This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

A privacy-preserving framework for outsourcing location-based services to the cloud

Xiaojie Zhu, Erman Ayday, *Member, IEEE,* and Roman Vitenberg, *Member, IEEE*

Abstract—Thanks to the popularity of mobile devices numerous location-based services (LBS) have emerged. While several privacy-preserving solutions for LBS have been proposed, most of these solutions do not consider the fact that LBS are typically cloud-based nowadays. Outsourcing data and computation to the cloud raises a number of significant challenges related to data confidentiality, user identity and query privacy, fine-grained access control, and query expressiveness. In this work, we propose a privacy-preserving framework for outsourcing LBS to the cloud. The framework supports multi-location queries with fine-grained access control, and search by location attributes, while providing semantic security. In particular, the framework implements a new model that allows the user to govern the trade-off between precision and privacy on a dynamic per-query basis. We also provide a security analysis to show that the proposed scheme preserves privacy in the presence of different threats. We also show the viability of our proposed solution and scalability with the number of locations through an experimental evaluation, using a real-life OpenStreetMap dataset.

Index Terms—database outsourcing, privacy-preserving, efficiency, multi-location, Bloom filter, LBS.

1 Introduction

The ubiquity of mobile devices has brought the popularity of location based services (LBSs). LBSs are used in a broad variety of applications areas: location-based search such as Foursquare, social networks with location sharing in Google Latitude or Facebook, location-aware gaming and entertaining systems, e.g. BotFighters [1] and Ingress [2], or global navigation systems, such as GPS.

In order to mitigate the cost of scaling LBS deployment and data storage, location-based service providers (LBSPs) turn to cloud service providers (CSPs) for help. As a prominent example, Foursquare and Yelp use the cloud services of Amazon.com. However, outsourcing query processing and data to the cloud raises additional privacy and confidentiality concerns because query content and user's location need to be protected from both the LBSP and CSP, in addition to protecting LBS data from the CSP. Since the CSP can observe the search process, it may launch a variety of attacks on input patterns and distribution of values in the query. By doing so, the CSP may threaten the location privacy of the users. Unfortunately, most proposed solutions for privacy-preserving LBSs do not consider outsourcing data to CSP in a privacy-preserving way (see Table 1).

Non-disclosure of user identity and distribution of roles between the CSP and the LBSP also raise a challenge of query expressiveness (e.g., the ability to search by attributes other than location) because the LBSP cannot easily disclose its search index to the CSP. They additionally complicate mechanisms for fine-grained access control (see Table 1). For instance, the CSP can only return a record in response to a client query if the client is granted access to that record. The authorization process should not allow the CSP to establish client's identity, nor should it allow the LBSP or CSP to find a link between the query, identity, and location. Integration of additional mechanisms for fine-grained access control and for privacy-preserving attribute-based search into the existing solutions requires complex encryption methods, which leads to longer query processing times.

In many cases, an LBS client is a company or an organization that is interested in dozens of locations at the same time. For example, Uber handles multiple taxi scheduling requests in the same geographic area in parallel. To this end, Uber queries LBS data from Foursquare, which are outsourced to Amazon.com [12]. In such a scenario, it is advantageous if the system supports multi-location-queries because (a) it is more efficient to scan a search index once for multiple locations than to scan it multiple times for a single location, and (b) it makes it more difficult for the CSP to link returned query results to a location in the query. In particular, it is even possible to hide the number of locations in a query from the CSP.

There is a well-known inherent tradeoff between privacy and relevance/precision of query results. In most solutions, it is possible to control this tradeoff by explicitly setting a limit on the maximum privacy exposure or on another related system parameter. Ideally, it is desirable to let the user control the privacy limit dynamically on a per-query basis (or even on a per-location basis in case of a multi-location query) because various locations may have a different degree of sensitivity for the same user.

In this work, to the best of our knowledge, we propose the first privacy-preserving outsourced LBS system with multi-location queries and per-query privacy limit. We propose a novel query scheme in which the user specifies locations of interest along with a minimum privacy degree

- *X. Zhu and R. Vitenberg are with the Department of Informatics, University of Oslo, Oslo, Norway. E-mail: {xiaojiez, romanvi}@ifi.uio.no*
- *E. Ayday is with the EECS Department, Case Western Reserve University, OH, USA, and Computer Engineering Department, Bilkent University, Ankara, Turkey. E-mail:erman@cs.bilkent.edu.tr*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

TABLE 1: Comparing privacy-preserving location based service works

Approach	query encryption	DB outsourcing	search efficiency	fine-grained access control	per-query privacy	multi-loc query
Spatio-temporal cloaking [3]	No	No	$\perp$	No	No	No
Casper [4]	No	No	$\perp$	No	No	No
Personalized k-anonymity [5]	No	No	$\perp$	No	Yes	No
Mix zone [6]	No	No	$\perp$	No	Yes	No
User-centric LPPM [7]	No	No	$\perp$	No	Yes	No
Geo-indistinguishability [8]	No	No	$\perp$	No	Yes	No
Private query [9]	Yes	No	$O(N)$	No	No	No
Fine [10]	Yes	Yes	$O(N)$	Yes	No	No
EPLQ [11]	Yes	Yes	$O(\log N)$	No	No	No
Proposed solution	Yes	Yes	$O(\log N)$	Yes	Yes	Yes

"No" means not considered
$\perp$ means not described but can be implemented in $O(\log N)$
N denotes the number of locations in the dataset

(expressed as an entropy value), and for each location A, the CSP returns an area B containing A that is sufficiently large to satisfy the constraint on the minimum entropy. Importantly, the CSP cannot infer information about A beyond the fact that it is contained in B. The proposed framework supports search by location attributes in addition to locations themselves. In order to enable efficient search over the encrypted data, the LBSP prepares an auxiliary index structure and transfers it to the CSP, which utilizes it during the search. To construct it, the LBSP builds a hierarchical index, which closely mimics the geographic hierarchy of the locations. Then, each node in the index is replaced by a Bloom filter representing both the location and its attributes. The reason for using a Bloom filter is threefold: (i) a cryptographic hash function makes it hard to recover the data content from the hash result, (ii) a Bloom filter is space efficient which is important when dealing with many locations, and (iii) the size of a Bloom filter is independent of the number of locations in a multi-location query, which in combination with subsequent encryption, makes it difficult for the CSP to establish the number of locations in a query.

In order to hide the searched data and the pattern of the Bloom filter from the CSP, we encrypt the Bloom filter using function-hiding inner product encryption (FHIPE) [13]. The challenge, however, is to allow the CSP to search by the location or location attributes over the encrypted Bloom filter representing both. To this end, we utilize the ability of FHIPE to calculate the number of matching bits. This way, the CSP determines whether a query vector matches an index vector by separately comparing the number of matching bits for the location and for the attributes. Thanks to this design, the CSP can realize the search without learning the distribution of elements in the Bloom filter. The novel realization of searchable privacy-preserving hierarchical index for LBS data is a significant technical contribution of this work.

We analyze location privacy provided by the proposed technique in Section 6. Our analysis shows that the proposed scheme keeps location private from the LBSP under the semi-honest threat model. Furthermore, our solution allows verification of user subscription (i.e., access control) without violating his privacy. For this, we utilize blind signatures to allow the LBSP to sign the query without learning any information about it. We also employ key policy attribute-based encryption (KPABE) to realize fine-grained access control.

We conduct an experimental evaluation using the OpenStreetMap dataset [14] to evaluate the time cost of query signature and generation, as well as the search process.

2 Related work

Table 1 gives a fine-grained comparison between the proposed scheme and prior LBS-related solutions. In this section, we present a taxonomy of the state-of-the-art. LBS privacy protection problem has been studied for many years [15], [16]. However, the existing techniques have been designed for single-location queries, they do not consider location attributes, and outsourcing the LBS to a cloud environment has only been partially addressed. In the following, we briefly discuss the existing work in this domain.

Spatial cloaking: [3] proposes spatial cloaking, in which the exact location is replaced by a cloaking region that also contains the original location. In addition, the cloaking region should include at least k users in order to satisfy the privacy requirement. In [17], [18], [19] a cloaking region is created that includes at least k points sharing the same properties. [3], [4], [5], [20], [21] aim at hiding a user's identity by executing queries with $k-1$ other users. [22], [23] propose using dummy locations to achieve k anonymity. [24] proposes a technique that initially sends a fake location to the server and then incrementally searches the k nearest neighbours of this fake location. In [25] location privacy is quantified over a location obfuscation mechanism while in [7] an optimal location privacy preserving mechanism is proposed based on the former analysis result, considering privacy requirement for users, the adversary's background knowledge, and maximum tolerable service loss. The authors of [26] apply a Bayesian network to evaluate location privacy while considering both geographical and semantical dimensions of privacy. Our approach only focuses on the geographical dimension.

Differential privacy: Differential privacy [27] is a privacy concept mainly used to provide privacy guarantees against inference attacks from statistical databases. It is used in [28] to analyze the synthetic data generation in commute scenario. [29] proposes a differentially private location pattern mining algorithm. Both [8] and [30] are based on an approach of geo-indistinguishability. [8] aims at protecting a user's exact location and uses controlled random noise to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

achieve location obfuscation while [30] additionally considers the balance of the utility and privacy. [31] incorporates temporal correlations in location data based on differential privacy. [32] recommends locations for a user based on the query history while employing probabilistic differential privacy to obfuscate the data. [33] detects social relations by evaluating the correlation of movement trajectories. It achieves ϵ-differential privacy of released trajectories by adding noise following the Laplace distribution.

It may be possible to extend these aforementioned approaches to handle multi-location queries, however this will significantly increase the cost. For instance, assume a scenario in which a user is interested in two locations. If a traditional k-anonymity based technique is used, then $2k$ queries will be sent to the server and $2k$ data items will be retrieved. In the case of differential privacy-based approaches, taking [8] as an example, two queries are necessary and for each query, the retrieved area can be 10 times larger than the area of interest. Other differential privacy-based techniques also have similar overheads.

Searchable encryption: Searchable encryption was introduced in [34]. Its idea is to encrypt the content in such a way that the encrypted content is searchable. Since the first introduction, significant body of research, e.g., [35], [36], [37], [38], has been published. In [35] the authors propose a semantic security framework with the purpose of protecting against adaptive chosen keyword attack. In [38], a highly scalable symmetric encryption method is proposed. [39] aims at realizing similarity search over encrypted data while [40] considers similarity search in combination with outsourcing to the cloud. Although both schemes provide similarity search, they do not support per-query privacy setting. Apart from searchable encryption, the private information retrieval (PIR) technology is utilized to retrieve location indices from a cloud service provider. [41] proposes a retrieving mechanism based on *Ring-Leaning with error (Ring-LWE)* while [9] proposes private query based on *quadratic residuosity assumption (QRA)*. Since the client needs to have knowledge about the plaintext index structure before launching a query in PIR, we cannot use PIR in our proposed scheme where only the LBSP has knowledge about the index structure.

Beyond state-of-the-art: Existing body of work does not consider the privacy of location attributes used in a query, which can also reveal sensitive information. For example, if a user of the popular *Pokemon Go* app reveals location attributes (e.g., PokeStop or Gym), the adversary may infer the user location. Similarly, *Ingress* also reveals users' location as the task is set by the app based on a real location. [10] proposes an access control mechanism for outsourcing LBS data. However, it does not allow users to customize the privacy-sensitivity of the locations. Moreover, efficiency is an issue as all the records have to be sequentially scanned to search for a single location. In a recent work [11], the authors propose a range query mechanism. For improving search efficiency, *ss* tree is built by recursively clustering points into bigger clusters. The time complexity of the search process is improved from N to $logN$. However, the scheme in [11] does not achieve per-query privacy limit, nor does it support access control or location attributes.

In our proposed system, we address all these weaknesses of existing work and propose a privacy-preserving cloud-based LBS that supports fine-grained access control, multi-location queries, and also allows the users to set a different minimum privacy degree for different locations on a per-query basis.

3 PRELIMINARIES

In this section, we briefly present the cryptographic techniques that we utilize in our solution.

Asymmetric bilinear groups: Let G_1 and G_2 be two distinct groups of prime order p and $g_1 \in G_1$ and $g_2 \in G_2$ be the generators of G_1 and G_2, respectively. Let $e : G_1 \times G_2 \rightarrow G_T$ be a function which maps two elements from G_1 and G_2 to a target group G_T of prime order p. The tuple (G_1, G_2, G_T, p, e) is an asymmetric bilinear group if following properties hold:
(a) the group operations in G_1, G_2, G_T are efficiently computable.
(b) the mapping e from G_1, G_2 to G_T is efficiently computable.
(c) the mapping e is non-degenerate: $e(g_1, g_2) \neq 1$.
(d) the mapping e is bilinear: for all $a, b \in \mathbb{Z}_p$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.

In our work, vectors of group elements are often used. Let g^v represent $\{g^{v_1}, \cdots, g^{v_n}\}$ where $\{v_1, \cdots, v_n\} \in \mathbb{Z}_p^n$. The mapping of two vectors of group elements is written as: $e(g^u, g^v) = e(g, g)^{uv} = \Pi_{i=1}^{n} e(g, g)^{u_i v_i}$.

Function-hiding inner product encryption (FHIPE): The concept of FHIPE [13] is based on the correctness of the following: $iv \cdot qv = iv \cdot M \cdot M^{-1} qv$ and $iv \cdot qv = \alpha \cdot iv \cdot qv \cdot \beta / (\alpha \cdot \beta)$, where $\alpha \leftarrow \mathbb{Z}_p^*$ and $\beta \leftarrow \mathbb{Z}_p^*$. M is an invertible matrix. iv and qv are two n dimensional vectors, representing the index and the query in our scheme, respectively. Both the multiplication and division operations are computed in $\mathbb{Z}_p$. The FHIPE is designed to securely calculate the inner product between two vectors D_1, D_2 as follows:

$$f(D_1, D_2, S) = \begin{cases} z & \text{if } \exists\ z \in S, \text{ such that } D_1^z = D_2, \\ \textit{fail} & \text{otherwise.} \end{cases} \quad (1)$$

where S is the set containing all possible inner product results.

Key policy attribute-based encryption (KPABE): KPABE is based on the concepts of user attributes and access policy. *AP* denotes the access policy. It considers the attributes of a user and determines whether the user should be granted access to a data record. Technically, it is a secret sharing structure and if the input can be used to reconstruct the secret, then it is legitimate, otherwise the input is illegitimate. Details can be found in [42]. *MK* represents the master key of attribute based encryption, which is applied to generate decryption key k_c for client c. The technique has the following key steps [42]:
Setup: Given a security parameter that defines the key space, output public parameters *PK* and a master key *MK*.
Encryption: Given a record *m*, set of user attributes *AS*, and public parameters *PK*, output ciphertext *C*.
Key Generation: Given an access policy, master key, attributes and public parameters, output a decryption key k_c.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

Decryption: Given ciphertext C and decryption key k, output plaintext m.

In our work, KPABE is utilized to encrypt each data record in the database, which enables fine grain access control over the database.

Blind signature: The concept of a blind signature is first introduced in [43]. It is a form of digital signature in which the content is blinded before it is signed. The resulting signature can be publicly verified against the original, unblinded messages similarly to a regular signature. According to [44], the following key steps are involved in realizing blind signature:

$KeyGen(1^n)$: Given a security parameter n, output a secret-public key pair (sk, pk).

BS_{phase_1}: Given a public key pk and a message m, output a blinded message M and a random number r, which is used as a parameter in the blinding procedure.

BS_{phase_2}: Given M and a secret key sk, output an intermediate signature.

BS_{phase_3}: Given an intermediate signature and r used in BS_{phase_1}, unblind the message and output the final signature.

BS_{verify}: Given pk, m, and a signature, output 1 if the signature is valid, otherwise 0.

In our work, the client has to send a query to the LBSP for authentication. To prevent the LBSP from learning the query content during the authentication, blind signature is utilized. Specifically, BS_{phase_1} and BS_{phase_3} are conducted in the client, BS_{phase_2} in the LBSP, and BS_{verify} in the CSP.

Bloom filter: A Bloom filter is a bit array. At the beginning, all the values are set to 0. There exists a family of k different hash functions, each function mapping a data item to a position inside the array. Consequently, each data item will be represented by k non-zero bits inside the bit array. Even if a data item has not been mapped to the bit array, there is still a probability for the k corresponding bits to be non-zero because of the other data items represented in the array. Such a situation is called false positive. Let m be the length of the bit array and n be the number of distinct data items mapped to the array. The false positive probability is expressed as $(1 - e^{\frac{-(kn)}{m}})^k$, and it achieves minimum value when $k = \ln 2\frac{m}{n}$.

4 Model

In this section, we describe the system, query, and threat models.

4.1 System model

As shown in Figure 1, the system consists of three entities: client, LBSP, and CSP. Initially, the LBSP collects LBS data and constructs an index. Then LBSP encrypts LBS data and index by KPABE and FHIPE, respectively. After the encryption, both of them are outsourced to the CSP (step 1). The client registers itself with the LBSP, negotiates the service, and gets a certificate that it later uses to issue queries (steps 2 and 3). Every client's query is first encrypted, blinded, and sent to the LBSP for signature (steps 4 and 5). Then, the CSP processes the query and returns the results to the client (steps 6 and 7).

4.2 Access control

The access control scheme is based on the concept of user attributes. In order to be granted a read access to a record in the database, a user needs to possess a certain combination of attributes, as expressed through an access policy. When a user registers with the LBSP, the LBSP assigns the user a subset taken from the universe of user attributes, e.g., based on the paid subscription. We use KPABE (see Section 3) to allow the LBSP to manage the access rights and encrypt data records in order to protect the data from the CSP.

4.3 Query model

Our solution supports a query with three parameters: (i) a list of locations of interest, (ii) a list of desired location attributes, and (iii) a desired privacy degree, which governs the tradeoff between precision, privacy, and performance. Albeit we do not put an artificial cap, the number of locations in the list has a moderate impact on the performance, which we analyze in Section 7.

Prior to deploying the service and outsourcing it to the cloud, the LBSP constructs a hierarchical (tree-based) schema of locations. An illustrated example of such a schema is shown in Figure 2, where the Earth is partitioned into countries, countries are partitioned into states or regions, and states are further partitioned into cities (i.e., leaves of the schema). Each location in the schema is assigned a unique id string. In a query, only leaf locations are allowed, though the privacy degree may result in each leaf location being mapped to a non-leaf location, as explained below.

The depth and granularity of the partitioning significantly affects the balance between performance, privacy, and precision of query results. We explain how the LBSP decides upon the depth and granularity in Section 5.1.

Similar to the locations, the LBSP also generates a schema for location attributes. For example, the users might be interested only in locations that have entertainment or food amenities, or the ones that offer a discount entry on a given day. As we illustrate in Figure 7 later, the attribute schema is also hierarchical: e.g., entertainment may be divided into theme parks and cinema. Note that we support a query with a list of multiple desired attributes. Due to the space limitations, we only show how to implement query matching semantics based on a disjunction of attributes in the list. Support for conjunctive and other semantics is possible as well.

In the proposed solution, the user can control the balance between precision and privacy as follows. We assume

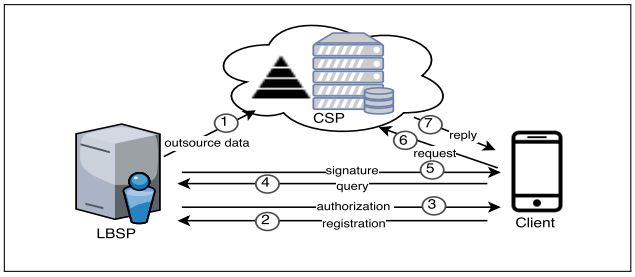

Fig. 1: System model for outsourced LBS.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

TABLE 2: System's core APIs.

API Call	Input	Purpose of the Call	Steps in Fig. 3 and Fig. 4
		For the client	
Registration	attribute set for the client	register the client	3.5
QueryGen	location list, attributes list, privacy degree	generate a query	4.1
TrapdoorGen	query signature, encrypted query vector	generate a trapdoor	4.2, 4.3, 4.6
Decrypt	encrypted results, a set of decryption keys	decrypt retrieved results	4.8
		For the LBSP	
Setup	security parameter	generate parameters	3.2
IndexGen	database	build index over the data	3.3
DBEnc	database	encrypt the data and index	3.4
Authorization	registration request	authorize a client and share private info	3.5, 3.7
Verify	public key	verify the legitimation	4.4
Sign	query	generate the signature	4.5
		For the CSP	
Verify	trapdoor	verify the trapdoor	4.7
Search	trapdoor, index, encrypted data	search over the index and retrieve data	4.7

Note: in the last column, x.y denotes step y in Fig. x.

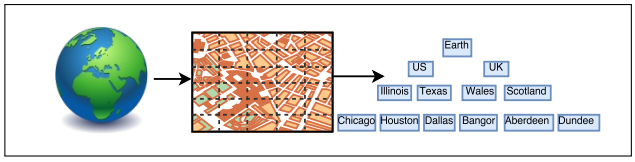

Fig. 2: Example of a hierarchical schema of locations.

that prior to outsourcing the service to the CSP, the LBSP collects information about the number F_i of accesses for each leaf node i in a schema of locations over a period of time. Then, for each non-leaf node i with a set L_i of leaf nodes under i, the LBSP computes access probabilities as follows: $\forall j \in L_i, P_i(j) = F_j / \sum_{jj \in L_i} F_{jj}$. Then, the LBSP calculates the entropy of the access probability distribution for the leaf set L_i as $S_i = -\sum_{j \in L_i} P_i(j) \log_2(P_i(j))$. The entropy values for leaf nodes are zero. Thus, the entropy values range from 0 to the entropy value for the root of the schema of locations. The LBSP uses entropy values for non-leaf nodes to construct the index structure, as explained in Section 5.4.

As part of the query, the client specifies the desired minimum privacy degree pd_j, which is essentially, the minimum entropy value. When performing the search for leaf location j, the CSP will traverse the schema of locations from the root to j and find the lowest node i in the traversal path whose entropy is as least as big as pd_j. Then, the CSP will return either information about i or the entire leaf set L_i, depending on how the system is configured. The sophistication of the traversal algorithm, however, is that the CSP cannot continue the traversal from i to j even if it wants to, and thus, cannot deduce additional information about j beyond the fact that it is a leaf node under i.

This method of controlling the balance between precision and privacy is superior to, e.g., traditional k-anonymity [3], [4], [5] because the latter does not take access frequencies into account. The recent work, e.g., [45], [46] consider access frequency for selecting proper dummy locations to obfuscate the target location. In contrast, dummy locations are not required in our solution and searching is conducted over encrypted data without data confidentiality loss.

The value of the privacy degree also has a moderate impact on the performance because it decreases the length of the traversal path (e.g., rather than traversing all the way to the leaves, the search ends at an intermediate node in the hierarchical schema).

The user sets minimum privacy degree on a per-query basis, which allows for greater configurability with respect to the sensitivity of locations. Note that the client software can apply machine learning algorithms to recommend a value of the privacy degree to the user. This can be done based on the historic data, e.g., about performance, precision, and user satisfaction by the privacy provided.

In summary, the response to a multi-location query includes a matching record for each valid location in the input location list. An input location is valid if the client is granted access to it by the access policy. The matching is subject to constraints on both location attributes and minimum privacy degree. Specifically, a leaf location in the input can be mapped to a non-leaf location in the output by the mechanism of privacy degree matching, as described above.

4.4 Threat model

Our threat model is mostly consistent with other works in this area [38], [11]. The CSP and LBSP are assumed to be honest-but-curious, that is they honestly follow the designed protocol while trying to infer and analyze available data. The LBSP may attempt to analyze the query generated by the client to learn the query content. It may also try to track client activities by correlating anonymized query requests with the information supplied by the client during service registration. The CSP may attempt to analyze not only the submitted query, but also the encrypted data in its storage which includes the index and encrypted database. Note that while the schema of locations is publicly available, the LBSP keeps the information (attribute values and other descriptions) about each location hidden from the CSP. Furthermore, the CSP and LBSP are assumed not to collude in their attempt to gather information about the client. We specifically consider the following threats in our system.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

Tracking threat: As the client may continuously send location queries, the CSP is able to record queries. Based on the analysis of recorded queries, it may attempt to infer approximate locations if CSP is able to learn any significant information from recorded queries.

Linkage threat: The index is encrypted before sent to CSP for protecting query and data privacy, e.g., location, location attributes and privacy degree inside the query. The CSP stores the encrypted index and executes the search operation. Based on these operations, the CSP may attempt to link the query with the retrieved result. In addition, the CSP may also try to infer the query based on the index.

5 Proposed Scheme

The APIs of the client, CSP, and LBS are described in Table 2. We illustrate our implementation of the initialization and query processing in Figures 3 and 4, respectively.

As shown in Figures 3, initialization encompasses system setup and client registration. Prior to deploying the service, the LBSP runs the *setup* to initialize the system and generates the index from the database by calling *IndexGen*. The database is encrypted by calling the *DBEnc* function. Then, the LBSP outsources the encrypted database to the CSP along with the index and setup parameters (see Table 3). The client needs to register with the LBSP in order to be able to get a service from the CSP and send queries. During the registration, the client negotiates client attributes that affect access rights to groups of data records, e.g. all records in New York, or all hotels in a given area. The groups can be overlapping; we use the concept of attribute-based access policy to determine whether the client has permissions to access a record. After the registration, the LBSP assigns a secret decryption key to client (based on the client's attributes) according to the access policy.

The flow of query processing is depicted in Figure 4. During the query generation phase, the client first generates a query and encrypts it by using FHIPE. Then the encrypted query is blinded by using the blind signature scheme and sent to the LBSP for signing, along with the user identity. The LBSP runs *Verify* to establish the legitimacy of the identity. If legitimate, the LBSP signs the blinded query with its private key using the blind signature scheme (previously described in Section 3), otherwise an error message is returned. Upon getting the signature from the LBSP, the client unblinds the query and sends it with the signature to the CSP. The CSP first verifies the legitimacy of the query. If legitimate, the CSP searches the index and returns the matching content. Otherwise, it returns an error message. Upon receiving the retrieved records, the client decrypts them using the decryption key and obtains the result.

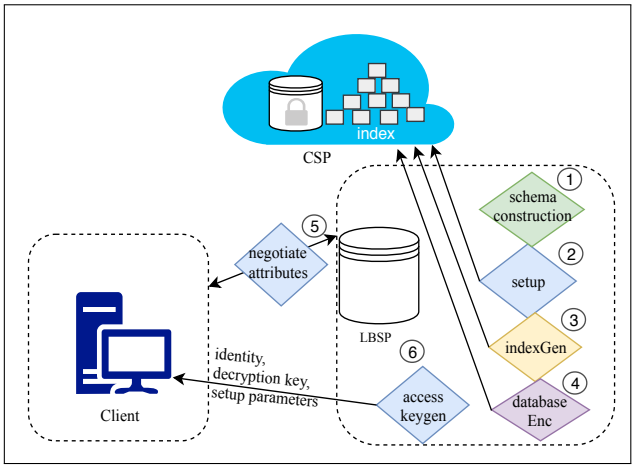

Fig. 3: Initialization

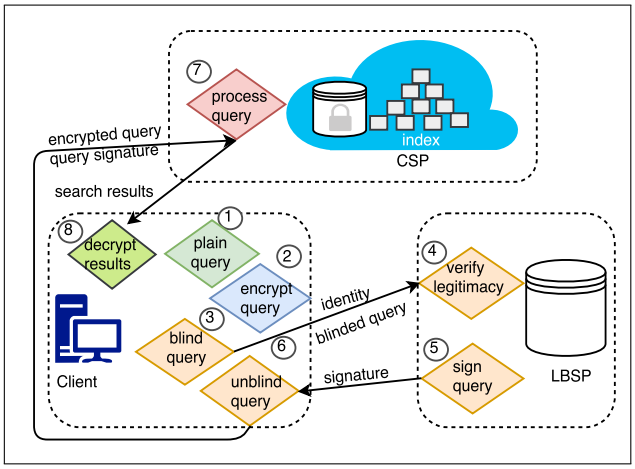

Fig. 4: Query processing

5.1 Constructing a schema for locations

The LBSP needs to provide a schema for locations, which represents a decomposition of the geographical space. In many cases, the schema is induced by the actual geographical hierarchy: a region is partitioned into cities and villages, which are partitioned into neighborhoods and streets, etc. Alternatively, the LBSP can partition the location space in a balanced way. To this end, there is a need to take into account the number of users, access frequency, and the number of attributes for each location. We assume without limiting the generality that the LBSP has an estimate about the number of users and access frequency for each individual location, e.g., thanks to past profiling.

To assess the suitability of different spatial data structures as a basis for the partitioning solution, we consider the analysis in [47]. According to the analysis, B-tree, R-tree, and cell tree structures do not lend themselves to the task: B-tree and R-tree do not result in a disjoint decomposition of space while cell tree relies on a specific data distribution. Since each of the three location characteristics (number of users, access frequency, and number of location attributes) can be considered as a separate spatial dimension, the point region octree [48] is a suitable data structure in which each node is represented as a point in a three-dimensional space and which is suited for arbitrarily distributed data.

Note that the above problem of schema construction needs to be solved only once in an offline fashion. If the values of the parameters significantly change, it may require repartitioning. However, such a need is not expected to arise too often. This implies that the overhead of partitioning is not as important compared, e.g., to the overhead of query processing.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

5.2 Notation

Let G_1 and G_2 be two asymmetric bilinear group, G_T be the target group, e be the mapping operation from G_1 and G_2 to G_T, and p be a big prime as previously described in Section 3. M is an invertible matrix of size $n \times n$, which is generated from a general linear group $GL_n(\mathbb{Z}_p)$. M^* is the adjoint matrix, equal to $det(M)(M^{-1})^T$.

The parameters h_1, h_2, and h_3 are three keyed-hash function families with corresponding keys of k_1, k_2, and k_3. As explained in Section 5.4, our solution uses Bloom filters. Each hash function within h_1 maps one location to a position inside a Bloom filter. Similarly, h_2 and h_3 are respectively used to map attributes and privacy degree into a Bloom filter. The input domain of all the hash functions is a binary string of length λ, where λ is a security parameter, and the output domain is an integer range from 1 to the size of the corresponding Bloom filter.

Note, however, that entropy values are real numbers. In order to transform an entropy value to a string, we first apply a normalization and rounding function F_N, that maps real values to a sufficiently big integer range so that different entropy values will likely be mapped to distinct integers. Then, we represent each integer as a string.

As discussed in Section 3, *AP* denotes the access policy. *MK* represents the master key of attribute based encryption and k_c is the decryption key generated from the master key.

The *BS* is the abbreviation of blind signature and the ith phase of blind signature is denoted as BS_{phase_i}. sk_s is the private key of blind signature and the corresponding public key is pk_s.

A identity for client c is denoted by id_c. id_c is generated by the LBSP. $T = \{t_{\text{loc}}, t_{\text{attr}}, t_{\text{pd}}\}$ is the set of thresholds applied in query processing, detailed in Section 5.8.

5.3 Initialization

Given a database and a set T of thresholds, the LBSP does the space partition (step one in Fig. 3). In the *setup* phase (step two in Fig. 3), the LBSP first selects a big prime p and constructs two distinct asymmetric bilinear group G_1 and G_2 with prime order p. Then the LBSP generates an invertible matrix M and its adjoint matrix M^*. After that, three keyed-hash function families (h_1, h_2, and h_3) with corresponding keys of k_1, k_2, and k_3 are selected. In addition, the LBSP generates a private key sk_s and a public key pk_s for the blind signature mechanism. Once the above parameters have been generated, the LBSP constructs an index (step three in Fig. 3) based on the location schema and encrypts the index and database before outsourcing them to CSP (step four in Fig. 3).

The LBSP registers a client c and assigns it a set (AS_c) of attributes based, e.g., on the paid subscription (step five in Fig. 3). After the negotiation of attributes, the LBSP generates an identity id_c for c. In addition, the LBSP uses access policy *AP*, the universe of the user attributes *UA*, and master key *MK* to generate a decryption key k_c that the client can use to decrypt query results. Finally, the identity and decryption key are sent to the client with the setup parameters (step six in Fig. 3).

The output of the initialization process includes public information $\{p, e, T, pk_s\}$ and private information $\{M^*, k_1, k_2, k_3, k_c, sk_s, MK, AP, UA\}$. Table 3 shows all the parameters used by different parties.

5.4 Index Generation

Given a hierarchical schema of locations, we construct a vector for each location in the schema using the following steps:

Vector construction for location: Each location in the hierarchical schema is mapped into a vector.

Vector construction for location attributes: The attributes of each node in the hierarchical schema are represented by a vector.

Vector construction for location entropy: Following the previous work [49], the privacy of each node in the schema is measured by using entropy.

In the construction of the hierarchical schema of location, the space partition is conducted in such a way that every point is accessed similarly, detailed in Section 5.1. The privacy is encoded by a vector rather than a single number for the sake of efficient privacy-preserving query matching, as explained below.

Thus, the index structure is a hierarchy mimicking the schema for locations, where each location is replaced by an encrypted vector of a fixed size. Figure 5 visualizes the index construction. We now describe each step in detail.

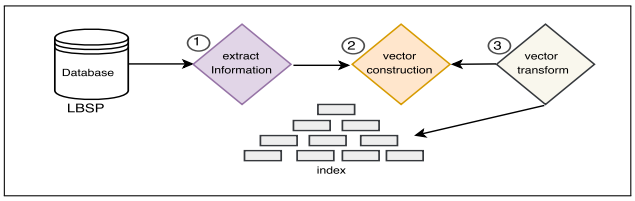

Fig. 5: Process of index construction.

Vector construction for locations: The construction is from the bottom to the top. We start by computing a Bloom filter vector for each of the leaf locations. A vector for a non-leaf location is calculated as a union of the vectors for child locations. We show a construction example in Figure 6.

Since we support multi-location query, a query includes a location list. A vector for the query is constructed by mapping each of the locations in this list to the Bloom filter. A query matches a location node in the index if and only if the inner product of the query and index vectors is greater than a threshold. If the threshold is set too small, there exists too many false positives. However, if the threshold is set too large, false negatives may occur. A practical solution is to set the threshold as the same as the number of hash functions used for mapping locations. Due to the construction, if a query vector matches the vector of a non-root node x in the index, it will necessarily match the vector of the parent node of x.

TABLE 3: Notation used in the implementation(defined in the initialization)

public info	G_1, G_2, G_T, T, p, e pk_s, h_1, h_2, h_3
info privately used by the LBSP	*AP*, sk_s, *MK*, *UA*
info shared by the LBSP and all clients	M^*, k_1, k_2, k_3
private info the LBSP gives to client c	k_c, F_N, AS_c
info locally generated by client c	id_c

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

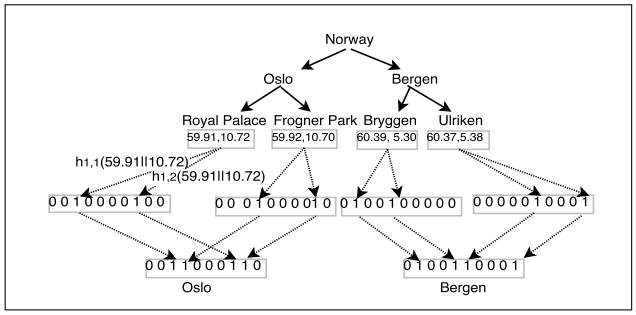

Fig. 6: Location vector construction. 59.91 and 10.72 are the latitude and longitude of the *Royal Palace* in Oslo. A Bloom filter with two hash functions is used to map this location into positions 3 and 8 in the array. Other leaf locations are processed similarly. The vector of the parent node *Oslo* having two children *Royal Palace* and *Frogner Park* is constructed by calculating the union of children's vectors.

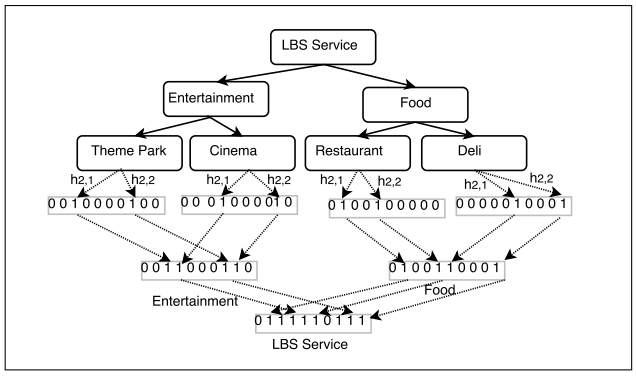

Fig. 7: Vector construction for location attributes. In the example, there are two second-level attributes *entertainment* and *food*. *theme park* and *cinema* belong to the *entertainment* attribute while *restaurant* and *deli* belong to *food*. Based on this structure, vectors are constructed from the bottom to the top.

The size of the Bloom filter represents a tradeoff between precision and efficiency. We use all hash functions from the h_1 hash function family to map values to the Bloom filter. More specifically, the Bloom filter is divided into $|h_1|$ slices and each slice is used for a different hash function in h_1.

Vector construction for location attributes: We utilize a hierarchical schema for attributes, as explained in Section 4.3 and illustrated in Figure 7. The schema is directly derived from the records in the database.

We construct a vector for each node in the attribute schema. The construction is from the bottom to the top, similar to how we construct vectors for locations: first, we use a Bloom filter for the leaf attributes. A vector for a non-leaf node is calculated as a bitwise OR of the vectors for child nodes. Finally, we construct an attribute vector for each location. For example, if the "Frogner park" location in Oslo has a cinema and a restaurant nearby, the attribute vector for "Frogner park" is a bitwise OR of the vectors for cinema and restaurant.

Vector construction for location entropy: The CSP also needs to check, in a privacy-preserving way, if the location

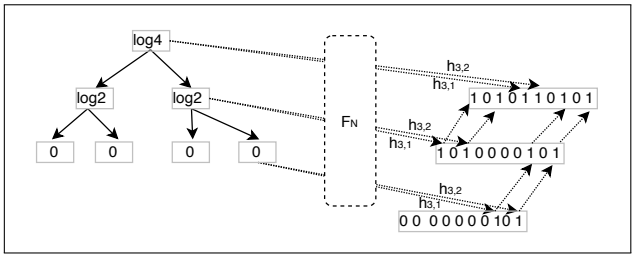

Fig. 8: Vector construction for location entropy. In the example, we assume that all leaf nodes have equal access frequency. There are 3 different location entropy values, 0, $\log 2$ and $\log 4$. The root matches minimum privacy degree of 0, $\log 2$, and $\log 4$, while internal nodes match 0 and $\log 2$. The leaf node only matches the minimum privacy degree of 0. Before the hash function is applied to the entropy values, the values are normalized and rounded using the F_N function.

entropy is at least as high as the minimum privacy degree specified by the user. To this end, the LBSP constructs a privacy vector for each location in the schema. The vector encodes location entropy as follows. First, the LBSP calculates the entropy for each location node as described in Section 4.3. Second, it prepares a sorted list (a sequence) of entropy values from 0 (the value for the leaves) to the value at the root node. Note that the list can be long (order of N entries) if the location hierarchy is unbalanced, yet it is short (order of $\log N$ entries) for balanced trees. Third, the LBSP normalizes all entropy values and rounds them to integers using the F_N function. Finally, for an index location with the (normalized and rounded to an integer) entropy value of e, the LBSP produces a vector for e as follows: it identifies all entropy values on the sorted list which are smaller than e and uses hash functions in h_3 to map these entropy values to the Bloom filter. Figure 8 provides an illustration of vector construction for location entropy, under the simplifying assumption that all leaf nodes have equal access frequency.

The construction for a vector in a query is simpler: given the minimum privacy degree, the client software normalizes it and rounds to an integer using F_N. Then, the client software finds the smallest entropy value on the sorted list that is greater than the privacy degree and maps this value to the Bloom filter, which becomes the privacy degree vector for the query. Note that the LBSP shares both F_N and the sorted list of entropy values with the client.

The requirement of the minimum privacy degree in the query is satisfied if the smallest entropy value on the sorted list that is larger than the privacy degree is present in the list of entropy values for the location. This occurs when the inner product of query and index vectors is equal to $|h_3|$ (as in the vector construction for locations, we divide the Bloom filter into slices, one for each hash function in h_3).

5.5 Database encryption

The database encryption consists of content encryption and index encryption.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

Algorithm 1 *DBEnc*(DB) $\rightarrow C_{DB}$

input AP, MK, and UA
for all $record \in DB$ **do**
    $C_{record} \leftarrow RecordEnc(record, AP, MK, UA)$
    $C_{DB} \leftarrow C_{DB} \cup C_{record}$
return C_{DB}

Content encryption: For content encryption, a KPABE scheme (previously described in Section 3) is applied. The whole process of content encryption is to encrypt each record in the database according to the designed fine-grained access control policy. Details are shown in Algorithm 1, where *RecordEnc* is the encryption procedure of KPABE and AP is the access policy.

Index Encryption: We utilize the FHIPE for the index encryption (previously described in Section 3). Based on the FHIPE, we construct an index encryption algorithm *IndexEnc* as shown in Algorithm 2, where M_i is the i^{th} column of the invertible matrix M (previously set in Section 5.3) and α is a random value from $\mathbb{Z}_q^*$.

Algorithm 2 *IndexEnc*(iv, M) $\rightarrow C_{IV}$

$\alpha \leftarrow \mathbb{Z}_q^*$
for $i \leftarrow 1, n$ **do**
    $C_{iv_i} = g_1^{\alpha \cdot iv \cdot M_i}$
$C_{iv} \leftarrow (C_{iv_1}, \cdots, C_{iv_n})$
return $C_{IV} \leftarrow (g_1^{det(M)\cdot\alpha}, C_{iv})$

5.6 Client registration

In negotiation phase of the initialization, the LBSP negotiates with the client c to decide the attribute set AS_c. After that, LBSP generates a id_c and selects k_c according to access policy based on the attribute set represented by id_c. The ensemble $K = (M^*, k_1, k_2, k_3, k_c)$ is sent to the client, which is setup parameters and described in the Section 5.3. Details are shown in Algorithm 3.

Algorithm 3 *Authorization*(id_c) $\rightarrow K$

$k_c \leftarrow AP(id_c)$
$K \leftarrow (M^*, k_1, k_2, k_3, k_c)$
return K

5.7 Query generation

Input parameters include the location set, location attribute set, and the minimum privacy degree and each of them will be mapped into a vector as discussed in Section 5.4. The BS is a blind signature mechanism achieving ciphertext-indistinguishability under chosen plaintext attack (details introduced in Section 3). All the above operations are detailed in Section 5.4. Then, we call the *QueryEnc* function to encrypt vectors as shown in Algorithm 4 where M_i^* is the i^{th} column of matrix M^* (previously defined in Section 5.3) and β is a random value from $\mathbb{Z}_q^*$. The approach to generate the optimal query for location anonymity is introduced in technical report [50].

Before sending the final query to the LBSP for authorization, client blinds the content of the query (BS_{phase_1})

Algorithm 4 *QueryEnc*(qv, M^*) $\rightarrow C_{QV}$

$\beta \leftarrow \mathbb{Z}_q^*$
for $i \leftarrow 1, n$ **do**
    $C_{qv_i} = g_2^{\beta \cdot qv \cdot M_i^*}$
$C_{qv} \leftarrow (C_{qv_1}, \cdots, C_{qv_n})$
return $C_{QV} \leftarrow (g_2^{\beta}, C_{qv})$

so that the LBSP cannot observe the location of the client. The pseudocode is shown in Algorithm 5, where | denotes bitwise *OR* operation over two vectors.

Algorithm 5 *QueryGen*($locs, attrs, pd$) $\rightarrow (Q, r)$

for all $loc \in locs$ **do**
    for all $h \in h_1$ **do**
        $qv_{loc_i} \leftarrow h(k_1, loc)$
        $qv_{loc} \leftarrow qv_{loc} | qv_{loc_i}$
for all $attr \in attrs$ **do**
    for all $h \in h_2$ **do**
        $qv_{attr_i} \leftarrow h_2(k_2, attr)$
        $qv_{attr} \leftarrow qv_{attr} | qv_{attr_i}$
for all $h \in h_3$ **do**
    $qv_{pd_i} \leftarrow h(k_3, pd)$
    $qv_{pd} \leftarrow qv_{pd} | qv_{pd_i}$
$qv \leftarrow (v_{loc}, v_{attr}, v_{pd})$
$eqv \leftarrow QueryEnc(qv, M^*)$
$(eqv', r) \leftarrow BS_{phase_1}(eqv, pk_s)$
$Q \leftarrow (eqv', id_c)$
return (Q, r)

Upon receiving a query authorization request, the LBSP extracts id_c from query Q and verifies whether id_c is accepted by the key policy AP. If id_c is legitimate, then LBSP signs the blinded query (BS_{phase_2}). Details are shown in Algorithm 6.

Algorithm 6 *Sign*(Q) $\rightarrow \sigma$

extract id_c, eqv' from Q
if $AP(id_c) = True$ **then**
    $\sigma \leftarrow BS_{phase_2}(eqv', sk_s)$
else
    return Not authorized

Upon receiving the result from the LBSP, the client removes the blinding factor from the signature σ returned by the LBSP. The trapdoor contains the stripped signature and *eqv* generated in *QueryGen*. The details are shown in Algorithm 7.

Algorithm 7 *TrapdoorGen*(σ, r, eqv) $\rightarrow Tr$

$\sigma' \leftarrow BS_{phase_3}(pk_s, r, \sigma)$
$Tr \leftarrow (eqv, \sigma')$

5.8 Query processing

Upon receiving the query, the CSP verifies the trapdoor first. If it is valid, it outputs *True*, otherwise *False*. The details are presented in Algorithm 8.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

Algorithm 8 *Verify*$(Tr) \rightarrow$ *True/False*

extract σ', eqv from *Tr*
if $BS_{verify}(\sigma', pk_s, eqv) =$ *True* **then**
    return True
else
    return False

After the verification, the CSP searches the index *Ind* based on the input trapdoor *Tr* and returns matching data items. Algorithm 9 details this function where function *MatchTest* is called to check whether a node matches the query or not. The *MatchTest* is detailed in Algorithm 10, where e denotes a non-degenerate bilinear mapping function and is efficiently computable over groups G_1, G_2 and G_T (defined in Section 5.3).

Algorithm 9 *Search*$(Tr, Ind, C_{DB}) \rightarrow C_{res}$

extract eqv from *Tr*
if *MatchTest*(root of *Ind*, *eqv*) **then**
    add the root of *Ind* onto a stack
while the stack is not empty **do**
    pop a node nd from the stack
    if nd is a leaf **then**
        add nd to *list*
    else
        addedChildren $\leftarrow 0$
        for all child ch of nd **do**
            if *MatchTest*(ch, *eqv*) **then**
                add ch to the stack
                addedChildren $\leftarrow$ *addedChildren* $+ 1$
        if *addedChildren* $= 0$ **then**
            add nd to C_{res}
return C_{res}

Algorithm 10 *MatchTest*$(C_{IV}, C_{QV}) \rightarrow$ *True/False*

extract $g^{det(M)\cdot\alpha}, C_{iv}$ from C_{IV}
extract g^{β}, C_{qv} from C_{QV}
if $\exists z = \{z_1, z_2, z_3\} \in S \wedge \frac{e(C_{iv}, C_{qv})}{e(g^{\alpha}, g^{\beta})^{det(M)\cdot z}} = 1 \wedge z_1 \geq t_{loc} \wedge z_2 \geq t_{attr} \wedge z_3 < t_{pd}$ **then**
    return True
else
    return False

Upon receiving the returned result, the client uses the secret key k_c to decrypt the results.

6 Security Analysis

In this section, based on the threat model in Section 4.4, we first define the leakage function capturing all the information that an adversary is allowed to learn about the query and database. Then, we formally define data privacy and query privacy under chosen plaintext attack model against tracking threat and linkage threat. Finally, we prove that our solution meets the security goals.

6.1 Leakage function

The leakage function is an important part of security analysis as it defines all the information that the adversary is allowed to learn during the interaction with the system. Following the concept proposed in [51] and [37], the leaked information consists of the search pattern, the access pattern, and the size pattern. The size pattern includes the size of the encrypted database, the number of records, the size of the index, the number of entries, and the size of the trapdoor. To define the leakage function formally, we first present the formal definition of the patterns.

Definition 1. Size Pattern τ: Let $C_{DB} = \{C_1, \cdots, C_n\}$, $C_{IV} = \{iv_1, \cdots, iv_m\}$, and *Tr* be the encrypted database, encrypted index and trapdoor respectively, where n is the total number of records in the database and m is the total number of entries in the index. The size pattern is $\tau = \{|C_{DB}|, |C_{IV}|, |Tr|\}$, where $|C_{DB}|$ denotes $\{|C_1|, \cdots, |C_n|\}$ and $|C_{IV}|$ denotes $\{|iv_1|, \cdots, |iv_m|\}$.

Definition 2. Search Pattern ζ: Let $\{t_1, t_2, \cdots, t_q\}$ be the tuple of location, attribute, and privacy degree for q consecutive queries and $\{C_1, \cdots, C_q\}$ be the corresponding retrieved records. Then, ζ is a three dimensional matrix and $\zeta[i, j, k] = 1$ if location loc_i, attribute $attr_j$, and privacy degree pd_k appear at same time in the retrived $C_i (1 \leq i \leq q)$. Otherwise it is zero.

Definition 3. Access Pattern ς: Let C_{IV} be the encrypted index and $\{C_1, \cdots, C_q\}$ be the retrieved records with corresponding trapdoors, $\{Tr_1, \cdots, Tr_q\}$. Then the access pattern is $\varsigma = \{(C_{IV}(Tr_1), C_1), \cdots, (C_{IV}(Tr_q), C_q)\}$.

The leakage function captures the leakage of the above defined patterns and is defined as follows.

Definition 4. Leakage Function $\mathcal{L}$: Let C_{DB}, C_{IV}, Tr be the encrypted database, encrypted index, and trapdoor, respectively. The leakage function is $\mathcal{L} = \{C_{DB}, C_{IV}, Tr, \tau, \zeta, \varsigma\}$.

6.2 Security definitions

Before giving the security definition of the proposed scheme, we first introduce the definition of ciphertext-indistinguishability of chosen plaintext attack [52], which is achieved by our building tools.

Definition 5. A public key encryption scheme $\pi = (Gen, Enc, Dec)$ has ciphertext-indistinguishability under a chosen-plaintext attack if for all probabilistic polynomial-time (*PPT*) adversary A there exits a negligible function $negl$ such that $Pr[PubK^{cpa}_{\mathcal{A},\pi} = 1] \leq \frac{1}{2} + negl(\lambda)$.

The CPA indistinguishability experiment $PubK^{cpa}_{\mathcal{A},\pi}$ is detailed in [52].

The private key encryption scheme with ciphertext-indistinguishability under a chosen-plaintext attack is defined similarly. The only difference is that the adversary can not get the encryption key at the challenging phase.

Then, we formally define our scheme with indistinguishability under Selective-Plaintext Attacks (*IND-SCPA*) [53]. Based on the above information leakage analysis, specifically we define our scheme's security in two aspects. One is data privacy and another is query privacy.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

Data privacy. Informally, the data privacy is defined by first uploading two databases DB_0 and DB_1 to the challenger and the adversary is allowed to send adaptive queries with constraint on the leakage function before making the final decision about which database is utilised. The formal definition is shown as follows.

Let Π=*(Setup, IndexGen, DBEnc, QueryGen, TrapdoorGen, Verify, Search)* be a privacy-preserving outsourcing LBS scheme. For a *PPT* adversary *A*, the advantage function $ADV_A^{\pi}(1^{\lambda})$ is defined as follows: $ADV_A^{\pi}(1^{\lambda}) = Pr(b^* = b) - \frac{1}{2}$, where b^* and b are generated as follows.

Init: The adversary submits two databases DB_0 and DB_1 to the challenger with same number of records and index structure.

Setup: The challenger runs *Setup*(1^{λ}) to generate required parameters and keys.

Phase 1: The adversary adaptively submits requests in one of the following types:

Ciphertext request: The adversary submits a database $DB_j(j > 1)$ and requests one of the encrypt records $C_i(1 \leq i \leq |DB_j|)$.

Trapdoor request: The adversary sends query Q to generate trapdoor Tr with the constraint $\mathcal{L}(Tr, C_{IV_0}, C_{DB_0}) = \mathcal{L}(Tr, C_{IV_1}, C_{DB_1})$.

Challenge: The challenger randomly selects a bit b from the set $\{0, 1\}$ and then invokes the *IndexGen* and $DBEnc$ to build the index C_{IV_b} and encrypted database C_{DB_b}, respectively.

Phase 2: The adversary continues to adaptively send queries to the challenge with the same constraint as described in **Phase 1**.

Guess: The adversary outputs a bit b' as the guess of b.

We say the scheme π is privacy preserving in data privacy under chosen plaintext model if for any PPT adversary A, the advantage function $ADV_A^{\pi}(DB, 1^{\lambda})$ is a negligible function in λ.

Query privacy. The query privacy is defined similarly and the main difference is to submit two queries instead of databases. Detail is shown as follows.

Let Π=*(Setup, IndexGen, DBEnc, QueryGen, TrapdoorGen, Verify, Search)* be a privacy-preserving outsourcing LBS scheme. For a PPT adversary *A*, the advantage function $ADV_A^{\pi}(1^{\lambda})$ is defined as follows: $ADV_A^{\pi}(1^{\lambda}) = Pr(b^* = b) - \frac{1}{2}$, where b^* and b are generated in following way.

Init: The adversary submits two raw queries q_0 and q_1 to the challenger.

Setup: The challenger runs *Setup*(1^{λ}) to generate required parameters and keys.

Phase 1: The adversary adaptively submits requests, which are one of the following types:

Ciphertext request: The adversary submits a database *DB* and requests one of the encrypt records $i(0 \leq i \leq |DB_j|)$. The challenger responses C_i if $\mathcal{L}(q_0, C_{IV}, C_{DB}) = \mathcal{L}(q_1, C_{IV}, C_{DB})$.

Trapdoor request: The adversary sends raw query q to the challenger. The challenger runs $QueryGen$ and then signs the query before running $TrapdoorGen$ to get trapdoor Tr. The challenger sends Tr to the adversary.

Challenge: The challenger randomly selects a bit b from the set $\{0, 1\}$ and then runs the algorithm $QueryGen$ with input q_b, resulting in query Q_b. Then the challenger signs the query and runs *TrapdoorGen* to generate trapdoor Tr_b.

Phase 2: The adversary continues to adaptively send queries to the challenge with the same constraint as described in **Phase 1**.

Guess: The adversary outputs a bit b' as the guess of b.

The scheme π is privacy preserving in query privacy under chosen plaintext model if for any PPT adversary A, the advantage function $\mathrm{ADV}_A^{\pi}(\mathrm{DB}, 1^{\lambda})$ is a negligible function in λ.

6.3 Security proof

In this section, we prove the security of the proposed scheme and show that it provides the data privacy and query privacy. In our scheme, the FHIPE is applied as the encryption algorithm of the index generation $IndexGen$ and query generation $QueryGen$. The blind signature mechanism is embedded into the algorithm $QueryGen$ and $TrapdoorGen$. Specifically, the first step of blinding the query happens in the $QueryGen$ while the step of removing the random scalar is in $TrapdoorGen$. The database is encrypted by the KPABE achieving semantic security under chosen plantext attack model, which is implemented in the algorithm $DBEnc$. Therefore, we can draw following conclusion.

Theorem 1. The proposed privacy-preserving outsourcing LBS scheme achieves data privacy if the FHIPE, the blind signature mechanism, and database encryption algorithm (KPABE) achieve ciphertext-indistinguishability under chosen plaintext attack.

Proof: The proof is based on the simulation of an PPT simulator working as an challenger and demonstrates compromising the proposed scheme is equivalent to break the security of the building tools. Details are shown as follows.

Init: The adversary A selects two database $DB_0 = \{DB_{0,1}, \cdots, DB_{0,n}\}$ and $DB_1 = \{DB_{1,0}, \cdots, DB_{1,n}\}$ and submits them to the challenger.

Setup: The challenger randomly selects parameters from $\mathbb{Z}_q$ and keys from 1^{λ}.

Phase 1: The adversary adaptively generates one of the following requests:

Ciphertext request: The adversary outputs a database $DB_j^*(j \in \{0, 1\})$ and target data record $DB_{j,i}^*$. As a response, the challenger invokes *DBEnc* to encrypt the database DB_j^* and returns the encrypted target data record $C_{j,i}^*$.

Trapdoor request: The adversary outputs a query Q_i^* by running *QueryGen*. Then the adversary sends the query Q_i^* to the challenger. The challenger signs the query by using the private key generated in the *Init* if the query satisfies $\mathcal{L}(Tr(Q_i^*), C_{IV_0}, C_{DB_0}) = \mathcal{L}(Tr(Q_i^*), C_{IV_1}, C_{DB_1})$. Once receiving the signed query, the adversary runs the *TrapdoorGen* and gets the trapdoor Tr_i^*.

Challenge: The challenger randomly selects a bit b from the set {0,1} and then invokes the *IndexGen* and *DBEnc* to build index C_{IV_b} and encrypted database C_{DB_b} respectively.

Phase 2: The adversary continues to adaptively send requests to the challenger as described in the **Phase 1**.

Guess: The adversary outputs a bit b' as the guess of the b.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

The scheme is successfully simulated by a PPT simulator and it shows if an PPT adversary can break the proposed scheme, it must be able to break one of the algorithms among the FHIPE, the blind signature mechanism and database encryption algorithm under chosen plaintext attack model. □

Theorem 2. The proposed privacy-preserving outsourcing LBS scheme achieves query privacy if the FHIPE, the blind signature mechanism and database encryption algorithm achieve ciphertext-indistinguishability under chosen plaintext attack.

The proof is similar to the proof of data privacy. Due to the space limitation, we skip the full proof.

7 Performance

In this section, we present our experimental findings. A complementary theoretical analysis for the search performance can be found in the technical report [50].

The experimental evaluation is performed on a 64-bit Ubuntu system with an Intel i7 processor and 16GB RAM. The open source Charm library [54] is used to implement the pairing group operations, which is supported by the standard *PBC* library [55]. *FLINT* [56] is utilized for the finite field arithmetic in $\mathbb{Z}_q$. We use a real-life dataset taken from OpenStreetMap [14]. Most of our experiments use the dataset for the New York state, which contains 38307 locations of interest. This is in line with the typical dataset size in the state-of-the-art (see, e.g., [11] and [57]). However, in order to explore the scalability of our solution, we also perform experiments with a dataset of bigger size.

The distribution of locations in the NY dataset is shown in Figure 9. In the experiments, we use the given schema for the dataset derived from the geography, rather than solving the optimization problem described in Section 5.1. The hierarchical schema of locations has five levels as shown in Figure 10. Specifically, there are 62 counties, 62 cities, 892 towns, and 498 villages. The average number of locations is around 297 in a city, 22 in a town, and 9 in a village.

Fig. 9: Distribution of locations in the New York state

All three thresholds $t_{loc}, t_{attr}, t_{pd}$ are set to be the same as the number of hash functions used in the Bloom filter, which is 4 in our experiments. The query contains multiple locations (leaf nodes) along with the desired privacy degree. The privacy degree effectively determines at what level in the hierarchical index structure we need to establish a match (see Section 5). As one of the metrics, we measure the time cost of traversing the index to reach a node at a specific level. For the sake of comparison, we ran the experiments with two different capacities of the Bloom filter: 500 and 1000. The minimum false positive probability is set to 0.1. Our experiments combine queries with a different number of locations, namely 1, 100, 500, and 1000. Each experiment is repeated ten times so that the reported values represent averages across these runs.

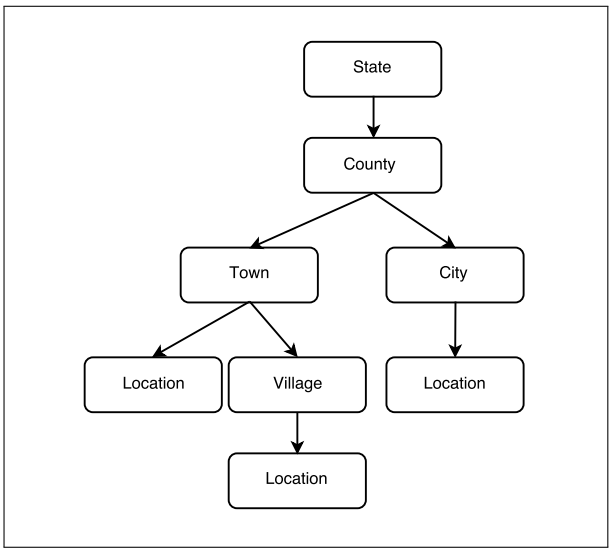

Fig. 10: Hierarchical schema of locations in New York

We use the abbreviation of *BF* to denote a bloom filter in the plots. Figure 11 shows the time cost of mapping locations to a Bloom filter and of data encryption while Figure 12 shows the time cost of the blind signature mechanism. In Figure 11, the left bar is based on the $(500, 0.1)$ Bloom filter while the right bar is based on the $(1000, 0.1)$ Bloom filter. In Figure 12, only the $(500, 0.1)$ Bloom filter is plotted since the $(1000, 0.1)$ Bloom filter results in very similar plots. Each bar consists of four parts, from bottom to top: blinding the query, signing the query, unblinding the query, and verification.

We observe that around 80 percent of the time cost is used for encryption in the query generation and that this cost is only moderately affected by the number of locations. The capacity of the Bloom filter is the most important factor for the time cost of query generation. We also observe that when the number of locations reaches 500, the query time does not increase significantly when increasing the number of locations. This results in an obvious advantage compared with issuing many single-location queries. In addition, the capacity of Bloom filter does not have significant influence on the time cost of blind signature and in the query generation, the percentage of time cost used by blind signature is negligible.

The search efficiency of the proposed scheme is presented in Figures 13 to 18. These plots also show the trade-off between efficiency and privacy. A search latency of an order of ten seconds is a norm in the state-of-the-art when searching encrypted dada (see, e.g., [11] and [58]). Our scheme provides better efficiency compared to this baseline.

Figure 13 shows the time cost of traversing the hierarchical structure to reach the county level (i.e., the second level in the hierarchy). While the time cost increases with

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

the number of locations, we observe that a single multi-location query is still significantly more efficient compared to multiple single-location queries. This trend also holds in the other figures.

Figures 14 and 15 show the time cost of traversing the index to reach a location at the city/town level (i.e., the third level in the hierarchy) for a city and town node respectively. Comparing the figures, we observe that the time cost of traversing the index to reach a node is significantly higher for a town compared to a city. Analyzing the dataset, we find that the total number of towns is significantly larger than the total number of cities, and so is the number of town nodes under an arbitrary county node compared to the number of city nodes. Therefore, it takes more time to scan town nodes under a given county node, which results in longer times required to reach a matching town node. As shown in Figures 16, 17 and 18, the time cost of traversing the hierarchical structure to reach a village node is close to the time required to reach a leaf location under a city node but it is significantly shorter compared to reaching a leaf location under a village node.

One conclusion we can make from Figures 13 to 18 is that the capacity of the Bloom filter does not have a significant effect on the search time at the CSP.

To show the scalability, generality, and consistency of the proposed scheme, we conduct two additional sets of experiments. The first set is for evaluating the scalability of the proposed solution by varying the dataset size and number of input locations. The second set is for proving the generality and consistency of our findings by using a different schema for the same locations and comparing the results.

To show the influence of the dataset size, we compare the results for a small, medium, and large dataset of 3831, 38307, and 383070 locations, respectively. All locations are sampled from OpenStreetMap [14]. In these experiments, we use 100 input locations in a query, which are selected randomly from all the locations in the dataset. The capacity of the Bloom filter used to generate queries is (1000, 0.1). Based on the results in Figure 19, we can observe that the search time grows sub-linearly as the number of locations increases.

To show the influence of the number of input locations, we conduct four additional experiments, where the number of input locations in a query is set to 1, 100, 500, and 1000, respectively. Based on the results in Figure 20, we observe that the search time increases approximately linearly with the number of input locations.

To show the generality and consistency of the proposed solution, experiments are conducted using the same dataset of 38307 locations in the New York state but organized in a different hierarchical schema. In the new schema, the index is built based on the location attributes instead of geography. We have analyzed the dataset and observed that out of the large number of location attributes, there are seven attributes that dominate the dataset, corresponding to almost 50 percent of locations. The seven attributes are *restaurant*, *school*, *park*, *fast food*, *cafe*, *bench*, and *convenience*. The remaining thousands of attributes cover the rest of the locations so that each of them only corresponds to one or two locations. Taking this into account, the new schema essentially consists of eight clusters: The locations corresponding to each of the seven attributes are organized in a separate cluster whereas the rest of the locations are clustered together. The big latter cluster is named *others*. Since there are thousands of attributes in *others*, we build a hierarchical index over the cluster. Specifically, we use k-means clustering algorithm to recursively cluster locations until each node directly above the leaf level contains fewer than 3000 leaf locations, i.e., has < 3000 children.

The resulting schema is shown in Figure 21. There are 18335 locations corresponding to the listed 7 attributes and 19972 locations to *others*. Figure 22 shows the time cost of search for the new schema. The notations used in the figure is explained as follows. The time to reach a leaf under named attributes is denoted as *leaf under named*. The time to reach a leaf under the root of *others* is denoted as *leaf under others*. The time to reach one of the seven named attributes or *others* is denoted as *top level*. The time to reach the *group* is denoted as *group level*. The time to reach the *sub-group* is denoted as *sub-group level*. If we consider the time to search for a leaf location under *others*, we observe that it is almost the same as the time to search for a leaf location under a city when using the geography-induced schema of Figure 10. Therefore, we can conclude that even though the locations are clustered in a different way in the two schemas, the search time used to reach leaf locations (which are at the fourth level in both schemas) is similar. We additionally observe that the time cost of searching for a leaf location is much higher for a leaf location under a named attribute compared to a leaf location under *others*. We conclude that already at a moderate scale of a thousand of leaf locations it is suggested to use a hierarchical index structure in order to improve the performance.

8 Conclusion

In this paper, we present a solution for outsourcing LBS to the cloud in a privacy-preserving fashion. We allow the cloud to perform the search while protecting the privacy of users' queries and identity. We also keep the service data confidential from the cloud provider. Additionally, we support multi-location queries in an efficient way and allow the user to explicitly control the tradeoff between precision and privacy on a per-query basis.

Since the CSP is able to track and record access frequencies, there is a risk that the CSP be able to infer some locations by analyzing the access frequency. For example, a user tends to visit the home and workplace more often compared to other places. In the technical report [50], we analyze the access frequency attack against the proposed scheme and propose a solution to defend against the attack. As future work, the experimental verification will be conducted.

Additionally, we will extend our framework to support k-nearest neighbor search by using locality-sensitive hashing (LSH) to build the Bloom filter instead of standard hash functions. Thanks to locality-sensitive hashing, close locations tend to be hashed into close positions. In that case, the inner product of the query and index can be be used by the CSP to evaluate the "distance". The inner product can be revealed to the CSP without disclosing other information because the inner product can be securely revealed by using

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2019.2892150, IEEE Transactions on Dependable and Secure Computing

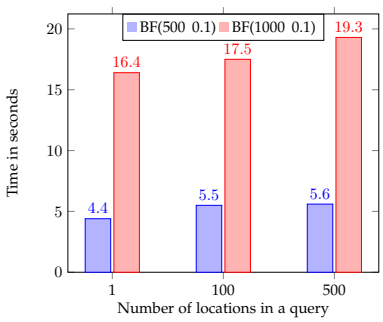

Fig. 11: Mapping and encryption in query generation

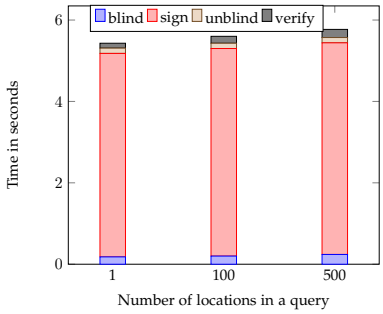

Fig. 12: Blind signatures in query generation

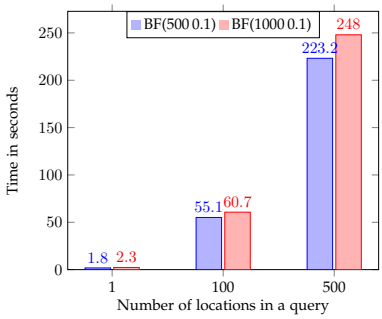

Fig. 13: Time to reach a county node

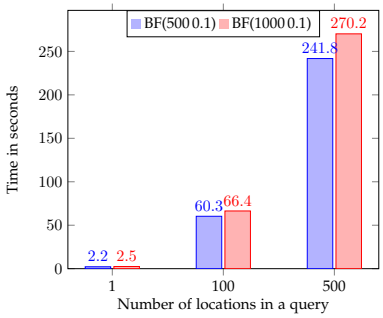

Fig. 14: Time to reach a city node

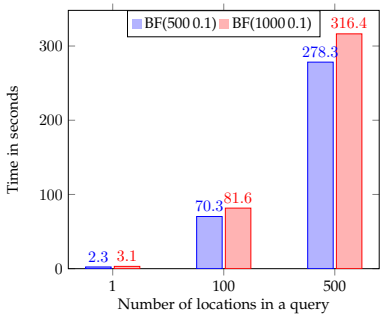

Fig. 15: Time to reach a town node

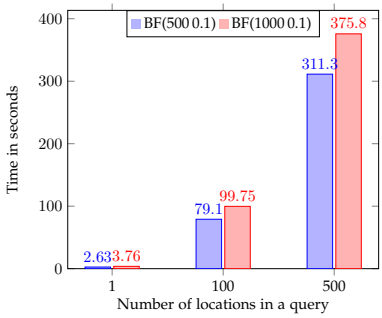

Fig. 16: Time to reach a village node

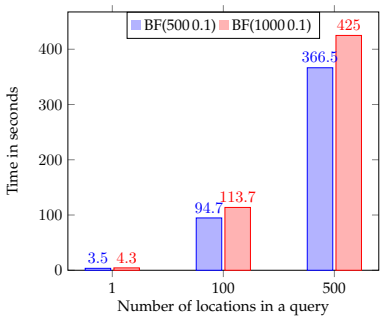

Fig. 17: Reaching a location under a city

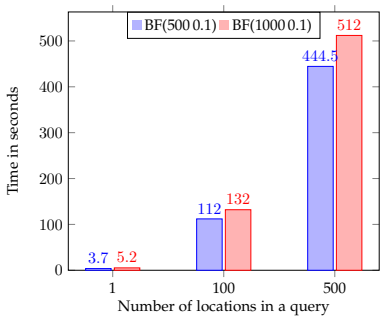

Fig. 18: Reaching a location under a village

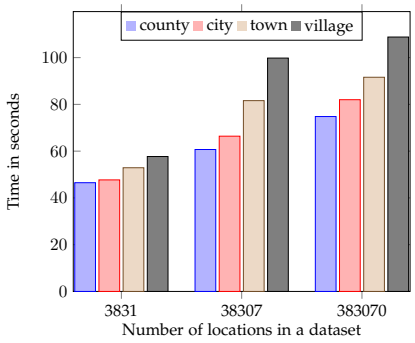

Fig. 19: Comparison of search times for different datasets

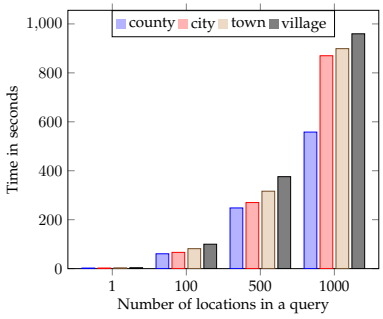

Fig. 20: Search time comparison for a different number of input locations

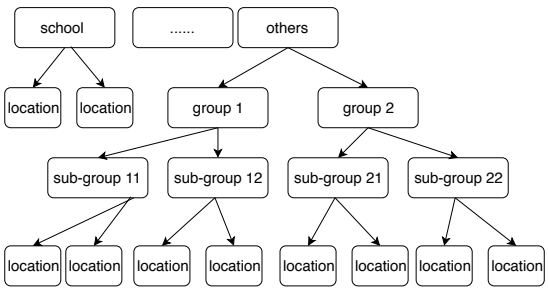

Fig. 21: Attribute-based schema

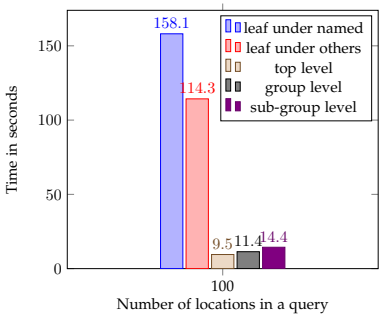

Fig. 22: Time cost of search for the attribute-based schema

the FHIPE encryption algorithm. The k-nearest neighbors are detected as the top k nodes in inner product among all the nodes.

REFERENCES

[1] O. Sotamaa, "All the world's a botfighter stage: Notes on location-based multi-user gaming." in *CGDC Conf.*, 2002.

[2] H. Hodson, "Google's ingress game is a gold mine for augmented reality," *New Scientist*, vol. 216, 2012.

[3] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *Proceedings of the 1st international conference on Mobile systems, applications and services*. ACM, 2003, pp. 31–42.

[4] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new casper: query processing for location services without compromising privacy," in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 763–774.

[5] B. Gedik and L. Liu, "Protecting location privacy with personalized k-anonymity: Architecture and algorithms," *IEEE Transactions on Mobile Computing*, vol. 7, 2008.

[6] A. R. Beresford and F. Stajano, "Location privacy in pervasive computing," *IEEE Pervasive computing*, vol. 2, 2003.

[7] R. Shokri, G. Theodorakopoulos, C. Troncoso, J.-P. Hubaux, and J.-Y. Le Boudec, "Protecting location privacy: optimal strategy against localization attacks," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 617–627.

[8] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: Differential privacy for location-based systems," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 901–914.

[9] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: anonymizers are not necessary," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 121–132.

[10] J. Shao, R. Lu, and X. Lin, "Fine: A fine-grained privacy-preserving location-based service framework for mobile devices," in *INFOCOM*, 2014.

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

[12] M. harkey. (2016) Foursquare + uber! [Online]. Available: https://medium.com/foursquare-direct/foursquare-uber-4e24dc7f829d

[13] S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu, "Function-hiding inner product encryption is practical," Cryptology ePrint Archive, Report 2016/440, 2016. http://eprint. iacr. org, Tech. Rep., 2016.

[14] OpenStreetMap contributors. (2017) Planet dump retrieved from https://planet.osm.org. [Online]. Available: https://www.openstreetmap.org

[15] J. Krumm, "A survey of computational location privacy," *Personal and Ubiquitous Computing*, vol. 13, no. 6, pp. 391–399, 2009.

[16] K. G. Shin, X. Ju, Z. Chen, and X. Hu, "Privacy protection for users of location-based services," *IEEE Wireless Communications*, vol. 19, no. 1, pp. 30–39, 2012.

[17] B. Bamba, L. Liu, P. Pesti, and T. Wang, "Supporting anonymous location queries in mobile environments with privacygrid," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 237–246.

[18] M. Duckham and L. Kulik, "A formal model of obfuscation and negotiation for location privacy," in *International Conference on Pervasive Computing*. Springer, 2005, pp. 152–170.

[19] M. Xue, P. Kalnis, and H. K. Pung, "Location diversity: Enhanced privacy protection in location based services," in *International Symposium on Location-and Context-Awareness*. Springer, 2009, pp. 70–87.

[20] J.-D. Zhang and C.-Y. Chow, "Real: a reciprocal protocol for location privacy in wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 4, pp. 458–471, 2015.

[21] Y. Zhang, W. Tong, and S. Zhong, "On designing satisfaction-ratio-aware truthful incentive mechanisms for k-anonymity location privacy," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 11, pp. 2528–2541, 2016.

[22] H. Kido, Y. Yanagisawa, and T. Satoh, "Protection of location privacy using dummies for location-based services," in *21st International Conference on Data Engineering Workshops (ICDEW'05)*. IEEE, 2005, pp. 1248–1248.

[23] P. Shankar, V. Ganapathy, and L. Iftode, "Privately querying location-based services with sybilquery," in *Proceedings of the 11th international conference on Ubiquitous computing*. ACM, 2009, pp. 31–40.

[24] M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu, "Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services," in *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 2008, pp. 366–375.

[25] R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, and J.-P. Hubaux, "Quantifying location privacy," in *2011 IEEE Symposium on Security and Privacy*. IEEE, 2011, pp. 247–262.

[26] B. Ağır, K. Huguenin, U. Hengartner, and J.-P. Hubaux, "On the privacy implications of location semantics," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 4, pp. 165–183, 2016.

[27] C. Dwork, "Differential privacy," in *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*, ser. ICALP'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1007/11787006_1

[28] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber, "Privacy: Theory meets practice on the map," in *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*. IEEE Computer Society, 2008, pp. 277–286.

[29] S.-S. Ho and S. Ruan, "Differential privacy for location pattern mining," in *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*. ACM, 2011, pp. 17–24.

[30] N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Optimal geo-indistinguishable mechanisms for location privacy," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 251–262.

[31] Y. Xiao and L. Xiong, "Protecting locations with differential privacy under temporal correlations," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1298–1309.

[32] J.-D. Zhang and C.-Y. Chow, "Enabling probabilistic differential privacy protection for location recommendations," *IEEE Transactions on Services Computing*, no. 1, pp. 1–1, 2018.

[33] L. Ou, Z. Qin, S. Liao, Y. Hong, and X. Jia, "Releasing correlated trajectories: Towards high utility and optimal differential privacy," *IEEE Transactions on Dependable and Secure Computing*, 2018.

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

[35] E.-J. Goh *et al.*, "Secure indexes." *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[36] B. Dan, W. Brent, and S. Conjunctive, "Range queries on encrypted data. theory of cryptography. ed. by salil vadhan. vol. 4392," *Lecture Notes in Computer Science. Springer Berlin/Heidel-berg*, pp. 535–554, 2007.

[37] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[38] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 353–373.

[39] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 2012, pp. 1156–1167.

[40] M. L. Yiu, I. Assent, C. S. Jensen, and P. Kalnis, "Outsourced similarity search on metric data assets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 2, pp. 338–352, 2012.

[41] C. Aguilar-Melchor, J. Barrier, L. Fousse, and M.-O. Killijian, "Xpir: Private information retrieval for everyone," *Proceedings on Privacy Enhancing Technologies*, vol. 2016, no. 2, pp. 155–174, 2016.

[42] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*. Acm, 2006, pp. 89–98.

[43] D. Chaum, "Blind signatures for untraceable payments," in *Advances in cryptology*. Springer, 1983, pp. 199–203.

[44] S. Goldwasser and M. Bellare, "Lecture notes on cryptography," 2008.
[45] B. Niu, Q. Li, X. Zhu, G. Cao, and H. Li, "Achieving k-anonymity in privacy-aware location-based services," in *INFOCOM*, 2014.
[46] ——, "Enhancing privacy through caching in location-based services," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 1017–1025.
[47] D. James, B. Trees, and C. S. G. CSG, "Spatial data structures," 2003.
[48] H. Samet, *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.
[49] A. Serjantov and G. Danezis, "Towards an information theoretic metric for anonymity," in *International Workshop on Privacy Enhancing Technologies*. Springer, 2002, pp. 41–53.
[50] X. Zhu, E. Ayday, and R. Vitenberg, "A privacy-preserving framework for outsourcing location-based services to the cloud," University of Oslo, 2018. http://www.tidal-news.org/techreports/framework-Zhu.pdf, Tech. Rep., 2018.
[51] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2010, pp. 577–594.
[52] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.
[53] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems." in *TCC*, vol. 5444. Springer, 2009, pp. 457–473.
[54] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.
[55] B. Lynn *et al.*, "The pairing-based cryptography library. internet: crypto," 2006.
[56] W. Hart, F. Johansson, and S. Pancratz, "FLINT: Fast Library for Number Theory," 2013, version 2.4.0. [Online]. Available: http://flintlib.org
[57] B. Wang, M. Li, and L. Xiong, "Fastgeo: Efficient geometric range queries on encrypted spatial data," *IEEE Transactions on Dependable and Secure Computing*, 2017.
[58] B. Wang and X. Fan, "Search ranges efficiently and compatibly as keywords over encrypted data," *IEEE Transactions on Dependable and Secure Computing*, 2016.

Xiaojie Zhu received M.S. degree in computer science from Chinese Academy of sciences, Beijing, China in 2015. Currently he is a Ph.D candidate in the University of Oslo, Oslo, Norway. His research focuses on the cloud security, data processing and privacy, and cryptography.

Erman Ayday received the MS and Phd degrees from the Georgia Institute of Technology, in 2007 and 2011, respectively. He is an assistant professor with Bilken University, Turkey. Before that, he was a Post-Doctoral Researcher with École Polytechnique Fédérale de Lausanne, Switzerland. His research interests include privacy-enhancing technologies (including big data and genomic privacy), wireless network security, trust and reputation management, and recommender systems.

Roman Vitenberg is a professor at the Department of Informatics, University of Oslo. His research interests lie broadly in the area of distributed applications, middleware and algorithms; including specification, design, analysis, implementation, performance evaluation, and software engineering. In particular, he has been working on large-scale communication, privacy and security, data storage, distributed event-based systems, fault-tolerant distributed computing, and more recently, blockchain. He is an Associate Editor for the EAI Transactions on Cloud Computing and a Steering Committee member for ACM DEBS. He has over 70 publications in peer-reviewed venues and 5 filed patents. His papers were presented best paper awards at ACM/IFIP/USENIX Middleware, ACM SAC, and ACM DEBS conferences.