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 a large number of location-based services (LBS) have emerged. While a large number of 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-grain access control, and query expressiveness. In this work, to the best of our knowledge, we propose the first privacy-preserving outsourced LBS system supporting continuous access control, multi-location queries, and per-query privacy limit. The proposed framework also supports search by location attributes in addition to locations themselves. We 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, 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 user identity, 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-grain 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. For example, k-anonymity mechanisms cannot easily adapt k in a dynamic fashion.

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. Additionally, this is the first LBS solution that considers the access frequency attack. The proposed framework supports search by location attributes in addition to locations themselves. In
In this section, 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] aims at hiding a user’s identity by executing queries with \( k - 1 \) other users. [20], [21] propose using dummy locations to achieve \( k \) anonymity. [22] 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 [23] 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.

**Differential privacy:** Differential privacy [24] is a privacy concept mainly used to provide privacy guarantees against inference attacks from statistical databases. It is used in [25] to analyze the synthetic data generation in commute scenario. [26] proposes a differentially private location pattern mining algorithm. Both [8] and [27] are based on an approach of geo-indistinguishability. [8] aims at protecting a user’s exact location and uses controlled random noise to achieve location obfuscation while [27] additionally considers the balance of the utility and privacy. [28] incorporates temporal correlations in location data based on differential privacy.

Order to enable efficient search over the encrypted data, the LBSP builds an hierarchical index structure to be used by the CSP upon search. The hierarchical index structure 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.

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 [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 function-hiding inner product encryption 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.

We analyze location privacy under threat model 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 privacy-preservation. 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 to realize fine-grain 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 fine-grained comparison between the proposed scheme and state of the art work. In this section, we present the taxolgy of the state of the art work. 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 the location attributes, and outsourcing the LBS to a cloud environment have not been properly addressed. In the following, we briefly discuss the existing work in this domain.

**Searchable encryption:** Searchable encryption was introduced in [29] and its concept is to encrypt the con-

<table>
<thead>
<tr>
<th>Approach</th>
<th>query encryption</th>
<th>DB outsourcing</th>
<th>search efficiency</th>
<th>fine-grain access control</th>
<th>per-query privacy</th>
<th>multi-loc. query</th>
<th>handles access freq. attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spatio-temporal cloaking [3]</td>
<td>No</td>
<td>No</td>
<td>( \perp )</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>–</td>
</tr>
<tr>
<td>Casper [4]</td>
<td>No</td>
<td>No</td>
<td>( \perp )</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>–</td>
</tr>
<tr>
<td>Personalized k-anonymity [5]</td>
<td>No</td>
<td>No</td>
<td>( \perp )</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>–</td>
</tr>
<tr>
<td>Mix zone [6]</td>
<td>No</td>
<td>No</td>
<td>( \perp )</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>–</td>
</tr>
<tr>
<td>User-centric LPPM [7]</td>
<td>No</td>
<td>No</td>
<td>( \perp )</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>–</td>
</tr>
<tr>
<td>Geo-indistinguishability [8]</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>–</td>
</tr>
<tr>
<td>Private query [9]</td>
<td>Yes</td>
<td>No</td>
<td>( O(N) )</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>–</td>
</tr>
<tr>
<td>Fine [10]</td>
<td>Yes</td>
<td>Yes</td>
<td>( O(N) )</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>–</td>
</tr>
<tr>
<td>EPLQ [11]</td>
<td>Yes</td>
<td>Yes</td>
<td>( O(\log N) )</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Proposed solution</td>
<td>Yes</td>
<td>Yes</td>
<td>( O(\log N) )</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

\( \perp \) means not described but can be implemented in \( O(\log N) \)

\( N \) denotes the number of locations in the dataset

---

TABLE 1: Comparing privacy-preserving location based service works
tent and search over the encrypted content. Since the first introduction, massive research work, e.g., [30], [31], [32], [33], has been done. In [30] it formulates a security model for indexes known as semantic security against adaptive chosen keyword attack. In [33], a highly scalable symmetric encryption method is proposed.

However, almost all of the searchable encryption schemes only supports exact matching. Since in the proposed scheme the number of input query is not static, the exact matching mechanism is not suitable. [34] aims at proposing similarity search over encrypted data and [35] considers similarity search over outsourced environment. Although both schemes provide the similarity search, the personal privacy control is ignored.

In addition to the searchable encryption, the private information retrieval (PIR) technology is applied in retrieving encrypted index. [36] proposes a retrieving mechanism based on Ring-Learning with error (Ring-LWE) and [9] proposes private query based on quadratic residuosity assumption (QRA). Since the client needs to know the target index before launching a query in PIR, it can not be applied in the proposed scheme where the client has no knowledge about the target index.

Furthermore, existing 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. The recent work [11] proposes a range query mechanism. For improving its search efficiency, it is built by recursively clustering points into bigger clusters. The time complexity of search process is improved from $logN$ to $\frac{1}{N} log N$. However, in their scheme per-query privacy limit is not available and both access control and location attributes are ignored. In addition, the impact of data access pattern leakage is not considered.

In our proposed system, we address all these weaknesses of existing work and propose a privacy-preserving cloud-based LBS that supports fine-grain 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 tools that we utilize in our solution.

Asymmetric bilinear groups: Let $G_1$ and $G_2$ be two distinct groups of prime order $q$ 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 $q$. The tuple $(G_1, G_2, G_T, g, e)$ 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 $G_1 \times G_2$ 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 $a, b \in Z_q$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.

In our work, vectors of group elements are often used. Let $g^r$ represent $\{g^{v_1}, \cdots, g^{v_n}\}$ where $\{v_1, \cdots, v_n\} \in Z_p$. The mapping of two vectors of group elements is written as: $e(g^{v_1}, g^{v_2}) = e(g, g)^{v_1v_2}$.

Key policy attribute-based encryption: The technique has the following key steps [37]:

Setup: Given a security parameter that defines the key space, output public parameters $PK$ and a master key $MK$.

Encryption: Given a record $m$, a set of attributes $AS$, and public parameters $PK$, output ciphertext $C$.

Key Generation: Given an access structure, master key, attributes and public parameters, output a decryption key.

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

In our work, key policy attribute-based encryption is utilized to encrypt each data record in the database, which enables fine grain access control over the database.

Blind signature: Blind signature is first introduced in [38]. 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 [39], the following key steps are involved in realizing blind signature:

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

$\text{BS}_{\text{phase1}}$: Given a public key $pk$ and a message $m$, output a blinded message $M$ and a random number $r$ used in blinding message.

$\text{BS}_{\text{phase2}}$: Given $M$ and a secret key $sk$, output an intermediate signature.

$\text{BS}_{\text{phase3}}$: Given an intermediate signature and $r$ for unblinding signature, output the final signature.

$\text{BS}_{\text{only}}$: Given $pk$, $m$, and a signature, output 1 if the signature is valid, otherwise 0.

In our work, client has to send a query to the LBSP for authentication. To prevent the LBSP from learning the query content and user identity during the authentication, blind signature is utilized.

Bloom filter: A Bloom filter is a bit array. At the beginning, all the values are set to 0. There exists $k$ different hash functions mapping various data items into the bit array. Specifically, each data item will be represented by $k$ non-zero bits inside the bit array. If a data item does not exist in the bit array, the bit array still gives the positive test result. It is called false positive. Let $m$ be the length of the bit array and $n$ be the number of data items inside the array. The false positive probability, $p = (1 - e^{-\frac{kn}{m}})^k$, achieves minimum value when $k = 2 \frac{m}{n}$.

4 Model

In this section, we describe the system, query, and threat models.
vided into theme parks and cinema. Note that we support
schema is also hierarchical: e.g., entertainment may be di-
a given day. As we illustrate in Figure 7 later, the attribute
be interested only in locations that have entertainment or
decides upon the depth and granularity in Section 5.1.

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 1, where the Earth is parti-
tioned 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.

The depth and granularity of the partitioning signif-
icantly 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 di-
vided into theme parks and cinema. Note that we support
a query with a list of multiple desired attributes. Due to
the space limit, 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.

Finally, the query also includes the desired minimum
privacy degree. Following the previous work [40], entropy
is applied to define the privacy. The entropy value repre-
sents the extent of uncertainty [41]. For any location in the
hierarchical location schema, if there exist $n$ leaves under it,
the entropy value can reach $\log n$. Assuming a total of
$N$ leaves in the schema, the entropy varies between $0$ (the
entropy for each leaf location) and $\log N$ (the entropy for
the root). Thus, the query should only include the locations
whose entropy is at least as high as the desired minimum
privacy degree (that is set by the user). Non-conforming
locations are simply removed from the query, and an error
message is returned. Since the range of privacy degree can
be embedded into the client software, it can perform this
removal locally.

In the proposed solution, the user can control the balance
between precision and privacy as follows: if the user is
interested in a specific location $A$, she can select the parent
or any ancestor of $A$ instead. For example, she can select
a neighborhood instead of a specific attraction. The CSP
cannot decrypt the query, yet it can see which locations
in the index match. By selecting a neighborhood, the user obtains quantifiably better privacy at the expense of reduced
precision. This selection 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 response to a query includes a description for each
location in the query list to which the client is granted
access, whose location attributes match those in the query
and whose entropy is at least as high as the given minimum
privacy degree.

4.3 Threat model
Our threat model is mostly consistent with other works
in this area [33], [11], with one important difference: we
additionally consider a threat of access frequency attack.
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 infor-
mation 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, 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.

Tracking threat: As the client may continuously send
location queries, the CSP is able to record queries. Based on
### TABLE 2: System’s core APIs.

<table>
<thead>
<tr>
<th>API Call</th>
<th>Input</th>
<th>Purpose of the Call</th>
<th>Steps in Fig. 3 and Fig. 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>attribute set for the client</td>
<td>register the client</td>
<td>3.5</td>
</tr>
<tr>
<td>QueryGen</td>
<td>location list, attributes list, privacy degree</td>
<td>generate a query</td>
<td>4.1</td>
</tr>
<tr>
<td>TrapdoorGen</td>
<td>query signature, encrypted query vector</td>
<td>generate a trapdoor</td>
<td>4.2, 4.3, 4.6</td>
</tr>
<tr>
<td>Decrypt</td>
<td>encrypted results, a set of decryption keys</td>
<td>decrypt retrieved results</td>
<td>4.8</td>
</tr>
</tbody>
</table>

For the LBSP

<table>
<thead>
<tr>
<th>API Call</th>
<th>Input</th>
<th>Purpose of the Call</th>
<th>Steps in Fig. 3 and Fig. 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>setup</td>
<td>security parameter</td>
<td>generate parameters</td>
<td>3.2</td>
</tr>
<tr>
<td>IndexGen</td>
<td>database</td>
<td>build index over the data</td>
<td>3.3</td>
</tr>
<tr>
<td>DBEnc</td>
<td>database</td>
<td>encrypt the data and index</td>
<td>3.4</td>
</tr>
<tr>
<td>Authorization</td>
<td>registration request</td>
<td>authorize a client and share private info</td>
<td>3.5, 3.7</td>
</tr>
<tr>
<td>Verify</td>
<td>public key</td>
<td>verify the legitimation</td>
<td>4.4</td>
</tr>
<tr>
<td>Sign</td>
<td>query</td>
<td>generate the signature</td>
<td>4.5</td>
</tr>
</tbody>
</table>

For the CSP

<table>
<thead>
<tr>
<th>API Call</th>
<th>Input</th>
<th>Purpose of the Call</th>
<th>Steps in Fig. 3 and Fig. 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verify</td>
<td>trapdoor</td>
<td>verify the trapdoor</td>
<td>4.7</td>
</tr>
<tr>
<td>Search</td>
<td>trapdoor, index, encrypted data</td>
<td>search over the index and retrieve data</td>
<td>4.7</td>
</tr>
</tbody>
</table>

the analysis of recorded queries, it may infer approximate locations if CSP is able to learn any significant information from recorded queries.

**Linkage threat:** The CSP stores the encrypted index and executes the search operation. The CSP can execute search operation to link the query with the retrieved result. In addition, it is also possible for the CSP inferring the query based on the index.

**Frequency analysis threat:** Since the CSP is able to track and record access frequencies, it might be possible for the CSP to infer some locations by analyzing the access pattern, for example, a user tends to visit the home and work more often compared to other places. The access frequency attack is fully analysed in the Appendix B.

### 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. 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 permissions to get access 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 are using the concept of client attributes and an attribute-based access policy to determine whether the client has permissions to access a record. After the registration, the LBSP assigns the private keys to client according to the access policy.

During the query processing phase, the client first generates a query and encrypts it before sending it for signature along with its public key. The LBSP runs **Verify** to establish the legitimacy of the client and signs the query if it is legitimate, otherwise returning an error message. Upon getting the signature from the LBSP, the client unblinds the query, encrypts it and sends it with signature to the CSP. The CSP first verifies the legitimacy of the query. If it is legitimate, the CSP executes **Search** to search the index and returns the corresponding content. Otherwise, it returns an error message. Upon receiving the retrieved result, the client decrypts it by using the private key provided by the LBSP.

**5.1 Space partition**

The LBSP needs to provide a schema for locations, which represents a hierarchical partition of the geographical space. To perform optimal partitioning, LBSP needs to consider the effect on the performance. We assume that the LBSP has an estimate about the number of users and access frequency for each individual location, e.g., thanks to past profiling.

Based on the history dataset analysis, we consider the following three metrics: (i) access frequency, (ii) number of users, and (iii) number of location attributes. In the schema, each region is represented by a node. The access frequency of a node \( i \) is calculated as the total number of accesses divided by the total time, \( f_i = \frac{F_i}{\delta} \) where \( F_i \) denotes the total number of accesses happened within time period \( \delta \). The number of users is defined as the total number of users...
metrics may have different weights in the construction, we express a single optimization objective combining different metrics as follows:

$$\min \sum_{i=1}^{l_{\text{max}}} \sum_{j \neq i, j=1}^{l_{\text{max}}} (\alpha |u^{i,j} - a^{i,j}| + \beta |f^{i,i,\delta} - f^{i,i,\delta}| + \gamma (|a^{i,j} \setminus a^{i,i}| + |a^{i,j} \setminus a^{i,i}|)), \quad (4)$$

where \(\alpha, \beta, \) and \(\gamma\) are different weights for the number of users, access frequency, and number of location attributes, respectively. All these parameters are set by LBSP according to its preferences.

Note that the optimization problem of partitioning 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.

5.2 Setup

Let \(G_1\) and \(G_2\) be two asymmetric bilinear group as previously described in Section 3. \(M\) is \(n \times n\) dimensional invertible matrix generated from a general linear group \(GL_n(\mathbb{Z}_2)\). \(M^\ast\) 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 \(k_3\). \(h_1\) is used to map location into the vector \(q_{\text{loc}}\) where \(l\) represents existence and \(0\) is the opposite side. Similarly, \(h_2\) and \(h_3\) are used to map attributes and privacy level into vector \(q_{\text{attr}}\) and \(q_{\text{uid}}\) respectively.

\(AP\) is the access structure. Given the client public key \(pk_c\), the LBSP uses \(AP\) and master key \(MK\) to return a decryption key \(k_c\) that the client can use to decrypt query results. \(sk_c\) is a secret key for LBSP to sign query, and \(pk_{c}\) is the corresponding public key. \(T = \{\text{loc}, \text{attr}, \text{uid}\}\) is the set of thresholds.

The output of Setup algorithm includes public information \(\{q, e, T, pk_c\}\) and private information \(\{M^\ast, T, k_1, k_2, k_3, sk_c, MK, k_c\}\).

Table 3 shows all the parameters used by different parties.

5.3 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.

<table>
<thead>
<tr>
<th>TABLE 3: Notation used in the implementation (defined in the Setup)</th>
</tr>
</thead>
<tbody>
<tr>
<td>public info</td>
</tr>
<tr>
<td>info privately used by the LBSP</td>
</tr>
<tr>
<td>info shared by the LBSP and all clients</td>
</tr>
<tr>
<td>private info the LBSP gives to client (c)</td>
</tr>
<tr>
<td>info locally generated by client (c)</td>
</tr>
</tbody>
</table>
Vector construction for location entropy: The privacy of each node in the schema is measured by using entropy. For the privacy of each node can be properly represented by entropy, in the construction of the hierarchical scheme of location, we control the space partition 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](image)

Vector construction for location attributes: We utilize a hierarchical schema for attributes, as explained in Section 4.2 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 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 the logarithm of the number of leaves under this node. Then, it prepares a sorted list (a sequence) of entropy values from 0 (the value for the root) to \( \log N \) (the value for the leaves) for balanced trees. Then, for an index location with the entropy value of \( e \), the LBSP produces a vector for \( e \) as follows: it uses hash functions in \( h_3 \) to map all entropy values on the sorted list which are smaller than \( e \) to the Bloom filter. Figure 8 provides an illustration of vector construction for location entropy.

![Fig. 6: Location vector construction](image)

![Fig. 7: Vector construction for location attributes](image)
the query.

The requirement of the minimum privacy degree in the query is satisfied if the smallest entropy value on the sorted list 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 \( |h_3| \) (again, we divide the Bloom filter into slices, one for each hash function in \( h_3 \)).

Fig. 8: Vector construction for location entropy. In the example, 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.

5.4 Database encryption

**Content encryption:** For content encryption, a key-policy attribute-based symmetric encryption algorithm is applied. It achieves ciphertext-indistinguishability under chosen plaintext attack. The whole process of content encryption includes two steps. First, the encryption mode (RecordEnc) is chosen (e.g., AES based CTR mode [44]). Second, setting an attribute based key-policy (previously described in Section 3) for each record, the LBSP uses the policy based symmetric key generated from master key \( MK \) to encrypt the corresponding record in database \( DB \). After the encryption, the database is sent to the CSP. Details are shown in Algorithm 1.

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

1. \( \text{for all record } \in DB \text{ do} \)
2. \( \text{generate a key } k \text{ with input } AP, MK, \text{record} \)
3. \( C_{record} \leftarrow \text{RecordEnc} (\text{record}, k) \)
4. \( C_{DB} \leftarrow C_{DB} \cup C_{record} \)
5. \( \text{return } C_{DB} \)

**Index Encryption:** We utilize the function-hiding inner product encryption [13] for the index encryption. The concept of encryption algorithm is based on the correctness of the following: \( iv \cdot qv = iv \cdot M \cdot M^{-1}qv \) and \( iv \cdot qv = \alpha \cdot iv \cdot qv / \beta (\alpha / \beta) \), where \( \alpha \leftarrow Z_q^{*} \) and \( \beta \leftarrow Z_q^{*} \). \( M \) is an Invertible matrix (as introduced in Section 5.2). \( iv \) and \( qv \) are two \( n \) dimensional vectors, representing the index and the query, respectively. Both the multiplication and division operations are computed in the field \( Z_q \). The function-hiding inner product encryption [13] is designed to securely calculate the inner product between two vectors \( D_1, D_2 \) as follows:

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

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

Based on the work [13], we construct an index encryption algorithm IndexEnc as shown in Algorithm 2, where \( M_i \) is the \( i \)th column of matrix \( M \) and \( \alpha \) is a random value from \( Z_q^{*} \).

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

1. \( \alpha \leftarrow Z_q^{*} \)
2. \( \text{for } i \leftarrow 1, n \text{ do} \)
3. \( C_{iv} \leftarrow g_{iv} \cdot \text{iv} \cdot M_i \)
4. \( C_{iv} \leftarrow (C_{iv_1}, \ldots, C_{iv_n}) \)
5. \( \text{return } C_{IV} \leftarrow (g_{iv} \cdot \det(M), \alpha, C_{iv}) \)

5.5 Client registration

In the initial phase, the client gets the attribute set \( AS_c \) from the LBSP. Based on the attribute set \( AS_c \), client \( c \) generates its identity \( pk_c \), (e.g., \( pk_c = \{g^{\text{attr}_1}, \ldots, g^{\text{attr}_n}\} \)). The client sends the identity to the LBSP as a registration request.

LBSP first verifies \( pk_c \). After that, LBSP selects \( k_c \) according to access policy based on the input of attribute set represented by \( pk_c \). The ensemble \( K = (M^*, k_1, k_2, k_3, k_c) \) is sent to the client, which is described in the Section 5. Details are shown in Algorithm 3.

Algorithm 3 Authorization(pk_c) \( \rightarrow K \)

1. \( k_c \leftarrow AP(pk_c) \)
2. \( K \leftarrow (M^*, k_1, k_2, k_3, k_c) \)
3. \( \text{return } K \)

5.6 Query generation

Input parameters include the location set, location attribute set, and the minimum privacy degree. The BS is a blind signature mechanism achieving ciphertext-indistinguishability under chosen plaintext attack (details introduced in Section 3). Each location in the location set is mapped into a vector. Similarly, attributes in the attribute set are mapped into a vector. The privacy degree is also mapped into a vector. All the above operations are detailed in Section 5.3. Then, we call the QueryEnc function to encrypt vectors as shown in Algorithm 4 where \( M^*_i \) is the \( i \)th column of matrix \( M^* \) and \( \beta \) is a random value from \( Z_q^{*} \).

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

1. \( \beta \leftarrow Z_q^{*} \)
2. \( \text{for } i \leftarrow 1, n \text{ do} \)
3. \( C_{qv_i} \leftarrow g_{qv} \cdot M^*_i \)
4. \( C_{qv} \leftarrow (C_{qv_1}, \ldots, C_{qv_n}) \)
5. \( \text{return } C_{QV} \leftarrow (g_{qv} \cdot \beta, C_{qv}) \)

Before sending the final query to the LBSP for authorization, client blinds the content of the query (BSphase.)
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($\text{locs, attrs, pd}) \rightarrow (Q, r)$
1: for all $loc \in \text{locs}$ do
2: for all $h \in h_1$ do
3: $qv_{loc} \leftarrow h(k_1, loc)$
4: $qv_{attr} \leftarrow qv_{loc}(qv_{loc})$
5: for all $attr \in \text{attrs}$ do
6: for all $h \in h_2$ do
7: $qv_{attr} \leftarrow h(k_2, attr)$
8: $qv_{attr} \leftarrow qv_{attr} \lor qv_{attr}$
9: for all $h \in h_3$ do
10: $qv_{pd} \leftarrow h(k_3, pd)$
11: $qv_{pd} \leftarrow qv_{pd} \lor qv_{pd}$
12: $qv \leftarrow (v_{loc}, v_{attr}, v_{pd})$
13: $eqv \leftarrow \text{QueryEnc}(qv, M)$
14: $(eqv', r) \leftarrow \text{BS}_{phase_2}(eqv, pk_1)$
15: $Q \leftarrow (eqv', pk_1)$
16: return $(Q, r)$

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

Algorithm 6 Sign($Q) \rightarrow \sigma$
1: extract $pk_c, eqv'$ from $Q$
2: if $AP(pk_c) = \text{True}$ then
3: $\sigma \leftarrow BS_{phase_2}(eqv', sk_1)$
4: else
5: return Not authorized

Upon receiving the result from the LBSP, the client removes the blinding factor from the signature $\sigma$ 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($\sigma, r, eqv') \rightarrow Tr$
1: $\sigma' \leftarrow BS_{phase_3}(pk_3, r, \sigma)$
2: $Tr \leftarrow (eqv, \sigma')$

5.7 Query processing

Once 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.

Algorithm 8 Verify($Tr) \rightarrow \text{True/False}$
1: extract $\sigma', eqv$ from $Tr$
2: if $BS_{verify}(\sigma', pk_3, eqv) = \text{True}$ then
3: return True
4: else
5: 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 5.2).

Algorithm 9 Search($Tr, Ind, C_{DB}) \rightarrow C_{res}$
1: extract $eqv$ from $Tr$
2: if $\text{MatchTest}(\text{root of } Ind, eqv)$ then
3: add the root of $Ind$ onto a stack
4: while the stack is not empty do
5: pop a node $nd$ from the stack
6: if $nd$ is a leaf then
7: add $nd$ to list
8: else
9: $\text{addChildren} \leftarrow 0$
10: for all child $ch$ of $nd$ do
11: if $\text{MatchTest}(ch, eqv)$ then
12: add $ch$ to the stack
13: $\text{addChildren} \leftarrow \text{addChildren} + 1$
14: if $\text{addChildren} = 0$ then
15: add $nd$ to $C_{res}$
16: return $C_{res}$

Algorithm 10 MatchTest($C_{IV}, C_{QV}) \rightarrow \text{True/False}$
1: extract $g^{det(M)} \cdot C_{IV}$ from $C_{IV}$
2: extract $g^3, C_{QV}$ from $C_{QV}$
3: if $\exists z_i = \{z_1, z_2, z_3\} \in S \land z_i \cdot (C_{IV}, C_{QV}) = 1 \land z_1 \geq thr_{loc} \land z_2 \geq thr_{attr} \land z_3 \geq thr_{pd}$ then
4: return True
5: else
6: return False

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

6 Security analysis

In this section, based on the threat model in Section 4.3, 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 satisfies security requirement.

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 [45] and [32], 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 $\tau$: Let $C_{DB} = \{C_1, \ldots, C_n\}$, $C_{IV} = \{iv_1, \ldots, iv_n\}$ 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 $\{|C_1|, \ldots, |C_n|\}$ and $|C_{IV}|$ denotes $\{|iv_1|, \ldots, |iv_n|\}$.

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

**Definition 3.** Access Pattern $\psi$: Let $C_{IV}$ be the encrypted index and $\{C_1, \ldots, C_q\}$ be the retrieved records with corresponding trapdoors, $\{Tr_1, \ldots, Tr_q\}$, then the access pattern is $\psi = \{C_{IV}, Tr_1, \ldots, C_{IV}, Tr_q\}$.

The leakage function captures above information leakage 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 $\mathcal{L} = \{C_{DB}, C_{IV}, Tr, \tau, \zeta, \psi\}$.

The search pattern and access pattern can be protect by technology Oblivious RAM and Private Information Retrieval but both of them are too inefficient to be applied in practical applications.

### 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 [46], which is achieved by our building tools.

**Definition 5.** An 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 $Pr[PubK_{A,\pi}^{\text{ciphertext}} = 1] \leq \frac{1}{2} + negl(\lambda)$.

The CPA indistinguishability experiment $PubK_{A,\pi}^{\text{ciphertext}}$ is detailed in [46].

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) [47]. Based on the above information leakage analysis, specifically we define our scheme security in two aspects. One is from the data privacy and another is from the query 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 query privacy is defined similarly and the main difference is to submit two queries instead of databases.

- **Data privacy.**
  Let $\Pi=(Setup, IndexGen, DBEnc, QueryGen, TrapdoorGen, Verify, Search)$ be a privacy-preserving outsourcing LBS scheme. For a PPT adversary $A$, the advantage function $Adv_{\Pi}^{\text{data}}(\lambda)$ is defined as follows: $Adv_{\Pi}^{\text{data}}(\lambda) = Pr[\pi^* = b] - \frac{1}{2}$, where $b^*$ and $b$ are generated in following way.
  - **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$(\lambda)$ 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 $DB_i (j > 1)$ and requests one of the encrypt records $C_i (1 \leq i \leq |DB_j|)$.
    - **Trapdoor request**: The adversary sends query $Q$ to the challenger for a trapdoor $Tr$ with the constraint $\mathcal{L}(Tr, C_{IV}, C_{DB}) = \mathcal{L}(Tr, C_{IV}, C_{DB})$.
  - **Challenge**: The challenger randomly selects a bit $b$ from the set $\{0, 1\}$ and then invokes the IndexGen and DBEnc to build the index $C_{IV}$ and encrypted database $C_{DB}$.
  - **Phase 2**: The adversary continues to adaptively send queries to the challenger 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 $\pi$ is privacy preserving in data privacy under chosen plaintext model if for any PPT adversary $A$, the advantage function $Adv_{\Pi}^{\text{data}}(DB, \lambda)$ is a negligible function in $\lambda$.

- **Query privacy.**
  Let $\Pi=(Setup, IndexGen, DBEnc, QueryGen, TrapdoorGen, Verify, Search)$ be a privacy-preserving outsourcing LBS scheme. For a PPT adversary $A$, the advantage function $Adv_{\Pi}^{\text{query}}(\lambda)$ is defined as follows: $Adv_{\Pi}^{\text{query}}(\lambda) = Pr[\pi^* = b] - \frac{1}{2}$, 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$(\lambda)$ 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 $i(0 \leq i \leq |DB|)$. The challenger responds $C_i$ 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 $Tr_i$.

- **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 $\pi$ is privacy preserving in query privacy under chosen plaintext model if for any PPT adversary $A$, the advantage function $ADV_\pi^\pi(DB, 1^\lambda)$ is a negligible function in $\lambda$.

### 6.3 Security proof

In this section, we prove the security of the proposed scheme achieving the data privacy and query privacy. In our scheme, the function-hiding inner product encryption algorithm is applied as the encryption algorithm of the index generation $\text{IndexGen}$ and query generation $\text{QueryGen}$. The blind signature mechanism is embedded into the algorithm $\text{QueryGen}$ and $\text{TrapdoorGen}$. Specifically, the first step of blinding the query happens in the $\text{QueryGen}$ while the step of removing the random scalar is in $\text{TrapdoorGen}$. The database is encrypted by an encryption algorithm achieving semantic security under chosen plaintext model, which is implemented in the algorithm $\text{DBEnc}$.

From above analysis, it is easy to find that the function-hiding inner product encryption algorithm, blind signature mechanism and database encryption algorithm are the basic tools in our scheme. Therefore, we can draw following conclusion.

**Theorem 1.** The proposed privacy-preserving outsourcing LBS scheme achieves data privacy if the function-hiding inner product encryption algorithm, the blind signature mechanism and database encryption algorithm 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 $1^\lambda$.

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

  - **Ciphertext request**: The adversary outputs a database $DB_j^j(j > 1)$. For each record $i$, the adversary generate a query $Q$ by running $\text{QueryGen}$ without the step of blinding. As a response, the challenger invokes $\text{IndexGen}$ to build index $C_{IV,i}$ and $\text{DBEnc}$ to encrypt the database $DB_j^j$. The challenger executes $\text{Search}(Tr_{iv,j}, C_{IV,i}, DB_j^j)$ and returns the results, $C_{iv,i}$.

  - **Trapdoor request**: The adversary outputs a query $Q^*_i$ by running $\text{QueryGen}$. Then the adversary sends the query $Q^*_i$ to the challenger. The challenger signs the query by using the private key generated in the $\text{Init}$ if the query satisfies $\mathcal{L}(Tr(Q^*_i), C_{IV,i}, C_{DB,i}) = \mathcal{L}(Tr(Q^*_i), C_{IV,i}, C_{DB,i})$. Once receiving the signed query, the adversary runs the $\text{TrapdoorGen}$ and gets the trapdoor $Tr_i^*$. The step of removing the random scalar is in $\text{Phase 1}$.

  - **Challenge**: The challenger randomly selects a bit $b$ from the set $\{0, 1\}$ and then invokes the $\text{IndexGen}$ and $\text{DBEnc}$ to build index $C_{IV,i}$ 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 $b$.

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 function-hiding inner product encryption algorithm, 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 function-hiding inner product encryption algorithm, the blind signature mechanism and database encryption algorithm achieve ciphertext-indistinguishability under chosen plaintext attack.

**Proof:** The proof is similar to the proof of data privacy. Due to the space limit, we skip the proof.

### 7 Performance

The evaluation is performed on a 64-bit Ubuntu system with an Intel i7 processor and 16GB RAM. The open source Charm library [48] is applied to implement the pairing group operations, which is supported by the standard PBC library [49] and FLINT [50] is applied for the finite filed arithmetic in $\mathbb{Z}_q$. We use a real-life dataset OpenStreetMap [14] focusing on the New York state that contains 38307 locations of interest. The distribution of locations is shown in Figure 9. In the experiment, we use the given schema for the dataset, 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, 63 cities, 893 towns, 499 villages and the average number of location is 292 in each city, 22 in each town, 9 in each village.

---

**Fig. 9:** Distribution of locations in New York

All three thresholds $I_{loc}, I_{attr}, I_{pd}$ 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) and the desired privacy level. The privacy level 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 bits. The minimum false positive probability is set to 0.1. All our experiments combine queries with different number of locations, namely 1, 500, and 1000. Each experiment is repeated ten times and we report the average result.

![Hierarchical schema of locations in New York](image)

Fig. 10: Hierarchical schema of locations in New York

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 12, for each pair of bars, the left bar is based on the bloom filter (500, 0.1) and the right bar is based on Bloom filter (1000, 0.1). Each bar consists of four parts, from top to bottom, 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 data (see, e.g., [11] and [51]). 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 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. In addition, there is no storage and communication overhead in the client side.

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.

As future work, 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 CSP may use the inner product of the query and index to evaluate the “distance” without getting other information because the inner product can be securely revealed by using function-hiding inner product encryption [13]. The $k$-nearest neighbors are the top $k$ nodes in inner product among all the nodes.

References

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


APPENDIX A

SEARCH EFFICIENCY ANALYSIS

Here, we briefly discuss the efficiency of query generation and search. Let $N$ be the total number of records and $n$ the dimension of the query vector which is decided by the size of the bloom filter.

During the query generation process, the main time complexity is due to function-hiding inner product encryption. It includes multiplying a vector by a $n \times n$ matrix and $n$ exponentiations. Therefore, the time complexity of the query generation is $O(n^2)$ using by Fourier transform.

The search procedure can be divided into three parts. The first part is to traverse the tree from the root to the target nodes. The second part is to check whether the current node matches the condition, which requires $n$ pairing and one comparison operations. The third part is to retrieve the encrypted information. The third part only needs one operation. Thus the entire procedure has the average search time complexity of $O(\log N \cdot n)$.

In real life, the $n$ and $N$ are set based on the context. If the precision requirement is high, the $n$ could be large, even close to $N$. Generally, $n$ and $N$ could be set as $n = \frac{1}{\epsilon^2} \cdot N$ where $\epsilon \leftarrow Z^\ast$. If $c = 1$, then the false positive is $1 - \frac{1}{2} \cdot \frac{1}{\sqrt{N}}$.

APPENDIX B

ACCESS FREQUENCY ATTACK

All the encrypted data and index are stored at the CSP, which provides the CSP an opportunity to analyze the access frequency. The access frequency can be considered...
in two ways: (i) index node access frequency and (ii) data item access frequency.

**Index node access frequency:** Upon receiving the trapdoor generated for the query, the CSP searches the index. Using received queries, the CSP can compute the access frequency of each index node, and hence build a mapping between the received trapdoor and corresponding search path. As a result, the CSP may learn the popularity of the search path and the index nodes, which gives it a chance to predict the possible path and target index node. For example, given a trapdoor, the CSP can utilize its knowledge to predict the most popular path with a high probability. It is also possible to use this knowledge to break the location entropy setting. For example, consider a node B and its parent A, with A having a much greater entropy than B. Assume that the clients set the minimum privacy degree to the entropy of A. If the CSP knows that 99 percent of the users that visit A go to B next, then the CSP has a very high confidence to guess that if node A is accessed, B is the target node. Therefore, the effective privacy degree becomes as low as the entropy of B, which violates the requirement.

**Data item access frequency:** Similar to the index access frequency, the CSP may calculate the access frequency of each data item retrieved during the search and build a mapping between the trapdoor and the data item. Combining this information with some auxiliary knowledge, the CSP may predict e.g., popular data items.

To prevent these attacks, we propose two mechanisms. One is to select and add obfuscating locations into the query. This operation will make the access frequency of index nodes and data items learned by the CSP lost the ground truth. Another mechanism is to adapt the target data item into the one that has the smallest probability to be inferred by the CSP with acceptable interest loss.

As the first mechanism is quite obvious, details are not given. We call the second mechanism optimal privacy-preserving query generation based on existing knowledge (of the client). The goal is to form the query in such a way as to minimize the probability of inference by the CSP within acceptable interest loss. The proposed mechanism can be considered as a “hidden game”. It sets the optimal trapdoor by adjusting the parameters in the trapdoor based on the knowledge of access frequency and interest. The knowledge about access frequency can be classified into local and global. Locally, the client records its own service history and each time, from client’s point of view, it generates a trapdoor which has the minimum probability to be correctly inferred by the CSP under acceptable interest loss. Globally, the client considers the same problem from the view point of the CSP (assuming the client has access to the access frequency knowledge of the CSP). This mechanism is discussed in detail in C.

**APPENDIX C**

**Optimal query generation for location anonymity**

In the proposed system, the client provides locations, location attributes, and privacy degree for every query. In this part, we only focus on the location input as the other parts can be analyzed similarly. In addition, we provide the analysis based on data items as each data item is related to a location.

**C.1 Local optimal query generation for location anonymity**

The local optimal location anonymity is based on the local history of the client (user). The client stores all his previous query records (to analyze the access frequency for each location and corresponding retrieved data item) and generates his next query by minimizing the correct inference probability of the CSP (about his query). The client stores his previous queries in the following format.

\[
(\text{location, count, retrieved data items, identity})
\]

Assume loc\(^*\) is a location from the location set that user has interest, \(S_{loc}\) is a set of retrieved data items with input location \(loc, obj\) is the retrieved data item, and \(m\) is the total number of retrieved data items. Let \(n\) be the total number of locations queried by a user. Then, the following mapping can be built by a user.

\[
\text{CSP}_{\text{loc}_1, pk_1}, \ldots, \text{CSP}_{\text{loc}_n, pk_n} \rightarrow (S_{\text{loc}_1, pk_1}, \ldots, S_{\text{loc}_n, pk_n}),
\]

where \(pk_i\) denotes the set of identities used for location \(i\).

The goal of the user is to achieve

\[
\max(\text{Int}(\text{loc}^*) \cdot w_1 + \frac{1}{\min \Pr(\text{loc}^* | S_{\text{loc}_1}, \ldots, S_{\text{loc}_n}) \cdot w_2}),
\]

where \(Pr\) represents probability, \(Int\) is a function to calculate user’s interest in a given location, \(w_1\) is a weight value for user interest, \(w_2\) is a weight value for privacy, and \(S_{\text{loc}_i} = \sum_{j=1}^{n} S_{\text{loc}_i, pk_j}\). Furthermore, \(\Pr(\text{loc}^* | S_{\text{loc}_1}, \ldots, S_{\text{loc}_n})\) can be computed by the user as follows:

\[
\begin{align*}
\Pr(\text{loc}^* | S_{\text{loc}_1}, \ldots, S_{\text{loc}_n}) &= \Pr(\text{loc}^* | obj_1, \ldots, obj_m) \\
&= \frac{\Pr(\text{loc}^*, obj_1, \ldots, obj_m)}{\prod_{i=1}^{m} \Pr(\text{obj}_i)} \\
&= \frac{\Pr(obj_1, \ldots, obj_m | \text{loc}^*) \cdot \Pr(\text{loc}^*)}{\prod_{i=1}^{m} \sum_{j=1}^{n} \Pr(\text{loc}^*) \Pr(pk_j | \text{loc}^*) \Pr(\text{obj}_j | \text{loc}^*, pk_j) / \prod_{i=1}^{m} \sum_{j=1}^{n} \Pr(\text{pk}_j) \Pr(obj_j | \text{pk}_j)}
\end{align*}
\]

**C.2 Global optimal query generation for location anonymity**

For this purpose, we assume clients share their local records, and hence each client has a global view of the combined history. As the CSP predicts the location by analysing the observed data items, \(obj_1, \ldots, obj_m\), we can simulate the evaluation process as follows:

\[
\begin{align*}
\Pr(\text{loc}^* | obj_1, \ldots, obj_m) &= \frac{\Pr(obj_1, \ldots, obj_m | \text{loc}^*) \cdot \Pr(\text{loc}^*)}{\sum_{u=1}^{n} \Pr(obj_1, \ldots, obj_m | u, \text{loc}^*) \cdot \Pr(u | \text{loc}^*) \cdot \Pr(\text{loc}^*)} \\
&= \frac{\sum_{u=1}^{n} \Pr(obj_1, \ldots, obj_m | u, \text{loc}^*) \cdot \Pr(u | \text{loc}^*)}{\sum_{u=1}^{n} \sum_{j=1}^{n} \Pr(\text{u}) \Pr(pk_j | \text{u}) \Pr(obj_j | pk_j, \text{u})}
\end{align*}
\]
Fig. 19: Access probability of nodes follows the normal distribution

Fig. 20: Cumulative access probability of nodes follows the normal cumulative distribution

where

\[
\sum_{l=1}^{t} Pr(obj_1, \cdots, obj_m, u_l) \cdot Pr(u_l | loc^*) \cdot Pr(loc^*) \\
= \sum_{l=1}^{t} \sum_{j=1}^{n} \sum_{i=1}^{m} Pr(obj_i | pk_j, u_l, loc^*) \cdot Pr(loc^*) \cdot Pr(pk_j, u_l | loc^*)
\]

Here, \( u \) denotes the user, \( t \) is the total number of users, and \( loc^* \) is a location from the location set that user has interest. Then, as before, the problem becomes finding a solution that maximizes the following:

\[
w_1 \cdot Int(loc^*) + \\
w_2 \cdot \frac{\prod_{i=1}^{t} \sum_{j=1}^{n} \sum_{i=1}^{m} Pr(u_l) \cdot Pr(pk_j | u_l) \cdot Pr(obj_i | pk_j, u_l) \cdot Pr(loc^*) \cdot Pr(u_l | loc^*) \cdot Pr(loc^*)}{\sum_{l=1}^{t} Pr(obj_1, \cdots, obj_m, u_l) \cdot Pr(u_l | loc^*) \cdot Pr(loc^*)}
\]  

(8)

C.3 Discussion

From above two different query generation mechanisms, we can see that when the user wants to gain more privacy, it may lose some precision. Gradually, the precision loss may become unacceptable. For long term, we suggest the access frequency of tree nodes follows the gaussian distribution so that the system can run sustainably. To be precise, as shown in Figure 19, the top of the tree has more possibility to be accessed while the bottom has the less possibility, which naturally matches the search pattern. As CSP searches from top to bottom to find the target node, the top nodes (e.g., root) have to be accessed before target nodes are found. Therefore, all the nodes in the tree structure follow the normal cumulative distribution, as shown in 20. After properly setting above two distributions, client can employ previous two query generation mechanisms to improve query privacy from access frequency attack.

Above scheme also works in the environment where not all the clients share records.

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 Bilkent 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.