - o_y)/(x - o_x) = \alpha_y/\alpha_x \end{cases}. \tag{2}$$

There are obviously two solutions, $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$, for Equation (2). Here, $\boldsymbol{q}_{\min} = (x_{\min}, y_{\min})$ and $\boldsymbol{q}_{\max} = (x_{\max}, y_{\max})$. Also, the generic idea of query translation is illustrated in Fig. 4.

Second, the corresponding token τ for query is generated and sent to the cloud by leveraging $\mathtt{Dic}_\mathtt{key}$ and $\mathtt{Dic}_\mathtt{H}$ as shown



(a) Rectangle query. (b) Circular query.

Fig. 4. Generic idea for bucket mapping of rectangle and circular queries.

in function ClientQuery() in Algorithm 3. Client requests $\mathtt{Dic}_\mathtt{key}$ and $\mathtt{Dic}_\mathtt{H}$ from the LBS provider. For each candidate bucket, a new generated key pair $< npk, nsk >$ is for re-encrypting points(Line 4), and csk and usk are retrieved from $\mathtt{Dic}_\mathtt{key}$ (Line 5). A token τ that contains several trapdoors for quick retrieval is generated (Lines 6-11). Then, $< upk, usk >$ is updated for the touched bucket (Lines 12-13). Finally, τ is sent to the cloud.

Algorithm 3. Query for Client in LS-RQ-S

1: **function** ClientQuery (h, $\mathtt{Dic}_\mathtt{key}$, $\mathtt{Dic}_\mathtt{H}$, Q)
2: $(\mathtt{h}_{\min}, \mathtt{h}_{\max}) \leftarrow$ BucketMap (h, $\mathtt{Dic}_\mathtt{H}$, Q), $\tau \leftarrow \emptyset$, $\mathtt{Dic}_\mathtt{sk} \leftarrow \emptyset$;
3: **for** each $\mathtt{h} \in [\mathtt{h}_{\min}, \mathtt{h}_{\max}]$ **do**
4: $< npk, nsk > \leftarrow$ PRE.KeyGen(), $B \leftarrow \mathtt{Dic}_\mathtt{H}(\mathtt{h})$;
5: $(< cpk, csk >, < upk, usk >) \leftarrow \mathtt{Dic}_\mathtt{key}(B)$;
6: $\Delta_1^{(B)} \leftarrow H(B, csk)$, $\Delta_2^{(B)} \leftarrow H(B, usk)$;
7: $\Delta^{(B)'} \leftarrow H(B, nsk)$;
8: $rk_{c\rightarrow n} \leftarrow$ PRE.ReKeyGen(csk, nsk);
9: $rk_{u\rightarrow n} \leftarrow$ PRE.ReKeyGen(usk, nsk);
10: Add $\{\Delta_1^{(B)}, \Delta_2^{(B)}, \Delta^{(B)'}, rk_{c\rightarrow n}, rk_{u\rightarrow n}\}$ into τ;
11: Add $\{B, nsk\}$ into $\mathtt{Dic}_\mathtt{sk}$;
12: $< upk, usk > \leftarrow$ PRE.KeyGen();
13: $\mathtt{Dic}_\mathtt{key}$.Upd$(B, < npk, nsk >, < upk, usk >, \phi)$;
14: **end for**
15: **return** τ and $\mathtt{Dic}_\mathtt{sk}$;
16: **end function**
17: **function** ClientDecrypt ($\mathtt{Dic}_\mathtt{Can}$, $\mathtt{Dic}_\mathtt{sk}$, Q)
18: $\sigma \leftarrow \emptyset$;
19: **for** each $(\Delta^{(B)}, \mathtt{CanSet}) \in \mathtt{Dic}_\mathtt{Can}$ **do**
20: $nsk \leftarrow \mathtt{Dic}_\mathtt{sk}$.Get$(\Delta^{(B)})$;
21: **for** each $(\Delta^{(ID)}, data) \in \mathtt{CanSet}$ **do**
22: Add $data' =$ PRE.Dec$(nsk, data)$ into ResSet;
23: **end for**
24: **end for**
25: **return** the refined ResSet with Q;
26: **end function**

Third, the cloud scans φDic and gathers all candidate points through Algorithm 4. A candidate set is empty-initialized at the beginning (Line 2). The token for query contains several sub-tokens. For each sub-token, all encrypted candidates are retrieved by matching $\Delta_1^{(B)}$ and $\Delta_2^{(B)}$ (Lines 4-6). Then the candidate set is prepared by two loops (Lines 7-18). Generally, the algorithm looks up all candidate points that are encrypted with cpk (resp. usk) in the first (resp. second) loop. In a single loop, the encrypted data are refreshed (Line 8, 14). Following that, φDic is updated to

ensure forward-security (Lines 9, 15). Meanwhile, the data encrypted under upk in $\varphi\texttt{Dic}$ is removed (Line 16). At the end of each loop, bucket labels are updated (Line 12) and the candidates are added into the candidate set (Lines 10, 17). The candidate set is returned to the client (Lines 19, 21).

Algorithm 4. Query for Cloud in LS-RQ-S

1: **function** CLOUDQUERY ($\varphi\texttt{Dic}$, τ)
2: $\texttt{Dic}_{\texttt{Can}} \leftarrow \emptyset$;
3: **for** each element in τ **do**
4: Parse the element and let CanSet be empty;
5: $\{(\Delta^{(ID)}, data)\}_1 \leftarrow \varphi\texttt{Dic.Get}(\Delta_1^{(B)})$;
6: $\{(\Delta^{(ID)}, data)\}_2 \leftarrow \varphi\texttt{Dic.Get}(\Delta_2^{(B)})$;
7: **for** each element in $\{(\Delta^{(ID)}, data)\}_1$ **do**
8: $data' \leftarrow \texttt{PRE.ReEnc}(rk_{c\rightarrow n}, data)$;
9: $\varphi\texttt{Dic.Upd}(\Delta^{(ID)}, data')$;
10: Add $(\Delta^{(ID)}, data)$ into CanSet;
11: **end for**
12: Replace $\Delta_1^{(B)}$ with $\Delta^{(B)'}$;
13: **for** each element in $\{(\Delta^{(ID)}, data)\}_2$ **do**
14: $data' \leftarrow \texttt{PRE.ReEnc}(rk_{u\rightarrow n}, data)$;
15: $\varphi\texttt{Dic.Dic}_{\texttt{I}}\texttt{.Ins}(\Delta^{(B)'}, data')$;
16: Remove elements with access entry $\Delta_2^{(B)}$;
17: Add $(\Delta^{(ID)}, data)$ into CanSet;
18: **end for**
19: $\texttt{Dic}_{\texttt{Can}}\texttt{.Ins}(\Delta^{(B)'}, \texttt{CanSet})$;
20: **end for**
21: **return** $\texttt{Dic}_{\texttt{Can}}$;
22: **end function**

At the client side, as shown in Algorithm 3, finally, the query result is derived (Lines 18-26). Specifically, the client first looks up the secret key nsk by matching bucket label $\Delta^{(B)}$ (Line 20). After that, the encrypted candidate points are correctly decrypted (Lines 21-23). The candidate set will be refined to derive the accurate result (Line 25), in which all decrypted candidate points are compared with the query itself and all false-positive candidate points are removed.

Updating. This primitive takes as input a point with an identifier $(ID, \boldsymbol{p})$ and $\varphi\texttt{Dic}$, and outputs a modified $\varphi\texttt{Dic}$ where p is updated. For a client, given an updating point $(ID, \boldsymbol{p})$, updating token τ is generated and sent to the cloud. The Client first looks up both cpk and upk with the help of $\texttt{Dic}_{\texttt{H}}$ and $\texttt{Dic}_{\texttt{key}}$ with the help of the LBS provider. The corresponding access entries, $\Delta_1^{(B)} \leftarrow H^{(B)}(B, csk)$, $\Delta_2^{(B)} \leftarrow H^{(B)}(B, usk)$ and $\Delta^{(ID)} \leftarrow \pi(ID)$, are generated. After that, the new point is encrypted with cpk and upk respectively such that $data_1 \leftarrow \texttt{PRE.Enc}(cpk, \boldsymbol{p})$ and $data_2 \leftarrow \texttt{PRE.Enc}(upk, \boldsymbol{p})$. The token combines the above components. At the cloud side, both $\varphi\texttt{Dic.Upd}(\Delta^{(ID)}, data_1)$ and $\varphi\texttt{Dic.Upd}(\Delta^{(ID)}, data_2)$ are executed afterwards.

Deletion. This primitive takes as input an identifier ID of a point and $\varphi\texttt{Dic}$, and outputs a modified $\varphi\texttt{Dic}$ where $\boldsymbol{p}$ is set to a random value. Generally, the deleting point is replaced by a fake point. Indeed, deletion is completed by covering the corresponding element with a fake point. At the client side, a fake point $\bar{\boldsymbol{q}}$ is randomly generated, and the following procedures are the same as that for updating. In fact, the cloud only receives a request to execute $\varphi\texttt{Dic.Upd}()$. So, it cannot distinguish it from a real process of updating. Hence, since the cloud is semi-honest, it will delete $\boldsymbol{q}$.

4.2 LS-RQ in Multiple Setting

In LS-RQ-S, for a range query, there is a non-negligible probability that the scale of the returned candidate set is much larger than that of the query's accurate result (a piece of evidence is in Fig. 6.). Hence, the communication cost between a public cloud and a client is heavy. In order to further reduce the communication burden of the client, a novel multiple setting mechanism is introduced in this section, by adopting multiple LSH functions.

The forward-secure range query based on multiple LSH functions is named LS-RQ-M. The main body is almost similar to that in single setting with a significant difference that, given a dataset, m LSH functions are adopted. Thus, the cloud holds m copies of $\varphi\texttt{Dic}$, and the LBS provider holds m copies of $\texttt{Dic}_{\texttt{key}}$ and $\texttt{Dic}_{\texttt{h}}$. When conducting a range query, the query process of LS-RQ-S is executed m times. There will be m copies of the generated candidate set. The query result is indeed the intersection of all candidate sets, since that the real result must be included in every candidate set. Other processes (i.e., *Creation*, *Insertion*, *Updating*, and *Deletion*) are implemented in a similar way.

It is worth noting that it is infeasible to directly remove redundant candidate points that are encrypted with different public keys. In order to further reduce the communication cost between a public cloud and a client, another public cloud (named removal cloud), that does not collude with the existing one (named query cloud), is introduced for duplicate removal.

Algorithm 5. Duplicate Removal in LS-RQ-M

1: **function** REMOVALCLIENT $\{\texttt{Dic}_{\texttt{sk}}\}_{i=1}^m$, $\{\pi\}_{i=1}^m$
2: $\tau \leftarrow \emptyset$, $< rsk, rpk > \leftarrow \texttt{PRE.KeyGen}()$;
3: **for** $i \in [1, m]$ **do**
4: **for** each $\{B, nsk\}$ in $\texttt{Dic}_{\texttt{sk}}^{(i)}$ **do**
5: $rk_{n\rightarrow r} \leftarrow \texttt{PRE.ReKeyGen}(rsk, nsk)$;
6: $\texttt{Dic}_{\texttt{sk}}^{(i)}\texttt{.Upd}(B, rk_{n\rightarrow r})$;
7: **end for**
8: Add $\{\pi_i^{-1}, \texttt{Dic}_{\texttt{sk}}^{(i)}\}$ into τ;
9: **end for**
10: **return** τ;
11: **end function**
12: **function** REMOVALCLOUD $\{\texttt{CanSet}\}_{i=1}^m$, τ
13: $\texttt{Can} \leftarrow \emptyset$;
14: **for** $i \in [1, m]$ **do**
15: **for** each $\{B, rk_{n\rightarrow r}\}$ in $\texttt{Dic}_{\texttt{sk}}^{(i)}$ **do**
16: **for** each $(\Delta^{(ID)}, data)$ in CanSet **do**
17: $data' \leftarrow \texttt{PRE.ReEnc}(rk_{n\rightarrow r}, data)$;
18: $\Delta^{(ID)'} \leftarrow \pi_i^{-1}(\Delta^{(ID)'})$;
19: Add $(\Delta^{(ID)'}, data')$ into Can;
20: **end for**
21: **end for**
22: **end for**
23: **return** $\cap\texttt{Can}$;
24: **end function**

Duplicate Removal. At the client side, a token is generated as shown in Algorithm 5 and is sent to a removal cloud. At the very beginning, a removal key pair $< rsk, spk >$ is generated as a session key (Line 2), which is only used in this process of range query. For each $\varphi\texttt{Dic}$, a re-encryption key is generated by the adopting proxy re-encryption algorithm

(Line 5). Then, the corresponding key in $\mathtt{Dic}_{\mathtt{sk}}$ is updated for each complex bucket (Line 6). Finally, the generated token (Lines 8, 10) is sent to the removal cloud. After receiving removal token and encrypted candidate sets, the removal cloud carries out ClientQuery() in Algorithm 5 to remove redundant points. First, in a loop (Lines 14-22), all candidate points under different LSH functions are re-encrypted (Line 17), and all labels for ID are inversely permuted to recover the original IDs (Line 18). Finally, all candidate points are intersected according to their IDs for removing redundant points (Line 23), which will be sent back to the client.

Finally, the client receives encrypted candidate points and correctly decrypts them with rsk, as they have been already re-encrypted with $rk_{n\to r}$ on the removal cloud. The candidate points will be refined to derive the accurate result in a similar manner of LS-RQ-S. For easy understanding, a workflow is also illustrated in supplementary 2, available online.

5 Theoretical Analysis

Herein we theoretically analyze the correctness, security, efficiency, and comparisons of LS-RQ sequentially.

5.1 Theoretical Correctness

An LBS provider holds two dictionaries, $\mathtt{Dic}_{\mathtt{h}}$ and $\mathtt{Dic}_{\mathtt{key}}$. Mapping relations between LSH values of data and identifiers of complex buckets are maintained in $\mathtt{Dic}_{\mathtt{h}}$. The key pairs for generating token are stored in $\mathtt{Dic}_{\mathtt{key}}$.

A client, in order to issue a query in single setting, submits the query itself to the LBS provider, which calculates LSH value by leveraging (even) LSH functions, and further picks up identifier B for corresponding complex buckets by looking up $\mathtt{Dic}_{\mathtt{h}}$. The complex buckets are generated, by adopting even LSH function and greedy merge method. In such a way, both rectangle and circular range are mapped into two bounded points, $\boldsymbol{q}_{min}$ and $\boldsymbol{q}_{max}$. According to line 2 and the outermost loop in Algorithm 3, $\boldsymbol{q}_{max}$ and $\boldsymbol{q}_{min}$ are mapped into LSH values, $\mathtt{h}_{min}$ and $\mathtt{h}_{max}$. All points in dataset that falls in complex buckets with LSH values $\mathtt{h} \in [\mathtt{h}_{min}, \mathtt{h}_{max}]$ are totally gathered as encrypted candidates. So, the encrypted candidates totally cover the real result of a range query.

Additionally, the candidates are encrypted with upk while inserting into dataset as shown in *Insertion*. The fetching candidate is stored as $c_u = \mathtt{PRE.Enc}(upk_i, \boldsymbol{p_i})$. According to line 17 in Algorithm 5, the encrypted candidate sent back to the client is $c_r = \mathtt{PRE.ReEnc}(rk_{n\to r}, \mathtt{PRE.ReEnc}(rk_{u\to n}, c_u))$. The client decrypts it and gets $m = \mathtt{PRE.Dec}(rsk, c_r)$. Obviously, according to the *correctness* of PRE as defined in Section 2.4, $\boldsymbol{p_i}$ can be correctly decrypted due to the following equation,

$$\begin{aligned}
&\mathtt{PRE.Dec}(rsk, c_r)\\
=&\mathtt{PRE.Dec}(rsk, \mathtt{PRE.ReEnc}(rk_{n\to r}, \mathtt{PRE.ReEnc}(rk_{u\to n}, c_u)))\\
=&\mathtt{PRE.Dec}(rsk, \mathtt{PRE.ReEnc}(rk_{n\to r}, c_n))\\
=&\mathtt{PRE.Dec}(rsk, c_r) = \boldsymbol{p_i}.
\end{aligned}$$

In a similar manner, the fetching encrypted candidates, that are encrypted in *Setup* and are re-encrypted in lines 8 and 9 of Algorithm 4, can also be correctly decrypted.

5.2 Theoretical Security

$\mathcal{L}$-Forward-Adaptive-Security of LS-RQ. During LS-RQ, all transmitting data are encrypted by PRE. Confidentiality of both data and query is satisfied since the leverage of PRE. In LS-RQ, the leakage during *Insertion* is only the set of IDs of newly added points. Furthermore, by re-encrypting the touched data, LS-RA is forward-secure. The formal analysis is particularly stated in supplementary 3, available online.

Secure Consideration Under Access Pattern Leakage. Recently, several works reconstruct 1-dimensional dataset only from serious of access pattern leakage's variants, such as the volume of range query's result [11], [12], access pattern itself [13], [14], etc. All the works lie on two underlying assumptions, 1) there must be a partial order relationship between elements in a dataset, and 2) the leakage of access pattern must be specific and accurate.

Analysis for Assumption 1. Actually, the above attacks work for datasets comprising keywords or 1-dimensional numerical values (such as ages, salaries, etc.), since that lexicographical order is inherent for keywords and 1-dimensional numerical values are inherently partial ordered. However, there is, currently, no effective partial order for geographical data (i.e., 2-dimensional points). Specifically, for any two points, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, the relations between cannot be legibly defined except equivalence. So, the above attacks do not work for LS-RQ.

Analysis for Assumption 2. In most existing secure (range) query mechanisms [4], [7], [16], [18], [19], [20], [21], [32], the returned encrypted candidate sets are specific and accurate, which means that the returned candidate sets include only accurate results without any false positive or true negative candidates. The above attacks work for the above mechanisms. LS-RQ is implemented in an approximate manner for the cloud. Specifically, from a cloud perspective, the candidate set retained is a supplementary collection of the accurate result. As illustrated in Fig. 2b, given $\ell = 10$, query's span is 1 percent and $\boldsymbol{\alpha} = (\alpha_x, \alpha_y)$, the accurate covered area is $A_I = (1\%)^2 = 0.0001$ and the minimally realistic touched area is $A_R = (1\% \cdot \sqrt{2})^2 = 0.0002$ when $\alpha_x = \alpha_y$. Thus, the realistic number of touched points is at least twice that of the accurate result. And, also, the scale fact is varying while the location of range changes. The precision that will be reported in Fig. 6 is evidence of this point. Hence, in the view of the cloud, the retained access pattern is unspecific and inaccurate.

In a word, the two crucial assumptions for attack under access pattern leakage are unsatisfactory. So, the presented LS-RQ can resist such attacks.

5.3 Theoretical Efficiency

In LS-RQ, several pivotal parameters matter efficiency. n is the scale of dataset. m is the number of the adopting LSH functions. The code length of LSH value is ℓ. $c = 2^\ell$ is the number of complex buckets for a single LSH function. The security parameter λ determines the bit length of keyed hash functions and the length of identifiers. Query's span is assumed to be a magnitude greater than c. In general, t_h, t_u, and t_c are the time for executing keyed hash function, $\varphi\mathtt{Dic.Upd}()$, and comparison once, respectively. The time of encryption, re-encryption, decryption, key generation and re-encryption key generation for PRE are t_e, t_r, t_d, t_k and t_{rk}, respectively. The time for executing PRP once is t_π.

TABLE 2
Theoretical Comparisons

Schemes	Type	Forward-security	Cloud Model	Interaction	Cryptographic Tools*
CRT [2]	Rectangle & Circular	Not Supported	Semi-honest	Yes	SE
RASP-QS [4]	Circular	Not Supported	Semi-honest	No	OPE
EPLQ [32]	Circular	Not Supported	Semi-honest	No	BP
SPSQ [7]	Polygons	Not Supported	Semi-honest	No	BP
DSSE [16]	Keyword	Supported	Fully Trusted	No	SE & PRF
Janus [18], Janus++ [19], Mitra [20], Bunker [21]	Keyword	Supported	Semi-honest	No	PE, SPE, PRF, OMAP, SGX
LS-RQ	Rectangle & Circular	Supported	Semi-honest	No	PRP & PRE

SE is symmetric encryption, OPE is order-preserving encryption, BP is bilinear pairing, PE is puncturable encryption, SPE is symmetric PE, PRF is pseudo-random function, PRP is pseudo-random permutation, OMAP is obvious map, Intel SGX is a set of security-related instruction codes that are built into some modern Intel CPUs, and PRE is proxy re-encryption.

Storage at Cloud. φDic is stored and maintained on a public cloud. For LS-RQ-S, $(\Delta^{(ID)}, \Delta^{(B)})$ and $(\Delta^{(B)}, \Delta^{(ID)}, data)$ are stored for a single piece of data. There is a certain scale of supplementary fake data. However, compared with the scale of a dataset, the scale of supplementary fake data is limited, which will be testified in Section 6. So, the storage costs for LS-RQ-S and LS-RQ-M at the cloud side are about $5\lambda n$ and $5\lambda mn$ bits.

Storage at LBS Provider. An LBS provider holds both $\mathtt{Dic}_{\mathtt{key}}$ and $\mathtt{Dic}_{\mathtt{h}}$. In LS-RQ-S, the former dictionary keeps $(B, (< cpk, csk >, < upk, usk >, < npk, nsk >))$ and the another dictionary keeps $(\mathtt{h}, B)$ for each complex buckets. So, the storage cost for LS-RQ-S and LS-RQ-M at the LBS provider side are $7\lambda c + 2^{\ell+1}\lambda$ and $7\lambda mc + 2^{\ell+1}\lambda m$ bits.

Time Cost at Cloud. For range query, the most time-consuming operation is $\varphi\mathtt{Dic.Upd}()$. In LS-RQ-S, the time cost are $10nt_u/c$. In LS-RQ-M, there is an additional duplicate removal operation. So, the time cost in LS-RQ-M is $10mnt_u/c + mt_c$, in which mt_c is the time cost for duplicate removal operation. In primitives of data updating, insertion and deletion, the time-consuming operation is $\varphi\mathtt{Dic.Upd}()$. In both LS-RQ-S and LS-RQ-M, time costs are t_u and mt_u respectively.

Time Cost at LBS Provider. For range query, the time-consuming operations are keyed hash function and re-encryption key generation. In LS-RQ-S and LS-RQ-M, the time costs are $3ct_h + 2ct_{rk}$ and $3mct_h + mct_{rk}$. For data insertion in LS-RQ-S and LS-RQ-M, the time costs are $ct_h + nt_e + t_\pi$ and $mct_h + mnt_e + mt_\pi$. For both data updating and deletion in LS-RQ-S and LS-RQ-M, the time costs are $2t_h + 2t_e + t_\pi$ and $2mt_h + 2mt_x + mt_\pi$.

Transmission Overhead Between a Client and a Public Cloud. For range query, in LS-RQ-S, τ is generated in the line 10 of Algorithm 3. The transmission overhead is $5c\lambda$ bits. In LS-RQ-M, the transmission overhead is $5cm\lambda$ bits. For inserting n new points, AddSet is a triple tuple. So, in LS-RQ-S and LS-RQ-M, the transmission overheads are $3\lambda n$ and $3\lambda mn$ bits. For data deletion, in LS-RQ-S and LS-RQ-M, the transmission overheads are 4λ and $4\lambda m$ bits.

For eLS-RQ-S and eLS-RQ-M that adopt even LSH functions, the costs are very similar to that of LS-RQ-S and LS-RQ-M. Hence, the analysis is omitted here.

5.4 Theoretical Comparisons

As shown in Table 2, there are three typical range queries in practice, i.e., rectangle, circular and polygons. LS-RQ schemes support both rectangle and circular range queries as other state-of-art schemes do. In terms of security, existing secure range query schemes are not forward-secure. LS-RQ is the first attempt to design a forward-secure mechanism that supports the range query. Also, a cloud is assumed to be semi-honest as other state-of-art schemes are.

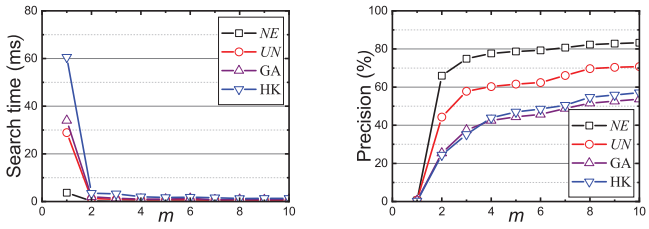
Interaction and cryptographic tools can significantly matter efficiency. The proposed LS-RQ, as well as the schemes [4], [7], [16], [32], need only a single round of interaction, which is an essential condition for providing high efficiency of query and lightweight characteristic for a client. Furthermore, in popular searchable encryption mechanisms [2], [4], [16], [18], [19], [20], [21] as well as LS-RQ, lightweight cryptographic tools are adopted.

Comparisons of asymptotic complexity between LS-RQ and other schemes that support range queries are analyzed in depth in Table 3. Note that only heavy computations are

TABLE 3
Asymptotic Comparisons[a]

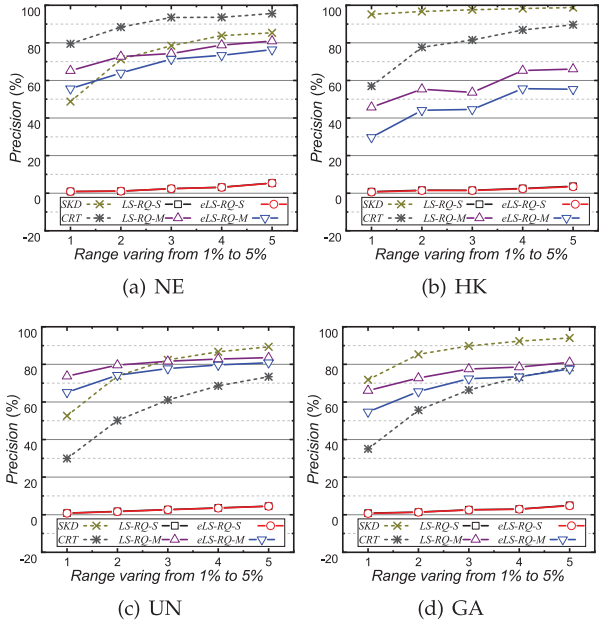
Schemes	# Interactions	Computation at Cloud
CRT [2][b]	$O(\log n)$	-
RASP-QS [4][c]	1	$O(m \log n) \cdot C_o$
EPLQ [32][d]	1	$O(\log(n + n_r)) \cdot n_c\hat{e}$
SPSQ [7][d]	1	$4nn_rE + 2nn_rM$
LS-RQ[e]	1	$\xi n_r E_r$
LS-RQ-M[e]	1	$\xi mn_r E_r$
Schemes	**Communication**	**Computation at Client**
CRT [2][b]	$O(\log n)$	$O(\log n) \cdot E_s^{-1}$
RASP-QS [4][c]	$O(n_r)$	$O(n_r) \cdot E_o^{-1}$
EPLQ [32][d]	$O(n_r)$	$2n_c^2M + 2n_c^2A + n_r \cdot E_s^{-1}$
SPSQ [7][d]	$O(n_r)$	$2rE + 4r\hat{e} + 2rM + n_r \cdot E_s^{-1}$
LS-RQ[e]	$O(\xi n_r)$	$\xi n_r E_p^{-1}$
LS-RQ-M[e]	$O(\xi mn_r)$	$\xi mn_r(E_p^{-1} + C) \approx \xi mn_r E_p^{-1}$

[a]*In this table, n is the scale of dataset, and $n_r << n$ is the accurate scale of range query's result. Additionally, $O(f(n))$ means that, for large enough n, the complexity for running time, interaction, or communication cost is at most $k \cdot f(n)$ for some constant k.*
[b]*E_s^{-1} is symmetric decryption.*
[c]*C_o and E_o^{-1} are comparison and decryption under OPE.*
[d]*M, A, E and $\hat{e}$ are multiplication, addition, exponentiation, and pairing under bilinear pairing. The length n_c(is suggested to be 37) of encoded vectors is particular for EPLQ [32].*
[e]*E_r is the re-encryption under PRE, and E_p^{-1} is the decryption under PRE. ξ is an expansion factor which is small. The number of adopting LSH functions in LS-RQ-M is m. C is the comparison between plaintexts.*



(a) Search time VS. m (b) Precision VS. m

Fig. 5. Effect of m on datasets.



(a) NE (b) HK

(c) UN (d) GA

Fig. 6. The evaluation of precision.

reported in the comparisons. Obviously, in a lightweight secure range query schemes, the number of interactions between a public cloud and a client must be limited at one single round. Also, the computational burden for clients should be minimal, which means that a client should only decrypt results without any other computation. By observation, only RASP-QS [4] and LS-RQs simultaneously satisfy the above two conditions, since that time for comparison C on the plaintext is much smaller than that for E_p^{-1} and can be ignored. However, EPLQ is not forward-secure, which is satisfied with LS-RQs. Furthermore, the complexities of LS-RQs are at the same level as other state-of-art schemes.

Additionally, during transmitting encrypted candidate points, since the length of the ciphertext in PRE is much bigger than that in symmetric encryption mechanisms, communication cost increases in LS-RQ. The latter cannot support forward-security, in spite of that, it can minimize the length of the ciphertext. In the experimental studies of LS-RQ, PRE in literature [27] is adopted. The length of the ciphertext is about 1,856 bits. In practice, the length of a 2-dimensional point with a unique identifier is $32 + 64 * 2 = 160$ bits. So, during transmitting encrypted candidate points in LS-RQ, the communication cost is about 12 times than that in the plain domain. However, in LS-RQ-M, the number of returned encrypted candidate points is strictly small than 500. Hence, the communication cost is only about 113 kB, which is still practical.

6 Experimental Study

In this section, we present experimental studies to evaluate LS-RQ under different workloads. First, we tune parameter m to find appropriate values to present a sound trade-off between efficiency and accuracy. Then we compare LS-RQs with two related secure range schemes (i.e., CRT [2] and SKD[6]) to show superiority. At last, we further investigate the insertion performance of LS-RQ. All experimental studies are evaluated on an x64 machine with Intel (R) Xeon (R) E5-2630 v3*2 @ 2.20 GHz and 384GB RAM.

In the experimental studies, there are two workloads that are extracted from publicly real dataset, HK and NE. HK[7] contains 1,384,420 points in Hong Kong, China. NE[8] contains 123,593 points in North East, USA. We also generate 2-dimensional uniform and Gaussian datasets, named UN and GA. Each dataset contains 1,000,000 data points that follow uniform/Gaussian distribution. We adopt PRE in literature [27]. For fair and valuable evaluations, 100 random queries are generated to gain average performances, and the number $c = 2^{10}$ (i.e., $\ell = 10$) of complex buckets is fixed. In experimental comparisons and performance evaluations, queries' spans are tuned from 1 to 5 percent.

6.1 Parameter Tuning

The number m of the adopted LSH functions is a pivotal parameter in LS-RQ-M. We varied $m \in [1, 10]$ to find the most appropriate values for each dataset, as shown in Fig. 5. Obviously, when m is greater than 3, the search time at the client side is sufficiently low for resource-constraint devices (i.e., smartphone, pad, end node in IoT, etc.). With the growth of m, the precision increases while the search time at the client side decreases and storage overhead grows linearly. For an optimized balance between storage overhead and precision, we let m be 4 for all testing datasets.

6.2 Experimental Comparisons

In this section, we compare LS-RQ approaches with both CRT and SKD from several aspects. The prefix e of eLS-RQ-S and eLS-RQ-M indicates that even LSH function in Section 3.1 is adopted instead of the standard one. Specifically, the even LSH function is calculated by Corollary 1 as shown in supplementary 4, available online. The query is restricted in the type of rectangle. In realistic applications, the performance of a circular range query is highly similar to that of a rectangle range query. So, the experiments of a circular range query are ignored here.

Precision on Various Datasets. The precision performances on different datasets showing in Fig. 6 are almost stable. The precision is calculated by the following equation,

$$Precision = \frac{\#(\sigma)}{\#(\texttt{CanSet})}. \tag{3}$$

6. SKD is a new-designed range query scheme by encrypting nodes in KD-tree with comparable encryption (CE) [33], which is an high-efficient encryption scheme that supports direct comparison between ciphertexts. Both CRT and SKD are adopted as reference schemes to illustrate φDic's superiority compared to popular index structures, R-tree and KD-tree, respectively.

7. http://metro.teczno.com/#hong-kong

8. http://www.rtreeportal.org

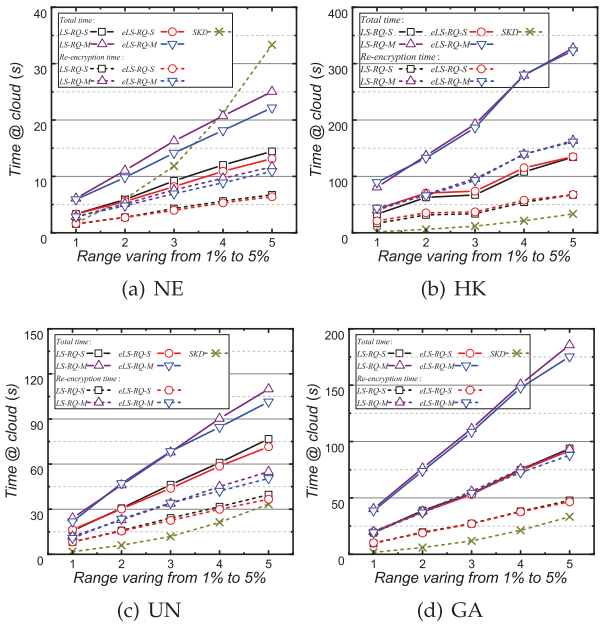


Fig. 7. The evaluation of search time at the cloud side. (CRT is absent since the vast majority of computing burden are at the client side and the cloud in CRT does nearly nothing except returning encrypted nodes in R-tree.)

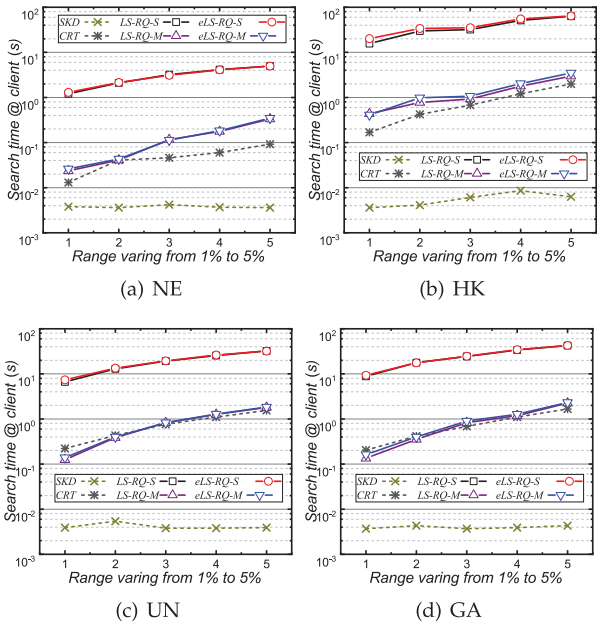


Fig. 8. The evaluation of search time at the client side.

According to Equation (3), the bandwidth for transmitting candidate set is $\frac{1}{Precision}$ times as many as that for transmitting the accurate result. The time shifts from 1.2 to 3.4 in multiple settings (i.e., eLS-RQ-M and LS-RQ-M). The bandwidth overhead is obviously sound and acceptable in practice, while LS-RQ brings confidentialities for both data and query. So, such expenditure is tolerable. Straightforwardly, in single setting (i.e., eLS-RQ-S and LS-RQ-S) the bandwidth overhead is obviously unacceptable in practice. It is very consistent with the characteristics of LSH. In fact, multiple LSH functions are adopted to construct an index for higher precision rather than a single LSH function. Hence, we also suggest that multiple setting should be adopted in practice for reducing bandwidth overhead. The precisions in multiple setting are at least 14 (at most 94) times as many as that in single setting.

By contrast, both CRT and SKD are a bit better than the proposed LS-RQ schemes on all datasets. It is mainly because that the index structures in CRT and SKD (i.e., R-tree and KD-tree) can filter more false candidate points that of φDic. However, it is still challenging for both R-tree and KD-tree to provide forward-security, which can be efficiently achieved in φDic of LS-RQ schemes.

Search Time on Various Datasets. In LS-RQ schemes, most part of the processing query is performed at the cloud side. In general, as shown in Figs. 7 and 8, the processes at the cloud side are accomplished in minutes, and that at the client side is in seconds. That means LS-RQ shifts the heaviest computation overhead to the cloud and the client does not need expensive modules to accomplish heavy computations. Additionally, the growth trends for search time both at the cloud and client sides are consistent with the number of accurate result points. In fact, the decryption time is the dominant part in total search time both at the client side and cloud sides. Compared with the total search time at the cloud side as shown in Fig. 7, the process of re-encryption costs half of the time. It is noteworthy that the time cost for re-encryption at the cloud side is still acceptable for practice since one can accelerate re-encryption by strengthening the cloud's computation capacity. For clients, the decryption time is unavoidable in a secure search scheme on encrypted data and is linearly dependent on the size of $\#(\texttt{CanSet})$. Hence, the smaller $Precision$ is, the more time that decryption needs. Additionally, LS-RQ schemes in multiple setting (i.e., LS-RQ-M and eLS-RQ-M) tremendously outperform that in single setting (i.e., LS-RQ-S and eLS-RQ-S). The speedup factor is between 14 and 69. Generally, in practice, LS-RQ schemes in multiple setting are recommended.

By contrast, for search time at both the cloud and client sides, SKD is much less than LS-RQ schemes and CRT on all datasets. The reason is twofold. First, precisions for both CRT and SKD are generally higher than LS-RQ schemes. So the number of decrypting points in LS-RQs is more than that in both CRT and SKD. Additionally, the decryption algorithm (i.e., PRE and AES) in both LS-RQ schemes and CRT are much time-consuming than that in SKD (i.e., CE [33]). However, it is worth highlighting that our solutions to secure range query take strictly less than 1 second.

Storage Expansion on Various Datasets. To facilitate range queries on massive data, an index is pre-established before the data is subcontracted to a cloud. Furthermore, for security concerns, noisy points are attached to a dataset as shown in Algorithm 2. Storage expansion directly reflects the amounts of attached noisy points as shown in Fig. 9. In LS-RQ schemes

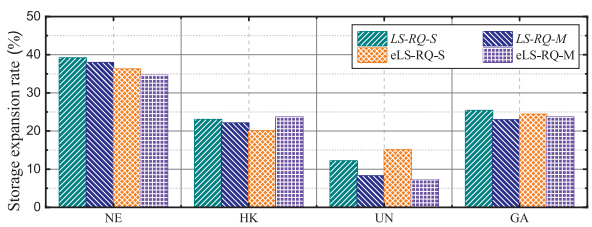


Fig. 9. The evaluation of storage expansion at the cloud side.

without even LSH, by observation, the storage expansion rate reaches only about 40 percent for a dataset with about 10,000 points (i.e., dataset NE). If the dataset contains about 100,000 points (i.e., dataset HK, UN, and GA), the storage expansion rate reduces to only about 22 percent or even more low. With embedding even LSH into LS-RQ, it is striking that the storage expansion rate reduces only to 7.3 percent for dataset UN. That means the proposed even LSH can divide the original dataset more uniformly with fewer noisy points and is more compatible with datasets with the pre-known distribution. There is a suggestion that eLS-RQ-M should be adopted while the distribution of a dataset is pre-known. In fact, the storage cost of LS-RQ on a public cloud is much heavier than CRT since the size of ciphertext usually is much bigger than the plaintext. The expansion factor is about 12 as analyzed in Section 5.4. Certainly, reducing the size of PRE's ciphertext is still challenging.

Additionally, LS-RQ is a dynamic scheme for range query, in which data can be integrated and indexed as time goes on. We thoroughly investigate insertion performance to exhibit the superiority of LS-RQ in supplementary 5, available online. In general, the insertion performances of LS-RQ are stable and sound in both storage cost and response time.

Summary of Experimental Comparisons. Both LS-RQ-M and eLS-RQ-M's performances on precision, search time both at the cloud and client sides are in the mainstream levels as related research articles have reported. Specifically, search time at the client side is in seconds and the precision remains at about 22 percent while LS-RQ is forward-secure which is, however, not satisfied in all existing secure range schemes. Additionally, for large scale datasets, the storage expansion rate is only about 30 percent at most. It is noteworthy that the storage expansion rate for eLS-RQ-M on dataset UN is only about 7.3 percent which is half of that for LS-RQ-M. It means that even LSH can dramatically reduce storage expansion rates and greatly release storage burden at the cloud side. Furthermore, forward-security is the dominant contribution in LS-RQ schemes while the state-of-art secure schemes cannot support.

7 Related Works

Searchable encryption is first presented by Song *et al.* [10]. Since that, many secure frameworks for various types of data have been proposed in the literature [34]. Recently, range query on geographically encrypted data has been resolved based on anonymity technique [1], [35], heavy encryption [3], [5], [7], secure index [2], [4], [6], etc. Here, we briefly summarize relevant methods.

Anonymity-Based Approaches. In [36], k-anonymity is proposed to provide guarantees of consumer's privacy protection. The core idea is that a compositional token consisting of the real query and $k-1$ confusing queries is submitted to a public cloud for query services. In this case, an adversary cannot distinguish between the real query and confusing queries. Based on k-anonymity, Kalnis *et al.* [37] proposed a framework supporting range query without the leakage of the query itself. After that, several works [2], [35] have been done to promote the efficiency of the range query. However, there is a noteworthy flow that k-anonymity may failure in several scenarios or specific locations, such as the confusing query is closest to the real query, the confusing query exceeds the valid scope of range query, etc. To overcome such flow, Chow *et al.* [1], [3] proposed a secure framework for range queries with leveraging cloaking techniques. It is an effective generalization of k-anonymity. Unfortunately, anonymity-based approaches suffer from huge transmission overhead, since the cloud must return a huge candidate set including multifold candidate points of confusing queries.

Encryption-Based Approaches. Song *et al.* [10] proposed a secure search scheme in the database community by leveraging stream cipher and symmetric under a weak secure assumption. After that, Boneh *et al.* [38] formalized the security model of searchable encryption and presented a concrete scheme based on bilinear pairing. In [39], the authors presented the first secure scheme supporting range query. Following that, there are several works [3], [5], [7] to improve efficiency and security. Liu *et al.* [3] presented a scheme that supports the nearest neighbor search by leveraging the polygon cloaking area. In [5], Homomorphic encryption is introduced to construct a concrete scheme for secure range queries. Zhu *et al.* [7] presented a scheme that supports polygon range queries by leveraging bilinear pairing-based cross products. Recently, in 2018, Yang *et al.* [40] introduce predicate encryption to achieve a secure range query. Most of these work leverages heavy cryptographic tools and result in low efficiency. The security in these works followed traditional security models, such as IND-CPA, semantic security, etc.

Secure Index-Based Approaches. Many indexes have been proposed for searching on large-scale databases [41]. In a similar miner, the security community began designing secure indexes for various searching types [42]. In [43], Khoshgozaran *et al.* proposed a secure index for kNN searching on geographically encrypted data. It can be facilely migrated for range queries with a weaker assumption of security. After that, many works are proposed. Yiu *et al.* [44], Demertzis *et al.* [6] and Cui *et al.* [45] proposed encrypted indexes based on R-tree by leveraging symmetric encryption, Paillier system, bilinear pairing, etc. However, all these schemes adopted traditional security models. The forward-security model is not investigated so far. Additionally, several schemes suffer multi-round interactions, which will lead to significant degeneration of efficiency.

Forward-Security Consideration. Forward-security is formally proposed by Bost [15]. It is for searchable symmetric encryption and is appropriate for highly dynamic environments as mentioned in Section 1. Since that, various work is proposed to promote the efficiency of searching on encrypted keywords [15], [16], [18], [19], [20], [21]. Bost *et al.* proposed a secure keyword search under the forward-security model and a secure keyword search under the backward security model in [15] and [18], respectively. In [16], Kim *et al.* proposed a dual and secure dictionary for promoting the efficiency of secure keyword searches. After that, there are several works on providing both forward-security and backward privacy [19], [20], [21], where retrieve objects are still keywords. Zuo *et al.* [46] presented a forward-secure range query mechanism on one-dimensional data. It is infeasible to be compatible with geographical data. So far, the range query on geographically encrypted data is not investigated under the forward-security model, which is the essential objective in this paper.

8 Conclusion

Range query on geographically encrypted data is a requisite module to alleviate concerns about data privacy and security for LBS services. We proposed, in this paper, a lightweight and forward-secure scheme for dynamic range query, named LS-RQ, which achieved several tangible advantages. Forward-security, which provides a guarantee such that expired token cannot be legally issued again, was satisfied by formally theoretical proof. LS-RQ schemes were also high-efficient due to two reasons. On one hand, the search time at the client side was at the level of seconds. On the other hand, LS-RQ-M and eLS-RQ-M needed to transfer more candidates than an accurate result from the cloud to the client. But the overhead factor of transferring shifted only from 1.2 to 3.4. It was not expensive and can be acceptable in practice since the accurate result only comprised a few points usually. Due to forward-security and high-efficiency, LS-RQ is a practical scheme of secure range queries.

Acknowledgments

The authors would like to thank the editors, anonymous reviewers, Dr. Xiaofang Xia, and Dr. Yingfan Liu (School of Computer Science and Technology, Xidian University) for their helpful comments on an earlier draft of this article. This work was supported in part by the National Natural Science Foundation of China (No. 61702403, 61702105, 61976168, 61672408, and 61972309), in part by the Key Research and Development Plan of Jiangxi Province (No. 20181ACE50029), in part by the Project funded by China Postdoctoral Science Foundation (No. 2018M633473), in part by the Key Research and Development Plan of Shaanxi Province (No. 2019ZDLGY13-09), in part by the Natural Science Basic Research Program of Shaanxi Province (No. 2019CGXNG-023), in part by the CCF-Huawei Database System Innovation Research Plan (No. CCF-HuaweiDBIR008B), in part by the China 111 Project (No. B16037), and in part by the National Engineering Laboratory of China for Public Safety Risk Perception and Control by Big Data (PSRPC).

References

[1] 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, pp. 1–48, 2009.

[2] M. L. Yiu, G. Ghinita, C. Jensen, and P. Kalnis, "Enabling search services on outsourced private spatial data," *The VLDB J.*, vol. 19, no. 3, pp. 363–384, 2010.

[3] Y. Liu, X. Chen, Z. Li, Z. Li, and R. C.-W. Wong, "An efficient method for privacy preserving location queries," *Front. Comput. Sci.*, vol. 6, no. 4, pp. 409–420, Aug. 2012.

[4] Z. Alavi, L. Zhou, J. Powers, and K. Chen, "RASP-QS: Efficient and confidential query services in the cloud," *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1685–1688, 2014.

[5] R. Gay, P. Méaux, and H. Wee, "Predicate encryption for multi-dimensional range queries from lattices," in *Proc. IACR Int. Workshop Public-Key Cryptography*, 2015, pp. 752–776.

[6] I. Demertzis, S. Papadopoulos, O. Papapetrou, A. Deligiannakis, and M. Garofalakis, "Practical private range search revisited," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 185–198.

[7] H. Zhu, F. Liu, and H. Li, "Efficient and privacy-preserving polygons spatial query framework for location-based services," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 536–545, Apr. 2017.

[8] Y. Yang and M. Ma, "Conjunctive keyword search with designated tester and timing enabled proxy re-encryption function for e-health clouds," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 4, pp. 746–759, Apr. 2016.

[9] Y. Miao, J. Ma, X. Liu, X. Li, Z. Liu, and H. Li, "Practical attribute-based multi-keyword search scheme in mobile crowdsourcing," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 3008–3018, Aug. 2018.

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

[11] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks on secure outsourced databases," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1329–1340.

[12] P. Grubbs, M.-S. Lacharite, B. Minaud, and K. G. Paterson, "Pump up the volume: Practical database reconstruction from volume leakage on range queries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 315–331.

[13] M. Lacharité, B. Minaud, and K. G. Paterson, "Improved reconstruction attacks on encrypted data using range query leakage," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 297–314.

[14] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson, "Learning to reconstruct: Statistical learning theory and encrypted database attacks," Cryptology ePrint Archive, Report 2019/011, 2019. [Online]. Available: https://eprint.iacr.org/2019/011

[15] R. Bost, "$\Sigma o \phi o \varsigma$: Forward secure searchable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1143–1154.

[16] K. S. Kim, M. Kim, D. Lee, J. H. Park, and W.-H. Kim, "Forward secure dynamic searchable symmetric encryption with efficient updates," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1449–1463.

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

[18] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1465–1482.

[19] S.-F. Sun *et al.*, "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 763–780.

[20] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1038–1055.

[21] G. Amjad, S. Kamara, and T. Moataz, "Forward and backward private searchable encryption with SGX," in *Proc. 12th Eur. Workshop Syst. Secur.*, 2019, pp. 4:1–4:6.

[22] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. ACM Symp. Comput. Geometry*, 2004, pp. 253–262.

[23] H.-P. Kriegel, P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering," *ACM Trans. Knowl. Discov. Data*, vol. 3, no. 1, pp. 1–58, 2009.

[24] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.

[25] F. Shen, Y. Yang, L. Liu, W. Liu, and H. T. S. Dacheng Tao, "Asymmetric binary coding for image search," *IEEE Trans. Multimedia*, vol. 19, no. 9, pp. 2022–2032, Sep. 2017.

[26] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 1998, pp. 127–144.

[27] D. Nuñez, I. Agudo, and J. Lopez, "NTRUReEncrypt: An efficient proxy re-encryption scheme based on NTRU," in *Proc. 10th ACM Symp. Inf. Comput. Commun. Secur.*, 2015, pp. 179–189.

[28] D. Nuñez, I. Agudo, and J. Lopez, "Proxy re-encryption: Analysis of constructions and its application to secure access delegation," *J. Netw. Comput. Appl.*, vol. 87, pp. 193–209, 2017.

[29] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Trans. Inf. Syst. Secur.*, vol. 9, no. 1, pp. 1–30, 2006.

[30] Y. Peng, J. Cui, H. Li, and J. Ma, "A reusable and single-interactive model for secure approximate k-nearest neighbor query in cloud," *Inf. Sci.*, vol. 387, pp. 146–164, 2017.

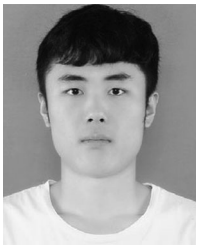
[31] M. L. Fredman, J. Komlos, and E. Szemeredi, "Storing a sparse table with O(1) worst case access time," in *Proc. 23rd Annu. Symp. Found. Comput. Sci.*, 1982, pp. 165–169.

[32] L. Li, R. Lu, and C. Huang, "EPLQ: Efficient privacy-preserving location-based query over outsourced encrypted data," *IEEE Internet Things J.*, vol. 3, no. 2, pp. 206–218, Apr. 2016.

[33] J. Furukawa, "Request-based comparable encryption," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2013, pp. 129–146.

[34] C. Bösch, P. Hartel, W. Jonker, and A. Peter, "A survey of provably secure searchable encryption," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 18:1–18:51, 2014.
[35] K. Vu, R. Zheng, and J. Gao, "Efficient algorithms for K-anonymous location privacy in participatory sensing," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2012, pp. 2399–2407.
[36] L. Sweeney, "k-anonymity: A model for protecting privacy," *Int. J. Uncertainty Fuzziness Knowl.-Based Syst.*, vol. 10, no. 05, pp. 557–570, 2002.
[37] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias, "Preventing location-based identity inference in anonymous spatial queries," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 12, pp. 1719–1733, Dec. 2007.
[38] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 2004, pp. 506–522.
[39] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. Theory Cryptography Conf.*, 2007, pp. 535–554.
[40] W. Yang, Y. Xu, Y. Nie, Y. Shen, and L. Huang, "TRQED: Secure and fast tree-based private range queries over encrypted cloud," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2018, pp. 130–146.
[41] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," *CoRR*, vol. abs/1408.2927, 2014. [Online]. Available: http://arxiv.org/abs/1408.2927
[42] E. Shmueli, R. Waisenberg, Y. Elovici, and E. Gudes, "Designing secure indexes for encrypted databases," in *Proc. 19th Annu. IFIP WG 11.3 Work. Conf. Data Appl. Secur.*, 2005, pp. 54–68.
[43] A. Khoshgozaran and C. Shahabi, "Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy," in *Proc. Int. Symp. Spatial Temporal Databases*, 2007, pp. 239–257.
[44] M. L. Yiu, G. Ghinita, C. S. Jensen, and P. Kalnis, "Outsourcing search services on private spatial data," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2009, pp. 1140–1143.
[45] N. Cui, X. Yang, L. Wang, B. Wang, and J. Li, "Secure range query over encrypted data in outsourced environments," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2018, pp. 112–129.
[46] C. Zuo, S.-F. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, "Dynamic searchable symmetric encryption schemes supporting range queries with forward (and backward) security," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2018, pp. 228–246.

Yanguo Peng (Member, IEEE) received the BSc degree in network engineering from North University of China, Taiyuan, China, in 2009, the MS degree in computer software and theory from Guizhou University, Guiyang, China, in 2012 and the PhD degree in computer systems organization from Xidian University, China, in 2016. Currently he is a full lecturer with the School of Computer Science and Technology, Xidian University, China. His research interests include cloud security, data privacy protection and blockchain.



Long Wang received the BSc degree in computer science and technology from Xidian University, Xi'an, China, in 2017. He is currently working toward the MS degree in the School of Computer Science and Technology, Xidian University, China. His current research interests include searchable encryption, secure searching.

Jiangtao Cui (Member, IEEE) received the MS and PhD degrees both in computer science from Xidian University, Xi'an, China, in 2001 and 2005, respectively. Between 2007 and 2008, he has been with the Data and Knowledge Engineering group working on high-dimensional indexing for large scale image retrieval, in University of Queensland (Australia). He is currently the execute dean and a professor with the School of Computer Science and Technology, Xidian University, China. He has published more than 50 journal and conference papers, including VLDB, SIGMOD, ICDE, the *IEEE Transactions on Knowledge and Data Engineering*, *VLDB J*, *IEEE Transactions on Big Data*, etc. His current research interests include data and knowledge engineering, and high-dimensional indexing. He is a distinguished member and a fellow of CCF and is now committee members of CCF TCDB, CCF TCAPP, CCF TCBC.

Ximeng Liu (Member, IEEE) received the BSc degree in electronic engineering from Xidian University, Xi'an, China, in 2010 and the PhD degree in cryptography from Xidian University, China, in 2015. Currently, he is a full professor with the College of Mathematics and Computer Science, Fuzhou University, China. Also, he is a research fellow with the School of Information System, Singapore Management University, Singapore. He has published more than 100 research articles including the *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Computers*, *IEEE Transactions on Industrial Informatics*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Cloud Computing*. He has awarded "Minjiang Scholars" distinguished professor, "Qishan Scholars" in Fuzhou University, Fuzhou, China, and ACM SIGSAC China Rising Star Award (2018). His research interests include cloud security, applied cryptography and big data security. He served as a leader guest editor for Wireless Communications and Mobile Computing and a member of the ACM, and CCF.

Hui Li (Member, IEEE) received the BEng from the Harbin Institute of Technology, China, and the PhD degree from Nanyang Technological University, Singapore. He is currently a professor with the School of Cyber Engineering, Xidian University, China. His research interests include data mining, knowledge management and discovery, privacy-preserving query and analysis in big data. His work have been published in SIGMOD, VLDB, KDD, the *VLDB J.*, *IEEE Transactions on Knowledge and Data Engineering*, *ACM Transactions on Intelligent Systems and Technology*, ICDE, INFOCOM, CIKM and EDBT. He has been nominated as the Best Paper Award in SIGMOD 2015.

Jianfeng Ma (Member, IEEE) received the BS degree in computer science from Shaanxi Normal University, Xi'an, China, in 1982, the MS degree in computer science from Xidian University, Xi'an, China, in 1992, and the PhD degree in computer science from Xidian University, Xi'an, China, in 1995. Currently he is the directer of the Department of Cyber Engineering and a professor with the School of Cyber Engineering, Xidian University, Xi'an, China. He has published more than 150 journal and conference papers. His research interests include information security, cryptography, and network security.