Policy-Aware Sender Anonymity in Location Based Services

Alin Deutsch *, Richard Hull †, Avinash Vyas *, Kevin Keliang Zhao *
* UC San Diego {deutsch,avyas,kezhao}@cs.ucsd.edu
† IBM Research hull@us.ibm.com

***Abstract*— Sender anonymity in location-based services (LBS) attempts to hide the identity of a mobile device user who sends requests to the LBS provider for services in her proximity (e.g. "find the nearest gas station" etc.). The goal is to keep the requester's interests private even from attackers who (via hacking or subpoenas) gain access to the request and to the locations of the mobile user and other nearby users at the time of the request. In an LBS context, the best-studied privacy guarantee is known as *sender k-anonymity*. We show that state-of-the art solutions for sender k-anonymity defend only against naive attackers who have no knowledge of the anonymization policy that is in use. We strengthen the privacy guarantee to defend against more realistic "policy-aware" attackers. We describe a polynomial algorithm to obtain an optimum anonymization policy. Our implementation and experiments show that the policy-aware sender k-anonymity has potential for practical impact, being efficiently enforceable, with limited reduction in utility when compared to policy-unaware guarantees.**

I. Introduction

Recent years have witnessed increased demand for Location-Based Services (LBS), which answer requests of mobile device users for services in their proximity (e.g. "find the nearest gas station", "Thai restaurant", "hospital"). While some such LBS providers are available in wireless networks since 2001 [1], their proliferation has been limited, among other reasons, by privacy concerns.

In this paper we address one such concern, which pertains to hiding the identity of the sender of more sensitive requests (e.g. for the local campaign headquarters of a given political party, spiritual center for a given religion, etc.) in order to keep her interests private. The sender's identity must be protected even against attackers who, via hacking or subpoenas, gain access to *a)* the request (from the LBS provider's log) and *b)* to the locations of the mobile user and other nearby users at the time of the request (from the wireless service provider) and *c)* who know the "design" of the system used to provide this protection. The assumption *c)* about the attackers is based on a well accepted principle of designing a private and secure system - "The design is not secret" [24]. This is indeed a realistic threat since an attacker with subpoena powers (e.g. a federal agency) or a disgruntled ex-employee can obtain the "design" of the system.

In the context of LBS, the best-studied identity protection measure is known as *sender k-anonymity* [17], which is intended to guarantee that the request log and precise location information are insufficient to distinguish among the actual requester and k-1 other possible requesters. Typical sender anonymization algorithms [16], [23], [17], [27] are based on hiding the sender's precise location in the request, substituting instead a *cloak*, i.e. a region containing this location. The cloak is usually chosen from among regions of a pre-defined shape (circular, rectangular, etc.), to include at least k-1 other mobile users. To maximize the utility of the answer to the service request, usually the tightest cloak containing k users is picked. We refer to these utility-maximization policies of choosing cloaks as *k-inside*.

In the version of sender k-anonymity provided by the k-inside policies [16], [23], [27], the principle that the design of a system is not secret is ignored. Therefore, such privacy guarantee does not hold against an attacker who knows the "design" i.e. the policy used to pick the cloaks for locations. The next example illustrates how an attacker who knows the cloaking policy ("policy-aware" attacker) can identify the sender when the cloaks are selected using a k-inside policy.

Example 1. *Figure I shows the 2-inside policy obtained using the algorithm described in [23] for the location database of Table I. The algorithm assumes a static quad-tree based partitioning of a geographic space and uses quadrants and sub-quadrants (combination of two adjacent quadrants) as possible cloaks. For a given requester, the algorithms picks the smallest cloak (containing the requester) that contains k-1 other users. For k=2 the algorithm cloaks* A *and* B *to* R_1, C *to* R_3 *and* S *and* T *to* R_2. *Since each of these cloaks contains at least 2 locations this is a 2-inside policy. Assume there is an attacker that has access to the location database* D_1 *(via hacking or subpoena) and is "design-aware" i.e. knows the 2-inside policy used to provide sender 2-anonymity. If this attacker observes an LBS service request with cloak* R_3, *he can identify the sender as C!.* □

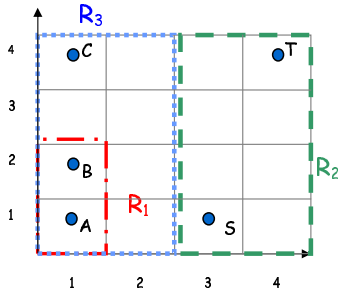


Fig. 1. 2-inside policy

userid	locx	locy
...	...	...
Alice	1	1
Bob	1	2
Carol	1	4
Sam	3	1
Tom	4	4
...	...	...

TABLE I
Location Database D_1

The privacy guarantee of the k-inside policy have been refined

by additional constraints such as *k-reciprocity*[17] and *k-sharing* [11]. We show (in Section VII) that since the policy-aware attacker is not considered, these additional constraints also fail to provide sender k-anonymity.

As described later in detail, to preserve sender k-anonymity against a policy-aware attacker, in some cases the cloak used for a location needs to be bigger (and include more than k locations) than the cloak picked by a k-inside policy. As a result, a cloaking that provides policy-aware sender k-anonymity (*policy-aware cloaking*) may have reduced utility in comparison to a k-inside policy. Moreover unlike k-inside policies where one can find the utility-maximizing (optimum) cloaking for each user by considering a small subset of all the users, for optimum policy-aware cloaking one has to consider all the users (as described later in Section IV), which is computationally more expensive than optimum k-inside policy. Hence policy-aware sender k-anonymity trades utility and performance for stronger privacy. In this paper, we describe our findings on identifying the "sweet spot" in this tradeoff.

Our contributions. In addition to showing that k-inside policies achieve sender anonymity only against attackers who are policy-unaware, and is not proof against policy-aware attackers, our contributions include the following.

[1] We formalize the classes of policy-unaware and policy-aware attackers, and define a novel, stronger privacy guarantee: *sender anonymity against policy-aware attackers*. We prove formally that this guarantee strictly subsumes sender anonymity against policy-unaware attackers.

[2] We study the problem of finding, among all policy-aware sender k-anonymizations of a set of mobile users, one with optimum utility. We show that the problem of finding optimum policy-aware sender k-anonymity depends upon the type of cloaks used. In particular, we show that when the cloaks are circles whose centers are selected from a given set of points, the problem is NP-complete, but becomes PTIME for cloaks picked from among the quadrants of a quad-tree-based partition of the map (a common choice in state-of-the-art anonymization solutions [16], [23]).

[3] We implement and evaluate experimentally our optimum policy-aware anonymization algorithm. Even though finding optimum policy-aware anonymization is computationally costly in comparison to finding optimum k-inside policy (that uses the same cloak types) we show that our algorithm is practical and scales extremely well with the number of service requests: it takes less than 1 second to anonymize 250k requests from users in the San Francisco Bay area (using a single anonymization server) and can scale up to 1 million requests using 16 servers in parallel.

[4] As stated earlier, the policy-aware cloaking may result in some loss of utility in comparison to a k-inside policy (that uses the same cloak type). We show empirically that the utility reduction traded for the stronger privacy guarantee is reasonable: the average cloak area is at most 1.7 times the average area of the tightest cloaks used for policy-unaware anonymity.

Scope of the paper More recently, several extensions to sender k-anonymity has been proposed, such as allowing *user specified k* (in [14], [11]) and defending against *trajectory-aware attacker* [6], [27], [11] where the attacker has knowledge of when multiple requests have originated from the same (a priori unknown) user, even if they are sent at different times and from different locations. While these extensions are important, our work improves upon the foundations of these extensions, namely make it policy-aware. We leave as future work the extension of the policy-aware sender k-anonymity to handle trajectory-awareness and user-specified k.

Paper outline The remainder of the paper is organized as follows. In Section II, we show how we model an LBS. We define sender anonymity and the classes of attacks it defends against in Section III. Section IV gives our PTIME algorithm for finding a policy-aware anonymization of maximum utility. In Section V we describe how we utilize the inherent parallelism in the problem to obtain greater scalability and report on the experimental evaluation in Section VI. We discuss related work in Section VII and conclude in Section VIII.

II. LBS Models

This section introduces a basic model of providing location based services, based on information about user locations provided by a wireless network. It then describes modifications and additional components required for privacy support.

A. Basic LBS Model

Wireless "Communications Service Providers" (CSPs) can derive the approximate location of user devices, through a variety of mechanisms, including triangulation based on signal strength or time-delay to multiple cell towers, and GPS capabilities on the device. In the US, the E911 Requirement [3] mandates that CSPs provide the necessary infrastructure to determine the location of mobile devices within a range of 50 to 300 meters, depending on the technology used. This infrastructure must be available when users call the emergency 911 number, but can also be used to support other services, including location-based information services. It is now common for CSPs to include specialized network components, which are called *Mobile Positioning Center* (MPC) in the CDMA standard, that serve as a logically centralized point that provides access to device locations for E911 and other location based services.

An abstract model is used here to study location-based services and their privacy characteristics. For simplicity, we model a geographic area as a 2-dimensional space and user's location as integer coordinates within this 2-dimensional space. There are four core elements in the delivery of a location-based service: the user making a request, typically called the *sender*, the (wireless) Communication Service Provider, denoted as *CSP*, the Mobile Positioning Center operated by the CSP, denoted as *MPC*, and the Location Based Service (LBS) provider, denoted as *LBS*. We view the CSP to be a trusted agent that operates the MPC. Although in practice the MPC provides the approximate location of each user's device, for simplicity we take the value produced by the MPC as the device's exact location. A sender's request for a location-based service is processed by the CSP, which obtains the user's location from the MPC and forwards the request to the LBS.

We abstract from the fact that location is usually determined only on demand, and assume in our investigation that the locations of all devices are eagerly computed and available. This eagerness assumption is appropriate in connection with the study of privacy guarantees, since we target attackers who might be able to reconstitute all device locations, perhaps by hacking in real-time, by hacking logs, or by subpoena-induced cooperation of the CSP.

Location Database In the abstract model, for simplicity we assume that the device locations made available by the MPC are stored in a relational database, called the *location database*. (This database might be virtual.) Although its actual schema can vary from CSP to CSP, it is essentially equivalent to a single relation schema

$$\mathbf{D} = \{userid, locx, locy\}.$$

Here, the domains of attributes $locx$ and $locy$ are the domains of x and y coordinates in the 2-dimensional space used to model the geographic region.

In the current paper, we assume that the location database is updated periodically (e.g., every 30 seconds) to reflect the movement of users. Multiple location-based requests can be made against each snapshot. Thus the state of a location database over a period of time can be modeled as a sequence of different instances of schema **D**.

We represent the set of all possible instances of **D** by $\mathcal{D}$. An example instance $D_1 \in \mathcal{D}$ is shown in the Table I, and illustrated diagrammatically in Figure I.

The following definition allows us to focus on the precise information associated with a sender's request for a location-based service. (In the following section we describe how this request might be modified by the CSP to provide privacy protections before forwarding to the LBS.)

Definition 1. A *service request* is a tuple $\langle u, (x, y), V\rangle$ where u is a sender identifier, (x, y) are coordinates in 2-dimensional space and V is a vector of name-value pairs. We say that the service request is *valid* w.r.t a location database D if $\langle u, x, y\rangle \in D$.

Intuitively, the name-value pairs contain the categories and specifics of the sought services.

We define the function $id(SR)$ that returns the user id in the service request and another function $loc(SR)$ that returns the location co-ordinates (x, y).

Although a service request SR itself is created by the CSP, based on a request from a sender u, we sometimes refer to SR as having been sent by u.

Example 2. *The following are examples of service requests sent respectively by users* Alice*,* Bob*,* Carol*,* Sam*, and* Tom*:*

SR_a = $\langle$Alice$, (1, 1), [(poi, rest), (cat, ital)]\rangle$,
SR_b = $\langle$Bob$, (1, 2), [(poi, groc), (cat, asian)]\rangle$,
SR_c = $\langle$Carol$, (1, 4), [(poi, rest), (cat, ital)]\rangle$,
SR_s = $\langle$Sam$, (3, 1), [(poi, rest), (cat, ital)]\rangle$,
SR_t = $\langle$Tom$, (4, 4), [(poi, cinema), (cat, drama)]\rangle$.

*(Here "*poi*" stands for "point of interest", "*cat*" stands for "category", "*rest*" stands for "restaurant", etc.) All five service requests are valid w.r.t. the location database instance D_1 of Table I.* □

In the abstract model, the service requests are created by the CSP using a combination of a request for information from a user, along with the user's location as provided by the MPC. We will therefore assume for our ongoing discussion that each service request is valid w.r.t. the current location database instance.

B. Privacy-conscious LBS Model

In realistic privacy solutions, the goal is not to hide information from everybody, but rather to minimize the number of parties one needs to trust to achieve the desired communication. The fundamental assumption underlying the privacy-conscious LBS model studied here is that the CSP is a trusted party, and nobody else is. In particular, the LBS is not trusted, reflecting the fact that it is usually a third-party provider that is not under the CSP's control.

We extend the basic LBS model of Subsection II-A to support mobile users in accessing the LBS without revealing their identity to anyone except for the CSP. Users rely on the CSP to ensure this goal. While we assume that the CSP can be trusted to perform the privacy-ensuring computations and not log them, we will assume that attackers may be able to obtain information about the locations of individual users at different times. (This might arise due to hacking, or to subpoenas, if the CSP is logging user locations for the purposes of advertising or service personalizations.) For the worst case, then, we assume that the sequence of location databases is available to the attacker.

In the privacy-conscious LBS model, a user sends a location-based request to the CSP over a channel that is assumed to be secure (this is typically the case in cell phone networks). The CSP constructs the corresponding service request SR, and based on this, will send a request on behalf of the user to an LBS L. The CSP cannot sent SR itself, because this includes the sender's identity, which would be revealed to both the LBS provider (which may be an attacker) and to any potential attackers listening on the channel. Also, simply removing the sender identity from SR does not suffice, because the identity can be obtained by examining the location database (which is assumed to be available to attackers). While there are many approaches to anonymize a request SR so that it does not reveal the requester's identity, for the current investigation we shall use the following classical one.

Definition 2. An anonymized request is a tuple $\langle rid, \rho, V\rangle$ where rid is a unique request identifier, ρ is a connected, closed region in the plane, and V is a vector of name-value pairs. If ρ is a rectangular region with vertical and horizontal sides, then we also denote this anonymized request as

$$\langle rid, (x_1, y_1, x_2, y_2), V\rangle$$

where (x_1, y_1) ((x_2, y_2)) specifies the southwest (respectively northeast) corners of a rectangular region.

From now on we refer to these regions in anonymized requests as *cloaks*. We also define a function *reg(AR)* that returns the cloak from an anonymized request AR.

Example 3. *The following is a list of anonymized requests, whose cloaks are depicted in Figure I.*

AR_a = $\langle 167, (0,0,1,2), [(poi, rest), (cat, ital)]\rangle$
AR_b = $\langle 168, (0,0,1,2), [(poi, groc), (cat, asian)]\rangle$
AR_c = $\langle 169, (0,0,2,4), [(poi, rest), (cat, ital)]\rangle$
AR_s = $\langle 170, (2,0,4,4), [(poi, rest), (cat, ital)]\rangle$
AR_t = $\langle 171, (2,0,4,4), [(poi, cinema), (cat, drama)]\rangle$ □

Definition 3. We say that an anonymized request $AR = \langle id, \rho, V\rangle$ *masks* a service request $SR = \langle u', (x', y'), V'\rangle$ if the location (x', y') is an element of the cloak ρ and V=V'.

Example 4. *For each x in {a, b, c, s, t}, the anonymized request AR_x of Example 3 masks the service request SR_x of Example 2.* □

Instead of sending a service request SR to the LBS provider, the CSP forwards to the LBS provider an anonymized request that masks SR. The next section discusses how such anonymized requests are constructed.

III. Policy-aware k-anonymity

Prior research considers anonymization algorithms that cloak the sender's location with a region covering the location of $k-1$ additional mobile users [16], [23], [27]. The intention is that an attacker who observes the anonymized request and has access to the location database can not reduce the number of possible senders below k, since there are k potential service requests, one for each of the senders covered by the cloak of the anonymized request, that could have lead to the same anonymized request. While the cloaking algorithms proposed in the literature use different cloak shapes (quadrants [16], [23], minimum bounding circles [27], etc.) they agree in one important aspect: to maximize utility of the service, the tightest cloak that includes k users is picked. We term this class of cloaking policies as *k-inside* policies.

The results in this section are motivated by the observation (described in Section I) that if the attacker is aware of the k-inside policy used by the CSP, then for some location databases he is able to reduce the number of possible senders below k, defeating the purpose of anonymization. We set out to defend against such "policy-aware" attackers (for general classes of cloaking policies, including but not limited to the k-inside policy). To this end we need to formalize the classes of policy-aware and -unaware attackers. In turn, this requires the formalization of the notion of cloaking policy, as the CSP's method of obtaining an anonymized request AR from a service request SR and a given location database instance D.

Definition 4. A *policy* is a deterministic procedure P that takes as input a location database instance D and a service request SR, and outputs an anonymized request AR:

P : {instances of location database} × {service requests} → {anonymized requests}.

A policy P is *masking* if for every service request, the location specified in the service request lies within the cloak in the anonymized request it is mapped to. Formally,

$$\forall D\ \forall SR\ loc(SR) \in reg(P(D, SR)).$$

In this investigation we consider only masking policies, so from now on we use the term policy to refer to a masking policy.

Example 5. *Assume that the current location database instance is D_1, as shown in Table I. The policy P_1 for the service requests shown in Example 2 is as follows:*

$$P_1(D_1, SR_a) = AR_a \qquad P_1(D_1, SR_s) = AR_s$$
$$P_1(D_1, SR_b) = AR_b \qquad P_1(D_1, SR_t) = AR_t$$
$$P_1(D_1, SR_c) = AR_c$$

where $AR_a, AR_b, AR_c, AR_s, AR_t$ are the anonymized requests of Example 3 and the cloaks used in these anonymized requests are shown in Figure I (where $reg(AR_a) = reg(AR_b) = R_1$, $reg(AR_c) = R_3$ and $reg(AR_s) = reg(AR_t) = R_2$). □

The Attacker Model We now proceed to formalizing the privacy guarantee of policy-aware sender k-anonymity. To this end, we need a framework for describing what an attacker knows about the policy being used by the CSP to anonymize requests. At the one extreme, an attacker may know exactly which policy is being used; as formally defined below these will be called "policy-aware" attackers. At the other extreme of interest here, an attacker may know only that the policy is based on the use of some family $\mathcal{C}$ of possible cloaking regions (e.g. rectangles, quadrants of a given quad tree, circles with center from a given set). Given such a family $\mathcal{C}$, we let $\mathcal{P}_\mathcal{C}$ denote the set of all policies that use cloaking regions from $\mathcal{C}$. As formally defined below, attackers who know only that the policy used is an element of $\mathcal{P}_\mathcal{C}$ for some set $\mathcal{C}$ will be called "policy-unaware (relative to $\mathcal{C}$)".

We target a strong, information-theoretic definition of privacy. To this end, we model attackers as a function taking certain input to launch the attack. There are no limiting assumptions on the computational resources expended to compute this function. The only assumptions are on what input the function has (intuitively, the information that the attacker sees). We classify this input into two groups as follows.

Design time: Even before the attacker observes any anonymized request, he may know

- the targeted level k of sender k-anonymity, and
- the family of candidate policies $\mathcal{P}$ (which in our study is typically either a singleton, or a set $\mathcal{P}_\mathcal{C}$ for some family $\mathcal{C}$ of cloaking regions)

Run time: The attacker can observe (or reconstruct after the fact, via log hacking or subpoenas)

- the instance D of the location database (corresponding to a snapshot of all of the sender locations), and
- the set of anonymized requests made against this snapshot.

Notice that hacking attacks may be unable to reconstruct the entire location database. If sender k-anonymity is provided under the above assumptions, then it is also provided if the attacker has only partial knowledge of D.

The attack function models the following attack: starting from the observation of a set A of anonymized requests and the full knowledge of the location database D, the attacker "reverse engineers" the anonymized requests to obtain the possible service requests masked by A and compatible with the candidate

policies in $\mathcal{P}$. We capture this result of the attack by defining the notion of *Possible Reverse Engineering (PRE)* of a set of anonymized requests.

Definition 5. Consider a family of policies $\mathcal{P}$, a location database D and a set of anonymized requests $A = \{AR_i\}_{1 \leq i \leq n}$. A *Possible Reverse Engineering (PRE)* π of A w.r.t. D and $\mathcal{P}$ is a function from anonymized requests to service requests such that

- $\pi(AR_i)$ is valid w.r.t. D for all $1 \leq i \leq n$, and
- there exists some $P' \in \mathcal{P}$, such that $P'(D, \pi(AR_i)) = AR_i$ for each $1 \leq i \leq n$.

Intuitively, a PRE π associates with every anonymized request AR in A a possible service request that could have generated AR, based on some fixed policy P' from the family of candidates $\mathcal{P}$. We represent the set of all PREs of a set A w.r.t. D and $\mathcal{P}$ as $PRE(A, D, \mathcal{P})$.

Sender k-anonymity We are now ready to define sender k-anonymity. Intuitively, this will capture the property that, even if the attacker uses the available information to flawlessly compute (no matter at what computational cost) all PREs of the observed set of anonymized requests, these PREs still point to at least k possible senders for each anonymized request. We consider it a breach of sender k-anonymity if the attacker succeeds in reducing the set of possible senders to fewer than k. We first define sender k-anonymity as a property of a set A of anonymized requests w.r.t. a location database D and a family of policies $\mathcal{P}$. Since the anonymized requests are obtained using a policy P, it is easy to extend the definition as a property of a policy P.

Definition 6 (Sender k-Anonymity). Let $\mathcal{P}$ be a family of policies and D a location database. Let A be a finite set of anonymized requests obtained using a policy $P \in \mathcal{P}$. We say that A provides *sender k-anonymity against $\mathcal{P}$-aware attackers on D* if there are PREs $\pi_1 \ldots \pi_k \in PRE(A, D, \mathcal{P})$ such that for each $AR \in A$ and each pair i, j satisfying $1 \leq i < j \leq k$, $id(\pi_i(AR)) \neq (id(\pi_j(AR))$.

We say that policy P provides *sender k-anonymity against $\mathcal{P}$-aware attackers on D* if for each finite set S of service requests (valid w.r.t. D), the set of anonymized requests $\{P(D, SR) \mid SR \in S\}$ provides sender k-anonymity against $\mathcal{P}$-aware attackers on D.

We say that P provides *sender k-anonymity against $\mathcal{P}$-aware attackers* if for every location database D, P provides sender k-anonymity against $\mathcal{P}$-aware attackers on D.

Since our attacker model is parameterized by the family of candidate policies $\mathcal{P}$ and the set of observed anonymized requests A, by varying these sets one can enumerate different classes of attackers and the corresponding flavors of sender anonymity. In this paper we focus on two extremes.

- A *policy-unaware* attacker (relative to family $\mathcal{C}$ of possible cloaking regions) does not know which particular cloaking policy in $\mathcal{P}_\mathcal{C}$ is used by the CSP, and observes only one anonymized request.
- A *policy-aware* attacker knows the specific policy P used by the CSP, and is able to observe and memorize all anonymized requests.

We show next that the class of policy-aware attackers is strictly more powerful (in terms of breaching sender k-anonymity) than the class of policy-unaware attackers.

Example 6. *Let's revisit Example 1 of Section I. It can be easily observed that for the users of Table I the policy P_1 in Example 5 is based on the cloaking described in Example 1 When the policy-unaware attacker observes AR_c and tries to reverse engineer the service requests that could have generated it. He finds 3 PREs $\pi_1(AR_c) = SR_c$ and $\pi_2(AR_c) = \langle Alice, (1,1), [(poi, rest), (cat, ital)]\rangle$ and $\pi_3(AR_c) = \langle Bob, (1,2), [(poi, rest), (cat, ital)]\rangle$ with distinct users, Alice, Bob and Carrol. So policy P_1 provides sender 2-anonymity against policy-unaware attackers on D_1. In contrast, the $\{P_1\}$-aware attacker who observes AR_c can construct only one PRE, involving Carol, whose identity is completely compromised. Thus the $\{P_1\}$-aware attacker breaches sender 2-anonymity in a case when policy-unaware attacker cannot.* □

In the remainder of the paper we target an anonymization algorithm that preserves sender k-anonymity against the class of policy-aware attackers. Such an algorithm will also defend against policy-unaware attackers. This claim is formalized below (for proof see[12]).

Proposition 1. *Let A be a set of anonymized requests obtained using policy P on location database D. If A provides sender k-anonymity against a policy-aware attacker on D, it also provides sender k-anonymity against a policy-unaware attacker on D.*

Sender k-Anonymity and k-inside Policies We first check the privacy provided by some of the cloaking algorithms proposed in the literature [16], [23], [27] against the two classes of attackers introduced above. As it turns out, they only defend against policy-unaware attackers.

Recall that all of these algorithms implement a k-inside cloaking policy, which we use to obtain generic results that hold for all algorithms in this class.

The following example shows a 2-inside policy that provides sender 2-anonymity against policy-unaware attackers.

Example 7. *For location database instance D_1 of Figure I, the policy P_1 of Example 5 uses cloaks R_1, R_2 and R_3 to anonymize the service requests. Since each of these cloaks contains at least 2 locations, P_1 is a 2-inside policy. Now recall from Example 6 that the policy-unaware attacker could not reduce the set of possible senders of each request to less than 2.* □

We can show that the finding in Example 7 is not accidental:

Proposition 2. *A k-inside policy provides sender k-anonymity against policy-unaware attackers.*

In contrast, recall that Example 6 illustrates a case in which a *k-inside* policy does not provide sender k-anonymity against a policy-aware attacker, leading to the following claim (for proof see[12]).

Proposition 3. *Not all k-inside policies provide sender k-anonymity against policy-aware attackers.*

Since by Proposition 3, the prior anonymization algorithms do not satisfy our goal of defending against policy-aware attackers, we need to search for a novel algorithm.

Before presenting this algorithm in Section IV, we illustrate a policy that does provide sender 2-anonymity against policy-aware attackers.

Example 8. *For the location database instance D_1, we describe a policy P_2 for the service requests shown in Example 2:*
$P_2(D_1, SR_a) = \langle 167, R_3, [(poi, rest), (cat, ital)]\rangle$,
$P_2(D_1, SR_b) = \langle 168, R_3, [(poi, groc), (cat, asian)]\rangle$,
$P_2(D_1, SR_c) = \langle 169, R_3, [(poi, rest), (cat, ital)]\rangle$,
$P_2(D_1, SR_s) = \langle 170, R_2, [(poi, rest), (cat, ital)]\rangle$, *and*
$P_2(D_1, SR_t) = \langle 171, R_2, [(poi, rest), (cat, thai)]\rangle$.
The cloaks R_2 and R_3 used in these anonymized requests are those depicted in Figure I. The readers can check that P_2 provides privacy against $\{P_2\}$-aware attackers since for each anonymized request one can construct 2 PREs using policy P_2. □

IV. Optimal k-anonymity

For the same location database, there may exist several policies that provide policy-aware sender k-anonymity, raising the obvious question of which one to use. In this section we address the problem of finding the policy of highest utility to the users. Prior work on policy-unaware anonymity proposes that one way to maximize utility is to minimize the cloak area. A smaller cloak allows for more efficient processing of range queries (e.g. find gas stations within 2 miles) at the LBS as well as more efficient filtering of results at clients. Since we don't know a priori the users who are going to send a request at a given snapshot of location database, we compare policies for the case when every user sends a request.

Cost of a policy We introduce the cost of a policy to quantitatively capture the fact that the utility is maximized as the cloak area is minimized. We define the *cost of an anonymized request AR* as the area of its cloak $reg(AR)$. Given a location database D and a set S of service requests valid w.r.t. D, the *cost of S under P* is defined as $\sum_{SR\in S}$ cost of $P(D, SR)$. The *cost of a policy P on D*, denoted $Cost(P, D)$, is computed as the cost of the set of service requests obtained if every user in D issues precisely one request (of immaterial parameters); it is the cost of the set of service requests

$$S = \{\langle u, (x, y), V\rangle |\ (u, x, y) \in D\}$$

where V is some arbitrary vector of name-value pairs.

Optimal policy We next focus on the problem of obtaining, for a given location database D, an optimal (cost-minimal) policy that provides sender k-anonymity against policy-aware attackers. We show that the complexity of this problem depends upon the type of cloaks used in the anonymization. In particular we show that the problem is NP-complete if the cloaks are of circular shape, and are picked by choosing the center from a given set of points (e.g. public landmarks such as libraries, train stations or cell towers) and by choosing the radius freely. It comes therefore as a pleasant surprise that the problem becomes PTIME for a version in which cloaks are picked among the quadrants of a quad-tree-based partition of the map.

We first detail the circular-cloak version of the problem. Let D be an instance of location database and SC be a set of possible centers. Find a policy-aware sender k-anonymous policy P that minimizes $Cost(P, D)$. P uses circular cloaks, each centered at some point from SC, with no restriction on the radius. We call this problem *Optimal Policy-aware Bulk-anonymization with Circular cloaks*. We find the following negative result.

Theorem 1. *Optimal Policy-aware Bulk-anonymization with Circular cloaks is NP-Complete.*

Note that the NP-completeness is in the size of the location database, meaning that optimal policy-aware anonymization is practically infeasible.

There is good news, however, if we consider a different type of cloaks from which the policy may choose. This result has high practical impact, since the cloak type in question is already widely used in the literature. We consider cloaks picked from among the quadrants corresponding to a quad-tree-based partitioning of a planar area. The quad tree is a well-known structure for organizing spatial data, and it has been used in a number of anonymization solutions [16], [23]. As the name suggests, it is a tree in which every non-leaf node has exactly 4 child nodes. In a quad tree representation of a (square shaped) map, the root node represents the entire map. The region is then divided equally into 4 non-overlapping square quadrants, each of whom represents a child node of the root. Each quadrant is then again divided into 4 equal sub-quadrants that correspond to grandchildren of the root. This four-way splitting goes on until the desired level of granularity for the minimum region is reached.

A policy that anonymizes locations to cloaks represented by nodes of the quad-tree representation of a given map is known as *quad-tree* policy.

This brings us to our main finding.

Theorem 2. *An optimal quad-tree policy providing sender k-anonymity against policy-aware attackers can be found in PTIME.*

In the remainder of this section, we describe a PTIME algorithm to find an optimal quad-tree policy.

A. Reducing the Policy Search Space

Given a map with its associated quad tree T, and a location database D, it is easy to see that the space of all quad-tree policies cloaking locations in D by nodes in T is exponential in the size of D. This rules out solutions based on enumerating all policies.

Intuitively, the following key observation reduces the search space to polynomial size: both the property of being policy-aware sender k-anonymous, and the cost of the policy depend only on *how many* locations are cloaked by each node N of the quad tree T, being indifferent to *which* particular locations are cloaked by N. Calling policies equivalent if every quad tree

node cloaks the same number of locations under both policies, one need not enumerate individual policies, enumerating policy equivalence classes instead. It turns out that only polynomially many such classes need to be inspected.

Equivalent Policies We formalize this intuition next. Given location database D and quad tree T, two policies are *equivalent under* D, T if every node N of T cloaks the same number of locations from D under both policies. When D and T are clear from the context, we may not mention them. The following justifies why we need not discriminate among equivalent policies.

Lemma 1. If policies P_1, P_2 are equivalent under D, T, then

(a) P_1 and P_2 have the same cost; and
(b) P_1 provides policy-aware sender k-anonymity on D if and only if so does P_2.

B. A First-Cut Algorithm

For simplicity of exposition, we start by presenting a first-cut PTIME algorithm derived in the most direct way from the insight that equivalence classes suffice. We leave its optimization to [12].

Configurations. The first-cut algorithm manipulates equivalence classes of policies. It represents an equivalence class by keeping track for each quad tree node of the number of locations it cloaks. For technical convenience, this is done by equivalently tracking for each node N the number of locations that are located within N yet are *not* cloaked by N or any of its descendants. It is easy to translate between the two equivalence class representations.

Definition 7. Let D be a location database, and T be a quad tree rooted at node r. Let $d(m)$ denote the total number of locations from D that occur in the quadrant m. A *configuration* C of T is a function from the nodes of T to natural numbers, such that

(i) for every leaf node m in T, $C(m) \leq d(m)$; and
(ii) for every internal node m, $C(m) \leq \sum_{i=1}^{4} C(m_i)$, where $m_1 \dots m_4$ are the children of m.

We say that C is *complete* if $C(r) = 0$.

Item (i) of Definition 7 simply states that a node can be used to cloak at most as many locations as contained in its quadrant (since we only consider masking policies). Item (ii) states that the number of locations not cloaked by m's children is higher than the number of locations not cloaked by m, which is an immediate consequence of the fact that each child quadrant is contained in its parent.

Note that, given a policy, location database D and quad tree T, and a configuration C for T, we can exhibit in linear time one of the policies C represents (by arbitrarily selecting the $C(m)$ locations for each node m). The first-cut algorithm's strategy is to find a minimum-cost configuration, then exhibit (in linear time) one of the policies represented by it, picked nondeterministically.[1] Note that a configuration is exponentially more succinct than an explicit listing of the policies it represents; if we focus on any node m alone, there are exponentially many ways to pick $C(m)$ locations among those occurring in m.

Before explaining how the desired configuration is found, we address two technical issues that need to be solved to allow the first-cut algorithm to manipulate configurations without materializing policies (except for outputting the result). First, how to compute the cost of the represented policies without materializing them. Second, how to check if a configuration corresponds to policy-aware sender k-anonymous policies.

Computing Cost from Configurations. We define the cost of a configuration as the cost of the policies it represents (uniquely defined by Lemma 1(a)). This cost can be computed without materializing any represented policy, using the following function.

Definition 8. Let D be a location database instance and C be a configuration of a quad-tree T. We define the *cost of* C *on* D, denoted $Cost_c(C, D)$, as

$$Cost_c(C, D) := \sum_{n \in nodes(T)} f(n, C)$$

where $f(n, C)$ is given by

$$f(n, C) = \begin{cases} (d(n) - C(n)) \times area(n), & \text{n is leaf} \\ ((\sum_{i=1}^{4} C(n_i)) - C(n)) \times area(n), & \text{n is internal} \end{cases}$$

where $n_1 \dots n_4$ are the children of n and area(n) is the area of the quadrant corresponding to quad node n.

We can show that the configuration cost is precisely the cost of the represented policies:

Lemma 2. *Given a location database D, a quad tree T, a quad-tree policy P based on T and a configuration C representing P's equivalence class, we have $Cost_c(C, D) = Cost(P, D)$.*

Checking Sender Anonymity from Configurations. We turn to checking if the policies in the equivalence class represented by a given configuration are policy-aware sender k-anonymous, without materializing them. By Lemma 1(b), either all represented policies qualify, or none does. It turns out that it suffices to check directly that the configuration satisfies a property we call *k-summation*.

Definition 9. (k-summation) Let D be a location database instance and C a configuration of a quad tree T rooted at n. C satisfies *k-summation* if

- for a leaf node m
  - (i) if $d(m) < k$, then $C(m) = d(m)$.
  - (ii) if $d(m) \geq k$, then either $C(m) = d(m)$ or $C(m) \leq (d(m) - k)$.
- for an internal node m let $\Delta = \sum_{i=1}^{4} C(m_i)$, where $m_1 \dots m_4$ are the children of m
  - (iii) if $\Delta < k$, then $C(m) = \Delta$.
  - (iv) if $\Delta \geq k$, then either $C(m) = \Delta$ or $C(m) \leq (\Delta - k)$.

Intuitively, in Definition 9, clause (i) states that if node m's quadrant contains less than k locations, none of them can be cloaked by m lest k-anonymity be compromised. The cloaking

[1] For the sake of conciseness, in the following we overload the term policy to denote functions from user locations to cloaks (instead of from service requests to anonymized requests as in Definition 4). The two notions of policy are interreducible.

responsibility for all $d(m)$ of them is "passed up" to m's ancestors ($C(m) = d(m)$). By clause (ii), if there are at least k locations, then either all of them are passed up, or at most $d(m) - k$ (since at least k must be cloaked together to preserve k-anonymity). Δ represents the number of locations whose cloaking responsibility is passed up from m's children to m. If there are too few of them (less than k) then they cannot be cloaked by m, who in turn passes the responsibility to its ancestors (in clause (iii)). Otherwise, m has the choice of either cloaking none of them ($C(m) = \Delta$ in clause (iv)), or cloaking at least k and passing up at most $\Delta - k$.

Lemma 3. *Let T be the quad-tree representation of a map and D be an instance of the location database for that map. If C is a configuration of T and P a policy in the equivalence class C represents, then P is policy-aware k-anonymous on D if and only if C satisfies the k-summation property.*

Algorithm $Bulk_{dp}$**.** Lemmas 2 and 3 justify an algorithm that explores the space of configurations satisfying k-summation, in search for a complete minimum-cost configuration under $Cost_c$.

The exploration is carried out as follows. Recall from Definition 8 that the cost of configuration C of a quad tree T rooted at m depends only on the number $C(m)$ of locations not cloaked by T's nodes, and is independent of the cloaking at the nodes outside of T. For this reason, it suffices if the search space includes, for every quad tree node m, all possible numbers u of locations whose cloaking responsibility is passed up to m's ancestors. That is, all possible values u for $C(m)$. For each such pair (m, u), the minimum cost is computed among all possible configurations C' of T with $C'(m) = u$. To this end, the algorithm considers all possible counts $u_1, \ldots, u_4$ of locations passed up to m by its children $m_1, \ldots, m_4$, and recursively computes the corresponding minimum cost for each (m_i, u_i) pair.

Redundant cost re-computation for m, u pairs is avoided by storing the result in the corresponding cell of a bi-dimensional matrix M indexed by quad tree nodes and by values for u. To enable the easy retrieval of the min-cost configuration from M, the entries for node m carry, besides the minimum cost, some bookkeeping information relating to the configurations at the children of m.

This yields the following dynamic programming algorithm $Bulk_{dp}$ that, given quad tree T and location database D, fills in a *configuration matrix* M of dimension $|T| \times |D|$, where $|T|$ denotes the number of nodes in T and $|D|$ the number of locations in D. Each entry $M[m][u]$ in the matrix is a tuple of the form $\langle x, u_1, u_2, u_3, u_4 \rangle$, pertaining to a configuration C for the quad sub-tree rooted at m, such that $C(m) = u$, $Cost_c(C, D) = x$, and $C(m_i) = u_i$ where $m_1, \ldots, m_4$ are the children of m. The algorithm traverses the quad-tree T bottom-up starting from its leaf nodes, and for each node m and $1 \leq u \leq n$ fills in the entry $M[m][u]$ using the rows for $m_1, \ldots, m_4$.

Notice that it is easy to retrieve in polynomial time a minimum-cost complete configuration from M, by a top-down traversal of T. First, pick a minimum-cost entry in the row corresponding to the root of T. This entry lists for each child m_i of the root the value $C(m_i) = u_i$ leading to the minimum cost. Now inspect for each m_i the corresponding row in M, picking again a minimum-cost entry, and continue recursively until all leaf nodes are reached.

Algorithm 1 $Bulk_{dp}$

```
 1: for 1 ≤ m ≤ |T| do
 2:   for 1 ≤ u ≤ |D| do
 3:     M[m][u] := ⟨∞, 0, 0, 0, 0⟩ {initialize}
 4: for all node m ∈ T do
 5:   if (m is a leaf node) and (d(m) < k) then
 6:     M[m][d(m)] := ⟨0, 0, 0, 0, 0⟩
 7:   else if (m is a leaf node) and (d(m) ≥ k) then
 8:     M[m][d(m)] := ⟨0, 0, 0, 0, 0⟩
 9:     for 0 ≤ u ≤ d(m) − k do
10:       M[m][u] := ⟨area(m) × (d(m) − u), 0, 0, 0, 0⟩
11:   else {m is a non-leaf node}
12:     let m1, m2, m3, m4 are children of m
13:     for all u in F(m) do
14:       pick u1 ∈ F(m1), u2 ∈ F(m2), u3 ∈ F(m3),
15:       u4 ∈ F(m4) that minimize the quantity
16:
17:       x := Σ_{l=1}^{4} M^1[m_l][u_l] +
                          area(m) × ((Σ_{l=1}^{4} u_l) − u)
18:       where
          F(m) denotes the set [0..(d(m) − k)] ∪ {d(m)},
          and M^1[i][j] returns the first component of the
          tuple at M[i][j]
19:
20:       M[m][u] := ⟨x, u1, u2, u3, u4⟩
21: return M
```

Function $F(m)$ in line 13 limits the possibilities of the number of locations whose cloaking can be passed up by m. Notice that it rules out the values $d(m) - k + 1$ through $d(m) - 1$ since these imply cloaking less than k locations at m, which would immediately compromise k-anonymity. Quantity x is the minimum cost among all configurations C with k-summation for which $C(m) = u$. This is computed from the costs of the configurations at the 4 children, and the term $area(m) \times ((\sum_{l=1}^{4} u_l) - u)$, where $(\sum_{l=1}^{4} u_l) - u$ is the number of locations actually cloaked by m. Recall that the cost is found in the first component of the tuple stored in the matrix entry, whence the need for the projection operation M^1.

Notice how the algorithm mirrors the definition of the k-summation property (Definition 9) to ensure that only configurations satisfying k-summation are considered. By Lemma 3, these configurations represent only policy-aware sender k-anonymous policies. For instance, line 6 corresponds to case (i) in Definition 9, which prescribes that no locations are to be cloaked by m (all $d(m)$ locations occurring in its quadrant are passed up, $C(m) = d(m)$). Thus by Definition 8, the resulting cost is 0, which is what line 6 fills into the first component of $M[m][d(m)]$. Similarly, line 8 gives the cost corresponding to the case in the first disjunct of line (ii) of Definition 9; line 10 corresponds to the second disjunct. It's easy to see that:

Lemma 4. *Algorithm $Bulk_{dp}$ computes in each $M[m][u] = \langle x, u_1, u_2, u_3, u_4\rangle$ the minimum configuration cost x among all configurations C with k-summation where $C(m) = u$ and where $C(m_i) = u_i$, with $m_1, \ldots, m_4$ the children of m.*

By the above discussion, the information in M suffices to retrieve in PTIME a minimum-cost configuration.

Complexity analysis. The running time of Algorithm $Bulk_{dp}$ is dominated by steps 13-17 , which, for internal node m, ranges each of u, u_1, u_2, u_3, u_4 over at most $|D|$ values (since $F(n) \leq d(n) \leq |D|$ for every n), resulting in $O(|D|^5)$ iterations. Summing up over all nodes m of the quad-tree, we obtain the complexity of $Bulk_{dp}$ in $(O|T||D|^5)$. Lemma 4 and this complexity analysis directly imply Theorem 2. While polynomial, and thus a welcome surprise in contrast to Theorem 1, the degree 5 is impractically high given the large size of the location database, which is why we consider optimizations in Section V

Incremental Maintenance of M. Algorithm $Bulk_{dp}$ computes the optimal policy for a snapshot D of the location database starting from scratch (hence the name). As the users of the mobile network move around, the location database snapshot changes from D to D' at the next snapshot. Any optimal policy computed at snapshot D may not remain optimal for D', or may not provide policy-aware sender k-anonymity to users in D'. One can simply re-compute the optimal policy from scratch, calling algorithm $Bulk_{dp}$ on D' and T. Alternatively, if there are a large number of users in D but only few of them move between consecutive snapshots it makes sense to consider *incremental re-computation* of the optimal configuration matrix for D' starting from the optimal configuration matrix for D. This is easily accomplished by running the same bottom-up steps as algorithm $Bulk_{dp}$, with the added twist that the algorithm starts only from the quad tree leaves m whose quadrants now contain a changed number $d(m)$ of locations.

V. Optimizations

While the first-cut algorithm $Bulk_{dp}$ is polynomial, it is far from practical yet, since a degree of 5 is prohibitive given the typical sizes of location databases (the location database of a wireless service provider in the San Francisco Bay may contain about one million users). In this section, we describe a series of optimizations of the naive algorithm to achieve practical running time, while guaranteeing to preserving the optimal cost.

From Quad to Binary Trees. The algorithm described in [16] pioneers the idea of using quadrants of a quad-tree as cloaks. In a quad-tree, if cloaking a location to a node does not provide the desired k-anonymity, the next possible option is the parent node. Since the parent node is 4 times the size of a child node, the granularity of cost increase is large. The cloaking policy in Casper [23] reduces this granularity by considering *semi-quadrants* as cloaks (where a semi-quadrant is obtained by splitting a quadrant into two rectangles, either vertically or horizontally). The cloaks obtained in this approach are never larger than the cloaks obtained with the original quad-tree, and on average the cloak size is reduced.

While [23] uses this idea to improve utility, we additionally exploit it here to improve running time. We too allow cloaks to be chosen among both the original quadrants of quad tree T, and their semi-quadrants. To this end we define a modified tree whose nodes are of either shape. It is a *binary tree* B obtained from T as follows. For each node m in T, let its 4 children in T be $m_{NW}, m_{SW}, m_{SE}, m_{NE}$, where the subscript gives their location (Southeast, etc.) in m. We divide m into two vertical semi-quadrants (rectangular) s_W in the West and s_E in the East. In B, m becomes the parent of s_W and s_E, s_W the parent of m_{NW}, m_{SW}, and s_E the parent of m_{SE}, m_{NE}. Notice that each node in B has only 2 children, each non-leaf quadrant node is a parent of two semi-quadrants, and each non-leaf semi-quadrant is a parent of two quadrants. Casper chooses between vertical or horizontal sub-quadrants at run-time, while for simplicity we statically partition quadrants into vertical semi-quadrants only.

We adapt the $Bulk_{dp}$ algorithm to this binary tree. The only change required is in step 5 of the algorithm. When computing each entry M[m][u] of the optimum configuration matrix we have to iterate through the configurations of two children only (compared to four in $Bulk_{dp}$). This reduces the complexity of the loop to $O(|D|^2)$ from $O(|D|^4)$, and that of the algorithm to $O(|B||D|^3)$.

Note that the cost of the optimal binary-tree based policy for a given location database instance may be different from the optimal cost of the original quad-tree based policy. If the size of a leaf node is kept the same in the binary tree and quad tree then the binary tree will need to have twice the height of the quad-tree to cover the same region. If k is also kept the same, than the cost of an optimal binary tree based policy is not more than the cost of an optimal quad tree policy, since any policy-aware anonymous quad tree policy is also a policy-aware anonymous policy for the binary tree. The remaining optimizations in this paper focus on the binary tree.

From $O(|B||D|^3)$ to $O(|B|(kh)^3)$. For any node *m* of the binary tree, in the for loop of step 5, $Bulk_{dp}$ inspects $(d(m) - k + 1)$ sub-tree configurations (all possible configurations that satisfy k-summation) for the sub-tree rooted at *m*. We realize that some of these configurations need not be considered, as they are guaranteed to be sub-optimal.

In fact we claim the following lemma (proven in [12]):

Lemma 5. For a node m with height $h(m)$ (where the height of the root is 0), any configuration in which m passes up to its ancestors the cloaking responsibility for more than $(k+1)h(m)$ but less than $d(m)$ locations, is not optimal.

By Lemma 5, it suffices to compute $(k+1)h(m)$ configurations, by simply replacing function F in step 5 of algorithm $Bulk_{dp}$ with function $F'(m) = [0..((k+1)h(m))] \cup \{d(m)\}$.

With this insight, the number of columns required in the optimum configuration matrix M becomes at most kh, where h is the height of the tree. In step 5, for a non-leaf node m, the algorithm computes $O(kh)$ configurations and to compute each such configuration, the "pick" action iterates over $O(kh)$ configurations of m's two children. This leads to a new upper bound of the overall running time, $O(|B|(kh)^3)$. Note that if we fix a minimum area corresponding to the leaves of the tree, the height depends only on the area of the covered map, and, remarkably, we find a complexity upper bound that is independent of the

size of the location database! However, this upper bound is only of theoretical interest, since we actually implement yet another optimization: we do not eagerly materialize all nodes of the binary tree. Instead, we split a (semi-)quadrant only if it contains sufficient users to maintain anonymity. In our experiments, we observed that the number of materialized nodes $|B|$ *does* depend on the size of the location database.

From $O(|B|(kh)^3)$ to $O(|B|(kh)^2)$. Again we focus on the FOR loop in step 5 of the $Bulk_{dp}$ algorithm. We observe that, across iterations of the loop, the "pick" command will repeatedly inspect certain configurations of m's children. For example, if one iteration works on the M entry for (m, u), inspecting for instance (m_1, u_1) and (m_2, u_2) such that $u_1 + u_2 = u$ for some v, then the next iteration $(m, u + 1)$ will inspect the cases (m_1, u_1+1), (m_2, u_2) and (m_1, u_1), (m_2, u_2+1), among others. The idea is to reuse this computation across iterations. To this end, we stage the computation in 2 parts. In the first stage we iterate over the $O(kh)$ configurations of both children to compute a temporary matrix $temp$. An entry $temp[m][j]$ in this matrix stores the minimum cost c of having $j = l_1 + l_2$ un-anonymized locations in the two children m_1 and m_2 of m (with l_1 in m_1 and l_2 in m_2).

$$temp[m][j] := \min_{l_1+l_2=j} \{M[m_1][l_1] + M[m_2][l_2]\}.$$

There are $O(kh)$ entries in this matrix and the complexity of this stage is bounded by $O((kh)^2)$. In the second stage, we create $O(kh)$ configurations using the $O(kh)$ entries of temp.

$$x := \min_{j=i \text{ or } j\geq i+k} \{temp[m][j] + (j - i) \times area(m)\}.$$

Thus the running time for the second stage is also bounded by $O((kh)^2)$. Therefore the overall complexity of the modified step 5 is $O((kh)^2)$ and the overall complexity of the algorithm is $O(|B|(kh)^2)$.

Complexity Analysis in terms of $|D|$. Our complexity analysis so far was carried out for precision in terms of the size and height of the quad tree. While a gross upper bound for $|B|$ and h is $|D|$, leading to cubic running time in $|D|$, the real values of h and B depend on the skew of the locations in D. For instance, if the location distribution is uniform, it follows that $|B| \in O(\frac{|D|}{k})$ and $h \in O(\log(\frac{|D|}{k}))$, and the overall running time becomes $O(k|D|\log^2(\frac{|D|}{k}))$, i.e. linear for practical purposes in both k and $|D|$. It turns out that this analysis is highly robust to relaxing the assumption on uniformity. Our experiments in Section VI-A confirm the above formula even for realistic data whose distribution is quite skewed from uniform: 1.75 million locations reflecting the actual population density in the entire San Francisco Bay Area. The only examples we could create to force non-linear behavior are contrived.

Parallel Anonymization. We next explore a powerful technique for scaling the anonymization algorithm to cover large areas. The result is based on the key observation that the spatial nature of the problem features inherent parallelism that is easily exploited: just partition the region into sub-regions, putting each under the jurisdiction of an independent anonymization server. The servers run in parallel, each maintaining their own binary tree and location database, and seeing only requests issued in their jurisdiction. The policy in this distributed setting is a master policy which anonymizes a location l by referring to the policy constructed by the individual server under whose jurisdiction l falls.

One concern is that the obtained anonymization cost may no longer be optimal. To see why, consider cases when the best way to anonymize location l by server 1 is to issue a cloak that spans the jurisdiction of server 1 and its neighboring server 2. Since server 1 does not have access to the requests and location database in server 2, it will use a different, larger cloak, completely contained within its own jurisdiction. However these cases occur only on the border of jurisdictions, and the case when the spanning cloak is unavoidable requires very low population density at the borders. We therefore expect only a minimal divergence from the optimal cost. We verify this expectation experimentally in Section VI-D, showing that the system throughput can be effectively increased as more servers are added, while the cost remains within 1% of the optimum.

Assuming a fixed pool of servers, we would like to partition the map into jurisdiction so as to balance server load (the number of locations per server). We show that this is satisfactorily achieved even by an unsophisticated partitioning scheme. We adopt a greedy scheme which, given a location database and a binary tree B, picks jurisdictions from among the nodes of B. The greedy partitioning algorithm starts with the root as the only jurisdiction in the list L. At every step, the algorithm picks a node from the list, all of whose children have either 0 or at least k locations. If multiple such nodes exist, pick the one with the higher number of locations. The node is then replaced in L with its children. This repeats until the size of the list reaches the desired number of servers. Intuitively, we first greedily split the nodes with the highest number of locations, to balance the number of locations falling in each jurisdiction.

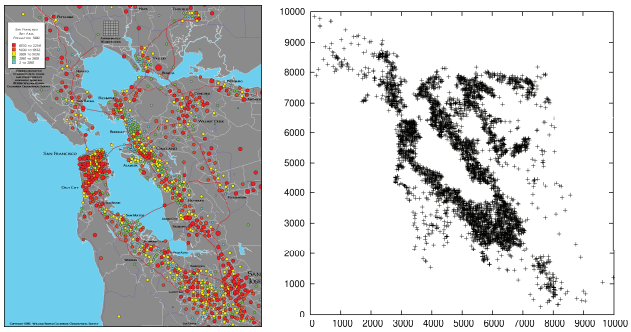
Our experiments reported in the next Section VI uncover the potential of parallel anonymization solutions. Note that in this work we do not advocate re-partitioning the map upon every location database snapshot, but instead picking a few representative snapshots and performing a static partition for each. The representatives pertain to various times of the day such as rush hours, night time, business hours, etc. In future work, we will study the systems issues related to the dynamic maintenance (and load re-balancing) of the server pool for highly dynamic fluctuations of the population density.

VI. Experiments

We next verify experimentally that our optimized algorithm scales well with the size of the location database, and that the stronger privacy guarantee comes at a reasonable cost increase.

Platform All our experiments were performed on an Intel Pentium4 3.20GHz machine running Linux with 2GB memory.

Location Data We set out to generate location data starting from a real-life map, using a real distribution of population density. Figure 2(a) illustrates the population density for the San Francisco Bay area in 1990, and is available from [7]. Unfortunately, the actual data values are not available, which is why we generated them as follows. We obtained a data set

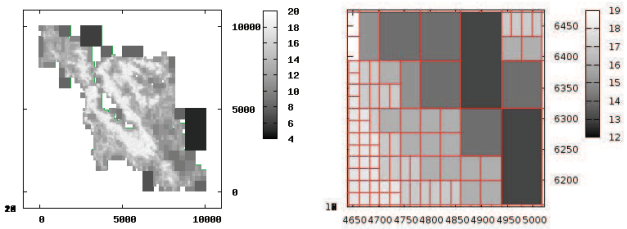


(a) Population Density (b) 5000 Street Intersections

Fig. 2. Street intersections and Population density

of street intersections in the same region (available at [8]). This dataset contains about 175k street intersection points. We conjectured that the population density is highly correlated with the intersection density. We validated this conjecture by plotting a random sample of 5000 points from this dataset (shown in Figure 2(b)), and observing that it is roughly similar to the actual population density graph of Figure 2(a).

We inserted 10 locations around each intersection using a Gaussian distribution with standard deviation of 500 meters. We obtained a *Master* dataset of 1.75 million locations. We believe this number to be realistic, since although the total population of the San Francisco Bay area is around 7M, it corresponds to the maximum market share at national level for any single national wireless provider (according to the statistics published in [2]). To scale the size of the location database for our experiments, we draw random samples of increasing sizes (100k, 200k etc.) from the Master data set.



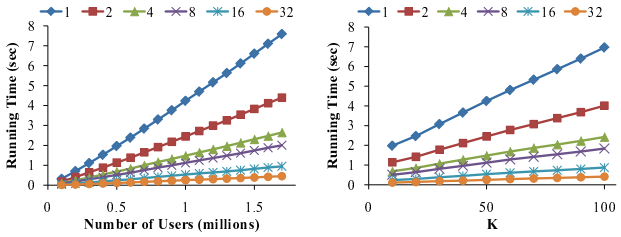
(a) Complete view (b) Zoomed-in view

Fig. 3. Tree structure built on 1M data

Warm-up experiment: shape of the quad tree. Recall from Section V that the binary tree is not computed eagerly; we split a (semi-)quadrant only if the resulting children contain sufficient users to maintain anonymity. Figure 3(a) illustrates the tree structure built on the 1M sample of the master file with $k = 50$. It plots the quadrants and semi-quadrants in a 2-dimensional plane. The height information is presented by the gray scale, so that nodes of greater height are brighter. It turns out that a binary tree of maximum height 20 suffices to cover 1M locations, with no leaves containing more than 50 locations. Even when growing $|D|$ to 1.75M locations, the height of the tree never reaches 25. As expected, the denser areas lead to greater height, showing that the algorithm exploits the larger density to materialize finer-grained (semi-)quadrants, which in turn lead to smaller cloaks and better utility. Figure 3(b) gives a zoomed-in view at a portion of the map, illustrating the variation of leaf (semi-)quadrant sizes as a function of population density.

A. Bulk Anonymization Time

In this experiment we evaluate the running time of our algorithm (the optimized version) varying the size of the location databases, the anonymization degree k, and the number of anonymization servers. Figure 4(a) shows, for fixed $k = 50$, the running time for computing an optimum configuration with increasing location database size, with one curve per number of servers. The horizontal axis shows the number of locations while the vertical axis shows the time in seconds. The running time is linear in the number of locations $|D|$ for up to 1.75M locations, as predicted by the complexity analysis at the end of Section IV. Notice that 16 servers can bulk anonymize 1.75M locations in less than 1s, and 32 servers in less than 0.5s. We note that this is extremely good scalability, especially since our experiment stress-tests the algorithm to sizes of the location database that far exceed the ones reported in prior work on policy-unaware sender anonymity: at most 300K in [17].



(a) Running time vs $|D|$ (b) Running time vs k

Fig. 4. Linear running time in $|D|$ and k

Next, we inspect how the running time scales with k, keeping the number of locations fixed at 1M. Figure 4(b) shows that the time increases quasi-linearly (really sub-linearly) with k, again as predicted by the analysis at the end of Section IV.

B. Cost Overhead of Stronger Privacy

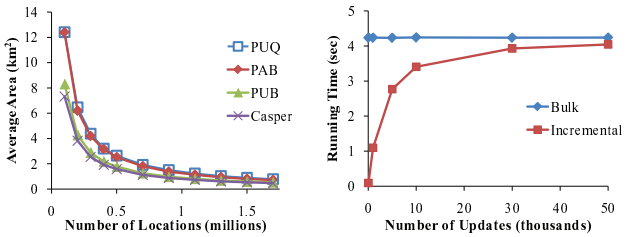
We expect that the stronger privacy guarantee will result in higher cost, by requiring larger cloaks in the anonymized requests. To evaluate the increase in cost from policy-unaware to policy-aware sender k-anonymity, we compared the *Cost* (in Definition IV) of the optimum policy-aware sender k-anonymous policy obtained using our algorithm, with that of

- Casper: since it is the state-of-the-art policy-unaware anonymizing system based on semi-quadrants [23], and our binary tree optimization was inspired by it.
- Optimum policy-unaware binary tree (PUB): since it uses the same type of cloak as our algorithm and a comparison would give a good measure of the penalty of stronger privacy.
- Optimum policy-unaware quad tree (PUQ): since this was the first system [16] that proposed to use quad-tree based cloak to provide (policy-unaware) sender k-anonymity.

We could not use the original implementation of Casper in our experiments since the interface allows no bulk anonymization: Casper reads the input one location tuple at a time and

for each location generates the cloak using only locations read up to that point. Instead of changing the original code, we decided to build a prototype of Casper based on the basic algorithm described in [23]. We did not implement the adaptive algorithm since it only affects the running time and not the size of the cloak. We also implemented the policy-unaware quad-tree policy described in [16] that finds the smallest quadrant that contains the requesting location and at least k-1 other location as the cloak. We implemented the same approach over a binary tree to obtain an optimum policy-unaware binary tree.

Figure 5(a) shows the comparison of average cloak areas obtained using the 4 algorithms described above. The horizontal axis represents the number of locations in the location database, and the vertical axis represents the average area (in square meters) of anonymized regions. k is fixed at 50. As expected, Casper has the minimum average cost among all the policies since it can select between horizontal or vertical semi-quadrants in contrast to fixed horizontal or vertical semi-quadrants selected by the policy-unaware binary tree. The cost of policy-aware sender k-anonymous policy is nearly identical to that of the policy-unaware quad-tree, and is at most 1.7 times that of Casper.



(a) Average cloak area for various policies

(b) Incremental Maintenance time for $|D| = 1M, k = 50$

Fig. 5. Parallel and Incremental Anonymization

C. Incremental Maintenance

We have also studied the performance of incrementally maintaining the optimum configuration matrix M from (the optimized version of) algorithm $Bulk_{dp}$. For the case of 1M locations and $k = 50$, we varied the number of locations that move from one snapshot of the location database to another. To this end, we randomly selected a set of distinct users updated their locations to a point at a randomly selected distance (bounded by 200 meters, that represents the maximum possible movement within 10 seconds) in a randomly selected direction. Figure 5(b) shows the comparison of performance of incremental maintenance with bulk re-computation. As expected, the time for incremental maintenance of M is always below that of the bulk re-computation as we increase the percent of moving users. However, we were surprised to notice that, once this percentage reaches 5%, the two times become virtually identical, and there is no gain in incremental maintenance. This is because most binary tree leaves require updating in that case, and incremental degenerates into bulk anonymization.

D. Utility Loss in Parallel Anonymization

Recall from Section V that one concern when splitting the map into jurisdictions is the sub-optimal utility due to cases when the best cloak to anonymize a location is missed because it spans jurisdictions. As discussed in Section V, we predict in general only a minimal divergence from the optimal cost. We verified this expectation experimentally by a stress test in which we increased the number of jurisdictions beyond reasonable limits: despite the above experiment showing that 16 suffice. Up to 2096 jurisdictions, the measured cost was identical to that of the optimal policy for the single-jurisdiction case, while the cost overhead for up to 4096 jurisdictions remained less than 1%.

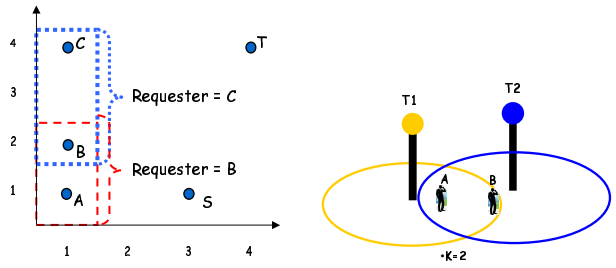
VII. Related Work

In the context of LBS, the two aspects of user privacy that have received the most attention are *location privacy* [5] and *sender anonymity*. The line of work on location privacy is complementary to this paper, as location privacy refers to hiding the precise location of the user (one is not required to hide the identity of the user) while sender anonymity refers to hiding the identity of the user (one is not required to hide the location, on the contrary, one assumes it falls in the attacker's hands). As described in the introduction, the extensions to *user-defined k* and *trajectory-aware attacker* are out of the scope of this paper and we leave them as future work.

Extensions of k-inside. Most of the proposals for sender anonymity are based on *k-anonymity* [26]. While a majority of these [16], [23], [27] are simply based on the *k-inside* policy (described earlier, and shown to not defend agaisnt policy-aware attacks), some use variations. In [14], the cloaking policy ensures that at least $k-1$ other users issue LBS requests from the cloaked region. It's been shown [16], [17], [11] that a k-inside policy fails to provide sender k-anonymity to "outlier" locations in some cases. To address this issue in k-inside policies, additional constraints of *k-reciprocity* [17] and *k-sharing* [11] have been proposed. *k-reciprocity* requires that among the $\geq k$ locations inside the cloak R of a location x, at least $k-1$ have x in *their* cloak, while *k-sharing* requires that at least $k-1$ of them have R as *their* cloak. We found that these additional constraints also fail to provide sender k-anonymity against a policy-aware attacker.

Consider the cloaking algorithm in [11] that takes into account the requesting locations to generate cloaking *groups* (set of locations that are cloaked to the same region). For locations in Figure 6(a), if the first request is made by C the algorithm groups C with B where as if the first request is made by B then it puts B and A in the same cloaking group to satisfy 2-sharing property. In the case when the initial request contains the cloak corresponding to $\{C, B\}$, a policy-aware attacker can infer that the sender is C!

Next, consider a cloaking algorithm that generates circular regions centered at the base station nearest to the cloaked location. As shown in Figure 6(b) user Alice is closest to station S_1, hence her cloak is centered at S_1. User Bob is closest to station S_2 so his cloak is centered at S_2. Since both users are inside the intersection of both circular cloaks, this cloaking satisfies 2-reciprocity. When a policy-aware attacker observes the cloaking region centered at S_1, he can infer that the only possible sender is Alice!



(a) with 2-sharing (b) with 2-reciprocity

Fig. 6. Privacy breach

Utilitity-maximizing cloaking. The problem of finding optimum k-anonymous cloaking that preserves privacy against a policy-unaware attacker has been considered (in [17]) to be NP-hard, by borrowing the results [22] from data k-anonymity. In [14] the authors study the problem of finding optimum cloaking using a minimum bounding box as the cloak and found it to be NP-hard and thus provide an approximate algorithm. Moreover it only uses the locations with pending requests for generating the cloak, as a result the cloak size can be quite large as not all users in a small region are expected to use LBS at nearly same time. The *FindMBC* algorithm in [27] computes the minimum bounding circular cloak that preserves privacy against a policy-unaware attacker. By Theorem 1, extending it to optimal policy-aware anonymization is likely hard.

Data k-anonymity.

One may argue that some of the algorithms developed for data k-anonymization can be applied to the location database, to reduce sender k-anonymization to the classical data k-anonymization problem. Data anonymization algorithms come to mind that are based on *generalization* [25], [18], [13]. They require as input a generalization hierarchy for the anonymized data. One could conceptually use quad-tree based regions to represent such a generalization hierarchy for the location data. However, the reduction from sender to data k-anonymization is inadvisable since the problem of finding the minimum-cost data anonymization is known to be NP-complete [22], leading to algorithms for optimal anonymization that run in time h^n, where h is the height of the generalization hierarchy (quad-tree for us) and n is the number of tuples (the size of the location database for us). As we show here, more headway can be made by exploiting the additional structure of the problem, namely that the data to be anonymized is location data.

Some of the recent proposals [20], [4], [9], [10] for k-anonymizing data are based on clustering techniques. Most of these clustering algorithms [20], [9], [10] use a static distance metric where the cost of including a point x in a cluster C is independent of the cost of including another point y in C. For location data, since the cost model uses the area of anonymizing regions, the point farthest from the center of the cluster determines (and thus affects) the cost for other points in the cluster. This precludes a reduction from sender anonymity to cluster-based data anonymity. The cost model in a recent proposal [4] takes care of this situation and thus its proposed clustering algorithm is a candidate to be investigated in future work for cloak generation in sender anonymity.

Private Information Retrieval [15] starts from the inital idea of having the requests mention no location information at all. The LBS sends all n points of interest *from the entire map* and the client filters them locally. An optimization consists in using cryptographic techniques that allows the sender to include its exact location in encrypted form, and the LBS to return $\sqrt{n}$ points of interest which include the ones closest to the sender. The result is directly obtained in encrpyted form and its clear form is hidden even from the LBS.

This solution addresses a different point in the space of possible trade-offs of privacy versus feasibility. It achieves maximal anonymity since all senders are cloaked by the entire map area. The price to pay includes costly adoption, low throughput, and limited billing model. Adoption is hindered by the need to change the operation of current LBS to include cryptographic query evaluation. Throughput is impacted because cryptographic query evaluation is expensive: [15] reports 20-45 seconds per query when the LBS maintains 65K points of interest . This time is lowered to 6-12 seconds per query when the computation is parallelized over 8 servers, depending on the length of the encryption key. Scaling to (tens of) thousands of requests per snapshot would therefore lead to either unacceptably slow response or prohibitively expensive massive parallelization. Secondly, the LBS's business model is limited because it does not know what query answers it returns. This precludes insertion of relevant ads, and rules out a Google-like model in which advertisers/service providers are charged by the volume of their ads/service postings reaching users. Since the LBS does not know what query answers it returns, this precludes insertion of relevant ads, and rules out a advertising-based business model in which advertisers/service providers are charged by the volume of their ads/service postings reaching users.

In contrast, our solution trades privacy (k is typically much lower than the number of all users) for feasibility: It requires virtually no change in existing LBS interfaces, whose input is essentially the one we describe above. It provides per snapshot a sub-second initialization time to bulk-anonymize over one million users, after which individual queries can be served in milliseconds. Indeed, serving a query requires looking up the location's cloak according to the computed policy, then evaluating a nearest-neighbor search for this cloak. Our experiments show that cloak lookup takes 0.3–0.5 ms. In [23], Casper reports 2ms per nearest-neighbor query when $k = 200$ and there are 10K points of interest, using GIS indexing techniques. Adopting Casper's GIS-based query evaluation results in a per-snapshot throughput increase of 3 orders of magnitude over the cryptographic query evaluation of [15]. Finally, flexible billing is facilitated since the LBS knows which advertisers/service providers to charge as it knows the query results.

Beyond k-anonymity: l-diversity and t-closeness

Taking a page from recent developments that improve on data k-anonymity, it is natural to ask if there are corresponding extensions of the notion of sender k-anonymity. The ansewer is positive, in the following sense. In data k-anonymity, there is a class of attacks based on counting the frequency of sensitive at-

tribute values in the anonymized table. L-diversity [21] defends against the situation when all tuples in an anonymized group share the same sensitive attribute value, in which case they are all compromised. T-closeness [19] goes beyond, defending even against attacks that compare the frequency of sensitive attribute values in the whole table against the frequency of sensitive attributes in individual anonymized groups. Whenever the two frequencies differ, the attacker learns something about the secret, and this fact is considered a privacy leak. The analogous attacks in our setting would consist of counting in each snapshot the number of duplicate requests, grouping by cloaking area and request values. For instance, the (unlikely) event of observing in a snapshot as many identical requests from the same cloak as the number of locations residing in it at that time exposes all senders. This assumes that a sender can issue a single request per snapshot, which is reasonable given the short snapshot duration.

The following simple modification of our approach to sender k-anonymity precludes the same class of frequency-based attacks as l-diversity and t-closeness: the anonymization server caches the query results returned by the LBS, indexed by the anonymized request. This means that the LBS does not even see duplicate anonymized requests during the same snapshot, and therefore cannot count their frequency (nor can it log it and thus make the count available to hacking or subpoena). For queries about stationary points of interest (most businesses and tourist attractions), the anonymizer can use the cache for a long time, thus precluding counting even across multiple snapshots. To adjust for the appearance and disappearance of points of interest, it suffices to flush the cache at infrequent intervals (for instance once a day). To help the LBS with billing, the anonymizer can keep a total count and submit it to the LBS at cache flushing time.

VIII. Conclusion

We introduce the notion of sender k-anonymity against policy-aware attackers. This privacy guarantee is stronger than the sender k-anonymity in prior work, which defends against policy-unaware attackers only. Our results show that the novel guarantee strikes a pragmatic balance in the trade-off between strength of the privacy guarantee, utility, and running time for enforcement.

We also show the considerable amenability of the problem to parallelization, which reduces the anonymization time while preserving the optimal utility in virtually all cases. Indeed, dividing the San Francisco Bay area among 4K servers –far more than needed since 16 suffice– leads to only 1% divergence of the cost from the optimum. 16 servers already provide anonymization time of about half a second for 1 million users.

References

[1] Au's GPS cell phone shows how to get to McDonald's. http://www.mobilemediajapan.com/headline2.asp?page=AU/KDDI.

[2] Verizon News Center: 4Q and Full-Year 2008. http://news.vzw.com/news/2009/01/pr2009-01-27.html.

[3] Wireless 911 Services. http://www.fcc.gov/cgb/consumerfacts/wireless911srvc.html.

[4] G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy, D. Thomas, and A. Zhu. Achieving anonymity via clustering. In *PODS*, 2006.

[5] A. R. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2(1):46–55, 2003.

[6] C. Bettini, X. S. Wang, and S. Jajodia. Protecting privacy against location-based personal identification. In *SDM*, 2005.

[7] P. W. Bowen. DIGITAL ATLAS OF CALIFORNIA. http://130.166.124.2/CApage1.html.

[8] T. Brinkhoff. A framework for generating network-based moving objects. http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator.

[9] J.-W. Byun, A. Kamra, E. Bertino, and N. Li. Efficient *k*-anonymization using clustering techniques. In *DASFAA*, 2007.

[10] C.-C. Chiu and C.-Y. Tsai. A k-anonymity clustering method for effective data privacy preservation. In *ADMA*, pages 89–99, 2007.

[11] C.-Y. Chow and M. F. Mokbel. Enabling private continuous queries for revealed user locations. In *SSTD*, volume 4605 of *LNCS*. Springer, 2007.

[12] A. Deutsch, R. Hull, A. Vyas, and K. Zhao. Policy-aware sender anonymity in location based services. TR CS2009-0939, UCSD, 2009. http://www.cse.ucsd.edu/users/avyas/Tech-Report/full-version.ps

[13] B. C. M. Fung, K. Wang, and P. S. Yu. Top-down specialization for information and privacy preservation. In *ICDE*, 2005.

[14] B. Gedik and L. Liu. Location privacy in mobile systems: A personalized anonymization model. In *ICDCS*, pages 620–629, 2005.

[15] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private queries in location based services: anonymizers are not necessary. In *SIGMOD*, 2008.

[16] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *MobiSys*, 2003.

[17] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preventing location-based identity inference in anonymous spatial queries. *IEEE Trans. on Knowl. and Data Eng.*, 19(12):1719–1733, 2007.

[18] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: efficient full-domain k-anonymity. In *SIGMOD*, 2005.

[19] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, pages 106–115, 2007.

[20] J.-L. Lin and M.-C. Wei. An efficient clustering method for k-anonymization. In *PAIS*, pages 46–50. ACM, 2008.

[21] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. ℓ -diversity: Privacy beyond κ -anonymity. In *ICDE*, 2006.

[22] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *PODS*, 2004.

[23] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The new casper: query processing for location services without compromising privacy. In *VLDB*, 2006.

[24] J. H. Saltzer. Protection and the control of information sharing in multics. *Commun. ACM*, 17(7):388–402, 1974.

[25] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):571–588, 2002.

[26] L. Sweeney. k-anonymity: a model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[27] T. Xu and Y. Cai. Location anonymity in continuous location-based services. In *GIS*, 2007.

Appendix