ABSTRACT

The widespread adoption of location-based services (LBS) raises increasing concerns for the protection of personal location information. A common strategy, referred to as obfuscation, to protect location privacy is based on forwarding the LBS provider a coarse user location instead of the actual user location. Conventional approaches, based on such technique, are however based only on geometric methods and therefore are unable to assure privacy when the adversary is aware of the geographical context. This paper provides a comprehensive solution to this problem. Our solution presents a novel approach that obfuscates the user location by taking into account the geographical context and user’s privacy preferences. We define several theoretical notions underlying our approach. We then propose a strategy for generating obfuscated spaces and an efficient algorithm which implements such a strategy. The paper includes several experimental results assessing performance, storage requirements and accuracy for the approach. The paper also discusses the system architecture and shows that the approach can be deployed also for clients running on small devices.

Categories and Subject Descriptors
H.2.0 [General]: Security, integrity, and protection; H.2.8 [Database Management]: Spatial Databases and GIS

General Terms
Design, Experimentation, Security

Keywords
Location Privacy, Location Based Services

1. INTRODUCTION

The ever increasing collection of personal location data, pushed by the widespread use of location-sensing technologies, like satellite positioning systems, RFID and sensors,

and the development of location-based services (LBS), motivates the great concern for the protection of personal location information (location privacy). The communication of a user’s position to a LBS provider upon a service request may result in the unauthorized dissemination of personal location data. Such data, combined with other available information, may in turn lead to the inference of sensitive information about individuals. Various approaches have been thus proposed to assure location privacy. Most of those approaches are based on obfuscation techniques that aim at disguising the actual position of the user by forwarding to the LBS provider fake or less accurate (generalized) location information. Approaches based on k-anonymization refine obfuscation techniques by making sure that the generalized location of each user is indistinguishable from the generalized locations of other \(k-1\) users.

A common problem to all the above approaches is that they do not take into account geographical knowledge that the adversary may have about the reference spatial context. We claim that by exploiting such knowledge, the adversary may be able to obtain more precise bounds about the actual user location (referred to as location inference), thus defeating the obfuscation mechanism. Another major drawback of such approaches is that they do not support location privacy preferences, that is, the specification of which locations are sensitive for which users. Not all locations may have the same sensitivity for all the users and therefore a suitable obfuscation mechanism should be able to generate obfuscation locations that are tailored to the privacy preference of each user. We believe that, as we move toward more personalized LBS, privacy should be one of key personalization dimensions. Before moving to introduce the key contribution of the paper, we introduce a running example to illustrate the location inferences that are possible when geographical knowledge is available to the adversary.

\textbf{Example 1.} Assume that a user of a LBS is located within a hospital which for this user is a sensitive place. Consider the geographical context in Figure 1.a: the hospi-
tal $H$ is close to a lake $L$ and to a residential district $R$; all these places, i.e., the lake, the district and the hospital, cover a polygonal region. Suppose that no boats are allowed on the lake and that the adversary has this knowledge. Assume also that the actual position of the user is obfuscated by region $O$ containing the user’s position (obfuscated location). From the observation of the spatial relationships existing between the obfuscated location and the spatial entities, like spatial containment, overlaps and disjointness, the adversary can easily infer whether the user is located in a sensitive place. In particular consider the following three cases:

i) The obfuscated location is spatially contained in the extent of the hospital (Figure 1.b). In this case, the adversary may easily infer that the user is located in a sensitive place, that is, the hospital, although the actual position is blurred to a coarser region.

ii) The region corresponding to the user’s obfuscated location includes the extent of both the hospital and the lake (Figure 1.c). Since the user cannot be physically located inside the lake, because no boats are allowed on the lake, the only realistic position is within the hospital and thus the obfuscated location is still sensitive. Notice that in this case information about the user’s obfuscated location is combined with publicly available information, i.e., that no boat is allowed on the lake, in order to infer more precise information about the actual location of the user.

iii) The region corresponding to the user’s obfuscated location overlaps part of the hospital and part of the residential district (Figure 1.d). Since the hospital is the only sensitive place, we can say that the obfuscated location is “sensitive to some extent”.

Suppose now that the user is a physician. In such case the fact of being located in the hospital is likely not to be sensitive for this user.

The example emphasizes the fact that a location, besides a geometric shape, has a semantics (e.g., hospital) which depends on the entities spatially related to such position. The example clearly shows that privacy breaches occur because existing obfuscation techniques are unable to protect against the inferences made by linking the geometric information with the semantic location which, depending on the perceptions of users, may represent sensitive information. The protection of sensitive location information thus requires techniques able to take into account the geographical context in which users are located, in particular the semantic locations and the spatial distribution of population, as well as the users’ privacy preferences. To our knowledge, a comprehensive approach to this problem has not been investigated yet.

In this paper we take a step in that direction and present a novel method for the personalized obfuscation of semantic locations. A key concept in our approach is the sensitivity metric which quantifies the sensitivity of a region, i.e., “how much private” a region is. The choice of the metric is crucial, and, indeed, different metrics can be devised. We define the sensitivity of a region $r$ with respect to a certain category of semantic locations (e.g., hospitals, religious buildings) as the probability that a user in $r$ is inside any place of that category. Users can specify in a profile which categories of semantic locations are sensitive as well as the desired degree of protection for each of those categories. We define the user profile as a set of constraints on the maximum sensitivity of obfuscated locations which is tolerated by the user. The privacy-preserving strategy is then articulated in two stages: in the first stage, an obfuscation algorithm generates a set of regions (obfuscated locations) masking the extent of sensitive locations. Each of those regions includes both sensitive and innocuous semantic locations, and satisfies the user profile constraints. In the second stage, the user’s position is checked against the obfuscated locations and, if the user falls inside location $r$, then $r$ is disclosed in place of the exact position. It is important to observe that the obfuscation algorithm does not take into account the user’s position. This way, an attacker cannot exploit the knowledge of the algorithm to infer more precise bounds over the user’s position inside the larger region. Therefore the method is robust against reverse engineering attacks.

The above elements are combined in the framework, referred to as PROBE (Privacy-preserving Obfuscation Environment). The PROBE framework can be flexibly deployed on either a two-tier architecture or in alternative, whenever the client devices have limited capabilities, on a three-tier architecture [5]. In the former case, the obfuscated locations are generated by the client; in the latter case, by a third system, e.g., a Web application or a laptop, and then downloaded on the client. To summarize, the key contributions of this paper are:

- A privacy model for the specification of privacy preferences on semantic locations. Semantic locations are defined in compliance with geo-spatial standards. The privacy model comprises the sensitivity metric and the user profile model.

- An obfuscation algorithm called $\text{Sens}_{\text{Hil}}$. Experimental results show that our algorithm is very efficient and the size of obfuscated maps is very small and thus suitable for storage on small devices.

The rest of the paper is organized as follows. Next section overviews related work. Then we introduce the location privacy model. The obfuscation algorithm is described in the subsequent section followed by the experimental evaluation. Final remarks and future plans are finally reported in the concluding section.

2. RELATED WORK

The bulk of research on location privacy in LBS has focused on the development of spatial cloaking techniques [3, 21, 4, 7, 12, 9, 10]. In place of the exact user’s position, spatial cloaking methods disclose a less accurate cloaked region $CR$ containing the user. Spatial cloaking is extensively applied to provide spatial k-anonymity [4, 7, 12, 9, 10]. The exact position of an individual is replaced by a CR containing $k$ users, while the privacy metric is defined by the probability $1/k$ of identifying a user in $CR$. However, spatial k-anonymity does not provide any protection against location inferences [1]. For example, if an attacker knows that John is among $k$ users in the cloaked region $CR$, and $CR$ is inside a hospital, then the attacker can promptly infer that John might have health problems. The reason of this privacy leak is that spatial k-anonymity only protects users’ identities and not the semantic locations.
A different approach which provides strong privacy guarantees is to encode a location-based query using cryptographic techniques. For example, Ghinita et al. [6] present a technique based on the PIR theory (Private Information Retrieval) to compute nearest-neighbor (NN) queries without disclosing any location information about the user and the requested points of interest. This technique, however, suffers from two main drawbacks: it is tailored on a particular class of queries, i.e., NN-queries; furthermore, it incurs in high communication costs.

The protection of semantic locations has been addressed by Xue et al. [20]. The goal is to prevent the individuals inside a spatial k-anonymous region from being located in a semantic location. The method extends the notion of l-diversity [11] to the spatial case by adding the privacy constraint that a k-anonymous region must contain not only k users, but also l different semantic locations. This approach, however, does not take into account the personal preferences of users (i.e., the hospital is sensitive for a patient, but it is not so for a doctor). Further, the resulting region can be very broad, especially in geographically homogeneous areas. Our approach overcomes those drawbacks and provides a flexible solution in which users can request personalized obfuscation while limiting the loss of spatial resolution.

Personalized privacy preservation is the major goal of policy-based approaches. Location privacy policies, in particular, are commonly applied to control the disclosure of personal locations to third parties [14, 22, 8, 18, 16]. Those policies rely on models and protocols which often derive from the W3C Platform for Privacy preferences (P3P) like in [14] or that are rooted in access control [18, 22, 8, 16]. Those approaches, however, are not able to contrast inference channels, and in particular location inferences. On the other hand, privacy personalization in the framework of spatial k-anonymity [4] allows the specification of very simple parameters, like the value of k and the minimum size of the cloaked region. More pertinent to our problem, is the approach by Tao et al. [19] presenting a privacy personalization technique for the protection of sensitive attributes in non-spatial, k-anonymous datasets. The substantial difference with PROBE, is that our system not only allows the specification of user preferences to control location inferences but also is able to generate generalized locations.

To summarize, there are various proposals which aim to protect location either through spatial cloaking, policy-based approaches or using strong but expensive solutions like cryptographic techniques. None of them, however, is able to provide at the same time a personalized and cost-effective protection of semantic locations, which instead is the unique contribution of PROBE.

3. THE PRIVACY MODEL

We now introduce the privacy model defined in PROBE. The model is articulated in four components, namely the space model, the sensitivity metric, the user profile, and the obfuscated location model, that we describe in detail in what follows.

3.1 Space model

The region of concern for the LBS application is referred to as reference space \( \Omega \). \( \Omega \) is a possibly bounded and connected area in a two-dimensional space. The geometric objects in \( \Omega \) have a spatial type compliant with geo-spatial standards [15]. The user’s position is a point. Moreover spatial types are closed under (appropriately defined) geometric union \( \cup^* \), intersection \( \cap^* \), difference \( \setminus^* \).

Following geo-spatial standards, the semantic locations are described in terms of spatial features (simply features hereinafter). Features have a type. Further features have an extent of region type. Moreover, we assume features to be spatially disjoint. Note that if two places are one contained in the other, the corresponding features must be defined so that they do not overlap. For example, if a restaurant is within a park, the extent of the park feature should have a hole corresponding to the extent and location of the restaurant feature. We denote with \( \text{Cov}(ft) \) a function which yields the spatial union of the extents of all features of type \( ft \). The pair of sets \((FT,F)\) representing respectively feature types and features is referred to as the geographical database of the application.

Feature types can be classified as sensitive. The classification easily extends to regions. Let \( r \) be a region and \( FS \) be the set of sensitive feature types. We say that \( r \) is sensitive if it overlaps \( \text{Cov}(ft) \), for some sensitive feature type \( ft \), that is,

\[
\bigcup_{ft \in FS} \text{Cov}(ft) \cap r = \emptyset.
\]

3.2 Sensitivity metric

The distribution of the user’s positions in the reference space is assumed to be known and is described by the probability density function pdf. \( P(r) = \int_{t} \text{pdf} \) denote the probability that a user, known to be located in \( \Omega \), is actually located in region \( r \); \( P(\Omega) = 1 \) and \( P(\emptyset) = 0 \). We say that a region \( r \) is unreachable if \( P(r) = 0 \); instead \( r \) is reachable if at least one subregion \( r' \subset r \) exists such that \( P(r') > 0 \). We refer to such a probability as user’s position probability.

The user’s position probability is used to define the sensitivity of a region. The sensitivity of a region is a value in the interval \([0,1]\) which quantifies “how much private” a region is. Consider a sensitive feature type \( ft \). We define sensitivity of \( r \) wrt \( ft \), denoted as \( P_{\text{sens}}(ft,r) \), the probability that a user, known to be in \( r \), is actually within the extent of any sensitive feature of type \( ft \) overlapping with \( r \). Such a sensitivity is expressed in terms of conditional probability as follows:

\[
P_{\text{sens}}(ft,r) = \begin{cases} 
P(\text{Cov}(ft) \mid r) & \text{if } P(r) \neq 0 \\
0 & \text{otherwise}
\end{cases}
\]

Note that the sensitivity of an unreachable region is set to 0 for any feature type because a user cannot be located in such a region. Further, if the region is entirely covered by sensitive features of the same type \( ft_0 \), then \( P_{\text{sens}}(ft_0,r) = 1 \). The function \( P_{\text{sens}}(ft,r) \) can be rewritten as:

\[
P_{\text{sens}}(ft,r) = \begin{cases} 
\int_{\text{Cov}(ft) \cap r} \text{pdf} & \text{if } P(r) \neq 0 \\
0 & \text{otherwise}
\end{cases}
\]

**Example 2.** Consider the region \( r \) in Figure 2. Such a region overlaps two features \( H_1 \) and \( H_2 \) of type Hospital, which is sensitive feature type. \( H_1 \) is partially contained in \( r \) and \( H_2 \) is entirely inside the region. Moreover the region includes lake \( L \). Assume the following distribution of user’s positions: \( L \) is unreachable; the user’s positions in \( r \setminus L \)
are equally probable. The sensitivity of \( r \) with respect to the feature type HealthOrganization can be computed as follows:

\[
P_{\text{sens}}(\text{Hospital}, r) = \frac{\text{Area}(H_1 \cap r) + \text{Area}(H_2)}{\text{Area}(r \setminus L)}
\]

Figure 2: Example of sensitive region including an unreachable region

3.3 The privacy profile

The privacy profile specifies the set of sensitive feature types, i.e., \( FT_S \) and the set of user preferences. A privacy preference (preference for short) is a constraint over the maximum sensitivity that the user tolerates in any location wrt a certain sensitive feature type in \( FT_S \). Given a feature type \( ft \in FT_S \), a preference takes the form

\[
T(ft) = v
\]

where \( v \in (0, 1) \) is the threshold sensitivity value of \( ft \). We say that a region \( r \) satisfies preference \( T(ft) = v \) if the following inequality holds:

\[
P_{\text{sens}}(ft, r) \leq v.
\]

Note that we do not consider the preference \( T(ft) = 1 \) because it would mean that \( ft \) is not sensitive, against the initial assumption. We also rule out the preference \( T(ft) = 0 \) because it can be only satisfied if \( ft \) has no instances which is not an interesting case.

The privacy profile takes the form of the tuple:

\[
< FT_S, T >
\]

where \( FT_S = \{ft_1, ..ft_n\} \) is the set of sensitive features types and \( T = \{T(ft_1), ..T(ft_n)\} \) is the set of preferences, one for each sensitive feature type.

Example 3. A possible privacy profile of a user who is concerned with the disclosure of positions in religious buildings and in health organizations can be defined as follows:

- \( FT_S = \{\text{Health Organization}, \text{Religious Building}\} \)
- \( T = \{T(\text{hospital}) = 0.4, T(\text{Religious Building}) = 0.1\} \).

It can be noticed that the threshold value is lower for the feature type Religious Building than for the feature type Health Organization to mean that the privacy demand is stronger in the former case.

3.4 Obsfuscated location and obfuscated map

At this point, we are able to formally define the concept of obsfuscated location. Consider the privacy profile \( p =< F_S, T > \) with \( F_S = \{ft_1, ..ft_n\} \). Let \( r \) be a region. We say that \( r \) is an obsfuscated location for profile \( p \) if and only if \( r \) is sensitive and every preference in the set \( T \) is satisfied. The latter property can be formally expressed as:

\[
\forall ft \in FT_S, P_{\text{sens}}(ft, r) \leq T(ft)
\]

where \( T = \{T(ft)\}_{ft \in FT_S} \) are the privacy preferences of the profile \( p \). A related concept is that of obfuscated map. An obfuscated map for profile \( p \) is a set \( S = \{r_1, ..r_n\} \) of obfuscated locations which are disjoint and cover the whole set of sensitive features. Since the regions are disjoint, a user can be located at most in one obfuscated location. We recall that the obfuscated locations are disjoint if their spatial intersection is the empty set, that is: \( \forall i, j \in \{1..n\}, i \neq j \Rightarrow r_i \cap r_j = \emptyset \).

Moreover, since the regions cover the sensitive features, every sensitive position falls inside an obfuscated location. Such a condition is verified if the spatial union of the obfuscated locations is a superset of the sensitive portion of space, that is:

\[
\bigcup_{i \in \{1..n\}} r_i \supset \bigcup_{ft \in FT_S} \text{Cov}(ft)
\]

Note that the obfuscated map does not cover necessarily the whole space.

4. THE OBFUSCATION PROCESS

Figure 3 shows the reference architecture of the PROBE system. PROBE assumes a conventional networked architecture consisting of a LBS server and a set of GPS-equipped mobile clients. The core component of the system is the Obfuscation Engine. The Obfuscation Engine computes the obfuscated locations. For the sake of efficiency, the obfuscation process is organized in two phases called off-line and run-time, respectively. In the off-line phase the user invokes the Obfuscation Engine to compute the obfuscated map based on the privacy profile. We emphasize that this way the obfuscated locations are all computed before any request is made with consequent gain of efficiency at run-time. Moreover, the obfuscated maps are to be re-calculated only when the geographical database or the user profile changes. At run-time, upon a service request, the client simply matches the user’s position against the obfuscated map. If the position falls inside an obfuscated location, then the actual position is replaced by the coarser position which is then transmitted to the LBS provider. Otherwise the position is transmitted without changes.

A key choice concerns the implementation of the obfuscation algorithm. Computing an obfuscated map means to determine a set of regions which satisfy the constraints specified in the privacy profile. The problem is not trivial because...
The idea is to obfuscate each cell \( ops \) by progressively enlarging the region containing \( ops \) and progressively aggregating neighbor cells. The process develops like a border. Given two adjacent cells \( GR_C \) in reference space and \( P_o \) in 

\[ GR_C \] space and \( P_o \) in 

\[ P \] space greater than the maximum sensitivity of a derived partition. It is shown that the maximum sensitivity value in a partition it is necessary to limit the loss of geometric precision because an obfuscated region may have a complex shape depending on 

\[ 4.1 \text{ Outline of the strategy} \]

A flexible approach is to adopt a grid-based representation of space [1]. Assume space to be subdivided in cells of regular and sufficiently small size. Features are typically agglomerates of cells. Given a sensitive feature type \( FT_o \) and an instance \( F_o \), each cell \( c \) within \( F_o \) has sensitivity \( P_{sens}(FT_o,c) = 1 \). Since such sensitivity is greater than the threshold sensitivity value we say that \( c \) is over-sensitive. The idea is to obfuscate each cell \( c \) which is over-sensitive by progressively aggregating neighbor cells. The process develops by progressively enlarging the region containing \( c \) until an obfuscated location is achieved or the region degenerates in the whole space. A key question is whether this method leads to a solution. It can be shown that, for how the sensitivity metric is defined, the method converges towards a solutions if that exists.

\[ 4.1.1 \text{ Soundness of the aggregation method} \]

We introduce first some preliminary definitions. Let \( GR = \{c_1, c_2, \ldots, c_n\} \) be the grid defined over the reference space and \( C \) be a partition (not necessarily the initial one) of \( GR \). Two cells \( c_1, c_2 \in C \) are adjacent if they have a common border. Given two adjacent cells \( c_1, c_2 \), the operation which merges the two cells generates a new partition \( C' \) in which cells \( c_1 \) and \( c_2 \) are replaced by cell \( c = c_1 \cup c_2 \). We say that partition \( C' \) is derived from partition \( C \), written as \( C' \supseteq C \). Consider the set \( \mathcal{P}_C \) of partitions derived directly or indirectly from the initial partition \( C_0 \) through subsequent operations of merge. The set \( H = \{\mathcal{P}_C, \supseteq \} \) is a bounded lattice in which the least element is the initial partition while the greatest element is the partition consisting of a unique element, that is, the whole space (called maximal partition).

We now show that when two cells are merged, the sensitivity of the resulting cell is lower than the maximum between the sensitivity values of the two starting cells. Then it is shown that the maximum sensitivity value in a partition \( C_0 \) (wrt each feature type and each region) is weakly anti-monotonic with respect to the “is derived” relation, that is, that the maximum sensitivity of a partition is equal or greater than the maximum sensitivity of a derived partition. Finally we show that whenever the sensitivity of the reference space \( \Omega \) is known, one can determine whether at least one solution exists. Proofs are reported in [2].

**Theorem 4.1.** Let \( c_1 \) and \( c_2 \) be two cells of a partition \( C \), and \( c = c_1 \cup c_2 \). Then, the region resulting from the merge of two cells has a sensitivity which is not higher than the maximum sensitivity of the initial cells, that is:

\[ P_{sens}(ft, c) \leq \max\{P_{sens}(ft, c_1), P_{sens}(ft, c_2)\} \]

**Theorem 4.2.** Let \( C_A \) and \( C_B \) be two distinct partitions of \( \mathcal{P}_C \) and let \( ft \in FT_3 \) be a sensitive feature type. Then the maximum sensitivity of a partition is equal or greater than the maximum sensitivity of a derived partition, that is:

\[ C_A \supsetneq C_B \implies \max_{r \in C_A} P_{sens}(ft, r) \leq \max_{r \in C_B} P_{sens}(ft, r) \]

**Algorithm 1 SensHil Algorithm**

\[ \begin{array}{ll}
1: \text{function } HilObfuscate(grid, pp) & \triangleright \text{Obfuscate grid using privacy profile } pp \\
2: \quad S \leftarrow \emptyset & \triangleright \text{Obfuscated regions} \\
3: \quad \text{for } idx \leftarrow 0 \ldots \maxHilbertIdx(grid) \text{ do} & \triangleright \text{Hilbert scan} \\
4: \quad \quad \text{cell} \leftarrow \text{getHilbertCell}(idx) & \\
5: \quad \quad \text{if } OverSensitive(r, pp) \text{ then} & \triangleright \text{Get the current (one-cell) interval} \\
6: \quad \quad \quad \text{if } Sensitive(r, pp) \text{ then} & \triangleright \text{Sensitivity} \\
7: \quad \quad \quad \quad \text{fixBackward}(S, pp) & \triangleright \text{Fix the last interval if needed} \\
8: \quad \quad \quad \text{add}(S, r) & \\
9: \quad \quad \quad \text{end if} & \triangleright \text{Obfuscated locations. In our algorithm, a cell } c \text{ is obfuscated by progressively aggregating the cells which are close to } c \text{ in the linear ordering. An obfuscated location is thus simply defined by an interval in the linear space.} \\
10: \quad \text{else } \text{SensHil} & \triangleright \text{SensHil maps the grid onto a Hilbert space-filling curve and then performs cell aggregation over such a space. The Hilbert space-filling curve is a one-dimensional curve which visits every point within a discrete two-dimensional space. Similarly to the approaches in [10, 9], we exploit the locality property of Hilbert curves [17] to generate obfuscated locations. In our algorithm, a cell } c \text{ is obfuscated by progressively aggregating the cells which are close to } c \text{ in the linear ordering. An obfuscated location is thus simply defined by an interval in the linear space.} \\
11: \quad \text{end if} & \\
12: \text{end for} & \\
13: \text{return } S & \\
\end{array} \]

**4.2 The obfuscation algorithm**

We emphasize that the above aggregation method is applied to the single cells and not to the whole sensitive feature. A single feature can thus be obfuscated by several regions, each covering a portion of such feature. An advantage of this method is that by fragmenting the original sensitive region and expanding each fragment separately, one can generate obfuscated locations which are smaller than the regions that would be obtained by obfuscating the entire region. The result is a finer-grained obfuscation and thus a better QoS.

We now present an obfuscation algorithm, called SensHil, which applies the above strategy to provide fine-grained obfuscated locations. SensHil maps the grid onto a Hilbert space-filling curve and then performs cell aggregation over such a space. The Hilbert space-filling curve is a one-dimensional curve which visits every point within a discrete two-dimensional space. Similarly to the approaches in [10, 9], we exploit the locality property of Hilbert curves [17] to generate obfuscated locations. In our algorithm, a cell \( c \) is obfuscated by progressively aggregating the cells which are close to \( c \) in the linear ordering. An obfuscated location is thus simply defined by an interval in the linear space.

**Algorithm 1 SensHil Algorithm**

1: function HilObfuscate(grid, pp) \triangleright Obfuscate grid using privacy profile pp
2: \quad S \leftarrow \emptyset \triangleright Obfuscated regions
3: \quad for idx \leftarrow 0 \ldots \maxHilbertIdx(grid) do \triangleright Hilbert scan
4: \quad \quad cell \leftarrow getHilbertCell(idx)
5: \quad \quad if OverSensitive(r, pp) then \triangleright Get the current (one-cell) interval
6: \quad \quad \quad if Sensitive(r, pp) then \triangleright Sensitivity
7: \quad \quad \quad fixBackward(S, pp) \triangleright Fix the last interval if needed
8: \quad \quad \quad add(S, r)
9: \quad \quad end if
10: \quad \text{end if}
11: \text{return } S
12: end function

**4.2.1 Details of the algorithm SensHil**

Algorithm 1 details the function HilObfuscate generating an obfuscated map. The algorithm consists of two phases. The first phase is called forward generalization. The algorithm starts scanning the cells sequence (in the linear ordering) from the first cell.

As an over-sensitive cell is found, the algorithm attempts to generate a obfuscated interval \( r \) starting from cell (function generalizeForward in Algorithm 2). If such interval is found, \( r \) is inserted into the result set \( S \) and the scan proceeds until every cell has been examined. In case cell is sensitive, but not oversensitive, no further generalization is needed and the one-cell interval representing the cell is inserted into \( S \).

Upon completion of the scan, it may happen that the last sensitive cell cannot be generalized, because, for example, represents the last cell in the cell sequence. If this is the case, the algorithm expands the current interval back-
into the set Sens most twice, once per phase. The overall complexity of the operation, at line 17, may entail a change of the set Sens to ensure that intervals are disjoint, the addition of backward generalization type. Further, we set the privacy threshold to 25%.

Algorithm 2 SensHil Algorithm subroutines

<table>
<thead>
<tr>
<th>Line</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>function GENERATEFORWARD(r, grid, pp)</td>
<td>Expand current interval</td>
</tr>
<tr>
<td>2</td>
<td>for idx ← r.first . . . maxHilbertIdx(grid) do</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>r.last ← idx</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>if ¬OverSensitive(r, pp) then</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>return r</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>end if</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>end for</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>return r</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>end function</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>procedure FIXBACKWARD(S, pp)</td>
<td>Backward expansion if the last interval r violates pp</td>
</tr>
<tr>
<td>11</td>
<td>r ← last(S)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>if OverSensitive(r, pp) then</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>r ← GENERALIZEBACKWARD(r, grid, pp)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>if OverSensitive(r, pp) then</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>return r</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>if ¬ ⊖ then</td>
<td>Obfuscation failed</td>
</tr>
<tr>
<td>17</td>
<td>add(S, r)</td>
<td>Also remove redundancies</td>
</tr>
<tr>
<td>18</td>
<td>end if</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>end if</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>end procedure</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>function GENERATEBACKWARD(r, grid, pp)</td>
<td>Expand backward</td>
</tr>
<tr>
<td>22</td>
<td>for idx ← r.first . . . 1 . . . 0 reverse do</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>r.first ← idx</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>if ¬OverSensitive(r, pp) then</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>return r</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>end if</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>end for</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>return r</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>end function</td>
<td></td>
</tr>
</tbody>
</table>

Towards until a convenient interval is found or the entire sequence of cells is scanned again from the last cell to the first one (function FIXBACKWARD in Algorithm 2). This phase is called backward generalization. Note that in order to ensure that intervals are disjoint, the addition of r to S in the backward phase through the add(S, r) operation, at line 17, may entail a change of the set S. For example, the operation add(\{1, 2\}, \{4, 7\}) results into the set S = \{1\}. Each cell is examined at most twice, once per phase. The overall complexity of the SensHil algorithm is, thus, O(|GR|), where |GR| is the number of grid cells. It can be shown that the non-empty set HilObfuscated(GR, pp) is an obfuscated map.

Example 4. Figure 4 illustrates the step-wise execution of the algorithm applied to a grid containing two sensitive features. The initial grid is labelled as 'START' in the figure (on the left) while the obfuscated map is labelled 'RESULT' (on the right). The two sensitive features are of the same type. Further, we set the privacy threshold to 25%. The reference space is described by a 4 x 4 grid, consisting of 16 cells. Cells are transversed following the Hilbert ordering (subfig. labeled as 'GRID', on the left). Subfigures 4.1–16 show the steps of the algorithm. At each step, the region being examined is labelled either 'OK' or 'NO', depending on whether the privacy preference is satisfied or not. Step 1 and 2 simply skip the first two cells in the ordering because not sensitive. The 3rd cell, instead, is largely covered by sensitive features. Thus, in the subsequent steps (subfig. 3-6) the cell is progressively aggregated with neighbor cells until an obfuscated region is found (subfig. 6). The algorithm proceeds until the last cell has been processed.

4.3 An example of obfuscation

In Figure 5 we apply SensHil to obfuscate two existing hospitals located in New Haven (US), named Hospital of S. Raphael and Yale-New Haven Hospital respectively. The two hospitals represent two features of the sensitive feature type Hospital. Figure 5.a. zooms on the two sensitive locations, situated respectively on the North and on the South. Figure 5.b shows the cell-based representation of the hospitals extent in a grid of 128 x 128 cells. Cells have a size of about 20 metres. The cells covering the hospitals have sensitivity value 1 wrt the feature type Hospital. We generate the obfuscated map using the profile <FT3, T> with FT3 = {Hospital} and T = {T(Hospital) = 0.3}. The resulting obfuscated locations are displayed in Figure 5.c in light grey (light blue in the color version). The obfuscated map consists of 7 obfuscated locations, covering the two hospitals. It can be noticed the irregular shape of the obfuscated regions. Each shape can however be simply described by an interval in the Hilbert space-filling curve.

5. EXPERIMENTAL EVALUATION

We have made experiments to evaluate various parameters using grids of fixed and varying size, with different percentage of sensitive cells (referred to as coverage) and privacy thresholds. We assume that all user’s positions are equally probable, except in those regions which are explicitly defined to be unreachable. SensHil is implemented in Java, using the library [13]. The experiments were run on a laptop PC equipped with an AMD Turion Mobile MT30 1.6GHz CPU, 1.37GB of RAM and Windows XP.

We have run the experiments over synthetic data. For the generation of synthetic data we developed the Spatially-aware Generalization (SAG, for short) tool. SAG enables the generation of grids randomly populated by features of user-defined type. Features have a rectangular shape, of varying size, and are represented as group of cells. Each cell c is either empty or completely covered by a feature.
of feature type $ft$. Thus, $P_{sens}(ft, c) \in \{0, 1\}$. Grids are generated based on the following parameters: the size of the grid; the set $FT$ of feature types; for each type $ft \in FT$, the percentage of cells covered by $ft$; the features types denoting unreachable regions, such as lake. Hence SAG populates the reachable portion of space with rectangles of varying size. The side of each rectangle is generated on a random basis using a binomial distribution. The average size of the rectangle side is 3 cells and the range is $[0, 6]$. SAG also enables the specification of privacy profiles. The user flags the features types in the set $FT$ that are sensitive and then specifies for each sensitive feature the threshold value.

5.1 The experiments

For comparison purposes, we developed the algorithm $Sens_{Pyr}$ [2] which provides an alternative implementation of the PROBE obfuscation strategy. $Sens_{Pyr}$ uses a pyramid data structure to represent the space grid similar to the structure used for spatial k-anonymization in the Casper system [12]. The pyramid takes the form of a tree in which the nodes represent the regions obtained by recursively subdividing space in four quadrants until the base cells are reached. The root at level 0 corresponds to the entire reference space; the leaves are the cells of the finest-grained grid; a cell $c$ which is not a leaf has four children, one for each quadrant of the region denoted by $c$. The process of cell aggregation works as follows: first, each leaf which is over-sensitive is aggregated with the cells of the quadrant the cell belongs to. If the resulting region remains over-sensitive, the aggregation is possibly recursively applied to the cells of the grid at the immediately lower level in the pyramid. Figure 6 highlights the different shape of the obfuscated locations generated by the two algorithms (the red cells are sensitive, the regions including blue cells are the obfuscated locations).

We have made five experiments with the two algorithms. Experiments 1, 2, 3, and 5 use a grid of size $1024 \times 1024$ cells. At a resolution of 10 metres, the reference space is thus about $10km \times 10km$ which is the size of an average city. The independent variable in the experiments is the coverage which ranges in the interval $[1, 45]$, which seems a reasonable choice; a value of $x$ means that the percentage of sensitive cells is $x\%$. Further, we consider three possible values for the threshold function, that is, $T(ft) \in \{0.1, 0.2, 0.4\}$. Each algorithm is run 100 times, for different values of the coverage and the threshold value and average values are reported for each experiment. Experiment 4 evaluates the two algorithms on grids of increasing size ranging between $64 \times 64$ and $4096 \times 4096$ with fixed coverages equal to 0.5% and 10%.

**Experiment 1: Success rate.** The outcome is the rate of successful generation of obfuscated maps (success rate). As the coverage increases and the privacy requirements become more restrictive, the probability of failure in the map generation increases. The graphs in Figure 7 show that the generation is successful until the percentage of coverage is below a breaking value. For example, when the threshold has value 0.4, the breaking value is nearly 40. It can be noticed that the breaking values are nearly the same for the two algorithms.

**Experiment 2: Average number of obfuscated regions.** The outcome is the average number of obfuscated locations computed by the two algorithms when the map generation process does not fail. The two graphs in Figure 8 show that the number of obfuscated regions generated by $Sens_{Hil}$ is significantly higher than the number of obfuscated regions generated by $Sens_{Pyr}$. It can be noticed that the cardinality increases up to a maximum value and then decreases. The reason of such behavior is that for low percentages of coverage, the number of cells to obfuscate is relatively low. The number of obfuscated regions however increases up to a maximum value. When the density of sensitive cells is too high the algorithms generate large obfuscated areas and thus the number of obfuscated regions globally decreases.

**Experiment 3: Average size of the obfuscated location.** The outcome is the average number of cells in an obfuscated region. Not surprisingly, the graphs in Figure 9 show that the $Sens_{Hil}$ generates more precise obfuscated maps than $Sens_{Pyr}$. Quantitative values are reported later on. It can be noticed that the size of the obfuscated maps...
grows very rapidly, especially for $\text{Sens}_{Pyr}$, as the percentage of coverage becomes closer to the breaking point.

<table>
<thead>
<tr>
<th>GridSize</th>
<th>NReg</th>
<th>NCell</th>
<th>GenTime (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 × 64</td>
<td>28</td>
<td>112</td>
<td>0.49</td>
</tr>
<tr>
<td>128 × 128</td>
<td>106</td>
<td>120</td>
<td>2.08</td>
</tr>
<tr>
<td>256 × 256</td>
<td>429</td>
<td>118</td>
<td>11.5</td>
</tr>
<tr>
<td>512 × 512</td>
<td>1726</td>
<td>118</td>
<td>72</td>
</tr>
<tr>
<td>1024 × 1024</td>
<td>6943</td>
<td>117</td>
<td>185</td>
</tr>
<tr>
<td>2048 × 2048</td>
<td>27735</td>
<td>117</td>
<td>758</td>
</tr>
<tr>
<td>4096 × 4096</td>
<td>111026</td>
<td>117</td>
<td>3255</td>
</tr>
</tbody>
</table>

Table 1: Measures for $\text{Sens}_{Pyr}$

<table>
<thead>
<tr>
<th>GridSize</th>
<th>NReg</th>
<th>NCell</th>
<th>GenTime (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 × 64</td>
<td>43</td>
<td>46</td>
<td>0.49</td>
</tr>
<tr>
<td>128 × 128</td>
<td>175</td>
<td>47</td>
<td>2.17</td>
</tr>
<tr>
<td>256 × 256</td>
<td>700</td>
<td>47</td>
<td>8.8</td>
</tr>
<tr>
<td>512 × 512</td>
<td>2835</td>
<td>46</td>
<td>31</td>
</tr>
<tr>
<td>1024 × 1024</td>
<td>11372</td>
<td>46</td>
<td>177</td>
</tr>
<tr>
<td>2048 × 2048</td>
<td>45483</td>
<td>46</td>
<td>543</td>
</tr>
<tr>
<td>4096 × 4096</td>
<td>181920</td>
<td>46</td>
<td>2104</td>
</tr>
</tbody>
</table>

Table 2: Measures for $\text{Sens}_{Hil}$

Experiment 4: Grid size. Table 1 and Table 2 report the measures obtained by running the two algorithms over grids of increasing size ranging between 64 × 64 and 4096 × 4096 with a 10% coverage. Each table row specifies: the grid size (GridSize), the average number of regions (NReg), the average number of cells per regions (NCell), and the map generation time (GenTime). If we look at the experiments over a grid of 1024 × 1024 we observe that:

- The number of obfuscated regions generated by $\text{Sens}_{Hil}$ is about 40% higher than the number of regions generated by $\text{Sens}_{Pyr}$.
- The average number of cells per obfuscated regions in $\text{Sens}_{Hil}$ is 46 against 118 of $\text{Sens}_{Pyr}$. At the given resolution (10m × 10m) the average area of the obfuscated region generated by $\text{Sens}_{Hil}$ is thus 4600m$^2$.
- The map generation time for $\text{Sens}_{Hil}$ is 177 ms against 185 ms of $\text{Sens}_{Pyr}$. Thus, the performance is thus not significantly different, but the $\text{Sens}_{Hil}$ computes significantly more precise obfuscated locations than $\text{Sens}_{Pyr}$.

The graph in Figure 10 shows that the generalization time increases linearly with the size of the grid for both algorithms. Moreover, such a time is not significantly affected by the coverage for coverages that are not close to the breaking point. Note that the generation time is high (few seconds) when the grid is 4096 × 4096. Consider, however, that these experiments have been run on a laptop.

Experiment 5: Obfuscation ratio. The outcome is the obfuscation ratio of an obfuscated map, that is the ratio of the total number of cells contained in obfuscated locations over the total number of sensitive cells. Figure 11 reports the result of the experiment for three different privacy thresholds and a varying average coverage value. For example, for a privacy threshold equal to 0.1 and coverage equal to 2%, $\text{Sens}_{Pyr}$ generates an average of about 18 obfuscated cells for each sensitive cell, whereas $\text{Sens}_{Hil}$ presents an obfuscation ratio which is about 10. It is important to observe that the obfuscation ratio of the maps generated $\text{Sens}_{Hil}$ is almost constant and independent of the average coverage. Notably, such obfuscation ratio is almost always equal to the best attainable one, i.e., no algorithm can obfuscate the same sensitive regions and obtain a smaller obfuscated area.

6. RUN-TIME PRIVACY ENFORCEMENT

We consider now the size of the obfuscated map sent to the client. We recall that the client requests an obfuscated map by forwarding a request to the Obfuscation Engine. Such a request contains the privacy profile. The privacy profile is a set $P = \{t_1...t_n\}$ of pairs representing the privacy preferences, with $t_i \equiv< ft, v >$, $i \in \{1..n\}$. The user can select the sensitive feature types for example from a predefined list. In order to limit the size of the obfuscated map, the user can specify the bounding box of the region of interest. The Obfuscation Engine generates an obfuscated map, if it exists. Such map consists of a non-empty set of obfuscated regions, where each region is represented by an interval $[a, b]$ with $a$ and $b$ Hilbert indexes. The obfuscated map is then stored on the client as a B-tree.

Regarding the size of map being generated, if the encoding of the interval representing an obfuscated region requires 8 bytes, the size of an obfuscated map is $n \times 8$ bytes, where $n$ is the number of regions in the obfuscated map. Based on the experiments reported in the previous section, Table 3 reports the average size of the obfuscated maps generated for grids of varying size, assuming a 10% coverage.

<table>
<thead>
<tr>
<th>Grid size</th>
<th># Obfuscated locations</th>
<th>≈ Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512 × 512</td>
<td>2835</td>
<td>22</td>
</tr>
<tr>
<td>1024 × 1024</td>
<td>11372</td>
<td>90</td>
</tr>
<tr>
<td>2048 × 2048</td>
<td>45483</td>
<td>363</td>
</tr>
<tr>
<td>4096 × 4096</td>
<td>181920</td>
<td>1455</td>
</tr>
</tbody>
</table>

Table 3: Avg. size of the obfuscated maps

6.1 Privacy Enforcement

At run time, the client computes the location information to be transferred to the LBS provider. This operation is referred to as privacy enforcement. The correspondent algorithm is reported in Algorithm 3. Given user’s position $p$, and obfuscated map $S$, the algorithm maps $p$ onto a Hilbert index and then checks whether such a value is included in an interval of the indexed obfuscated map(line 4). Because the obfuscated map can be stored as a B-tree, the complexity of the operation is $O\left(\log n\right)$. If $p$ does not fall inside any interval, then the function simply returns $p$ otherwise the enclosing interval $r$. The result is then transferred to the LBS provider possibly along with the parameters of the Hilbert-space filling curve.

7. CONCLUSIONS

PROBE is a comprehensive system for the protection of location privacy against location inference attacks in LBS. A key feature of the system is that it allows the subscribers of a LBS to specify location privacy preferences about the places that they consider sensitive as well as the desired degree of privacy protection. As part of PROBE we have also developed a technique for efficiently computing obfuscated maps that are personalized based on the user privacy...
Figure 7: Success rate

Figure 8: Avg. number of obfuscated regions

Figure 9: Avg. number of cells per obfuscated region

Figure 10: Avg. time for varying-size grids and different coverages.

Figure 11: Avg. number of obfuscated cells per sensitive cell
REFERENCES


