Preserving Location Privacy in Geo-Social Applications

Krishna P. N. Puttaswamy, Shiyuan Wang, Troy Steinbauer,*
Divyakant Agrawal, Amr El Abbadi, Christopher Kruegel and Ben Y. Zhao
Department of Computer Science, UC Santa Barbara
{krishnap, sywang, troysteinbauer, agrawal, amr, chris, ravenben}@cs.ucsb.edu

Abstract—Using geo-social applications, such as FourSquare, millions of people interact with their surroundings through their friends and their recommendations. Without adequate privacy protection, however, these systems can be easily misused, *e.g.,* to track users or target them for home invasion. In this paper, we introduce *LocX*, a novel alternative that provides significantly-improved location privacy without adding uncertainty into query results or relying on strong assumptions about server security. Our key insight is to apply secure user-specific, distance-preserving *coordinate transformations* to all location data shared with the server. The friends of a user share this user's secrets so they can apply the same transformation. This allows all location queries to be evaluated correctly by the server, but our privacy mechanisms guarantee that servers are unable to see or infer the actual location data from the transformed data or from the data access. We show that LocX provides privacy even against a powerful adversary model, and we use prototype measurements to show that it provides privacy with very little performance overhead, making it suitable for today's mobile devices.

Index Terms—Location privacy, security, location-based social applications, location transformation, efficiency.

1 Introduction

With billions in downloads and annual revenue, smartphone applications offered by Apple iTunes and Android are quickly becoming the dominant computing platform for today's user applications. Within these markets, a new wave of *geo-social* applications are fully exploiting GPS location services to provide a "social" interface to the physical world. Examples of popular social applications include social rendezvous [1], local friend recommendations for dining and shopping [2], [3], as well as collaborative network services and games [4], [5]. The explosive popularity of mobile social networks such as SCVNGR [6] and FourSquare (3 million new users in 1 year) likely indicate that in the future, social recommendations will be our primary source of information about our surroundings.

Unfortunately, this new functionality comes with significantly increased risks to personal privacy. Geo-social applications operate on fine-grain, time-stamped location information. For current services with minimal privacy mechanisms, this data can be used to infer a user's detailed activities, or to track and predict the user's daily movements. In fact, there are numerous real world examples where the unauthorized use of location information has been misused for economic gain [7], physical stalking [8], and to gather legal evidence [9]. Even more disturbing, it seems that less than a week after Facebook turned on their popular "Places" feature for tracking users' locations, such location data was already used by thieves to plan home invasions [10]. Clearly, mobile social networks of tomorrow require stronger privacy properties than the open-to-all policies available today.

**Work done while the first author was at UCSB. He is currently affilicated with Bell Labs, Alcatel-Lucent. He can be reached at* krishna.puttaswamy_naga@alcatel-lucent.com.

Existing systems have mainly taken three approaches to improving user privacy in geo-social systems: (a) introducing uncertainty or error into location data [11], [12], [13], (b) relying on trusted servers or intermediaries to apply anonymization to user identities and private data [14], [12], [15], and (c) relying on heavy-weight cryptographic or private information retrieval (PIR) techniques [16], [17], [18], [19]. None of them, however, have proven successful on current application platforms. Techniques using the first approach fall short because they require both users and application providers to introduce uncertainty into their data, which degrades the quality of application results returned to the user. In this approach, there is a fundamental tradeoff between the amount of error introduced into the time or location domain, and the amount of privacy granted to the user. Users dislike the loss of accuracy in results, and application providers have a natural disincentive to hide user data from themselves, which reduces their ability to monetize the data. The second approach relies on the trusted proxies or servers in the system to protect user privacy. This is a risky assumption, since private data can be exposed by either software bugs and configuration errors at the trusted servers or by malicious administrators. Finally, relying on heavy-weight cryptographic mechanisms to obtain provable privacy guarantees are too expensive to deploy on mobile devices [20], [21], and even on the servers in answering queries such as nearest-neighbor and range queries.

The challenge, then, is to design mechanisms that efficiently protect user privacy without sacrificing the accuracy of the system, or making strong assumptions about the security or trustworthiness of the application servers. More specifically, we target geo-social applications, and assume that servers (and any intermediaries) can be compromised and, therefore, are untrusted. To limit misuse, our goal is to limit accessibility of location information from global visibility to a user's social circle. We identify two main types of queries necessary to support the functionality of these geo-social applications: point queries and nearest-neighbor (kNN) queries. Point queries query for location data *at* a particular point, whereas kNN queries query for k nearest data *around* a given location coordinate (or up to a certain radius). Our goal is to support both query types in an efficient fashion, suitable for today's mobile devices.

To address this challenge, in this paper, we propose *LocX* (short for location to index mapping), a novel approach to achieving user privacy while maintaining full accuracy in location-based social applications (LBSAs from here onwards). Our insight is that many services do not need to resolve distance-based queries between arbitrary pairs of users, but only between friends interested in each other's locations and data. Thus, we can partition location data based on users' social groups, and then perform *transformations* on the location coordinates before storing them on untrusted servers. A user knows the transformation keys of all her friends, allowing her to transform her query into the virtual coordinate system that her friends use. Our coordinate transformations preserve distance metrics, allowing an application server to perform both point and nearest-neighbor queries correctly on transformed data. However, the transformation is *secure*, in that transformed values cannot be easily associated with real world locations without a *secret*, which is only available to the members of the social group. Finally, transformations are efficient, in that they incur minimal overhead on the LBSAs. This makes the applications built on LocX lightweight and suitable for running on today's mobile devices.

2 Scenarios and Requirements

Here we describe several scenarios we target in the context of emerging geo-social applications that involve heavy interaction of users with their friends. We use these scenarios to identify the key requirements of a geo-social location privacy preserving system.

2.1 Geo-social Application Scenarios

Scenario 1. Alice and her friends are excited about exploring new activities in their city and leveraging the "friend referral" programs offered by many local businesses to obtain discounts. Alice is currently in downtown and is looking to try a new activity in her vicinity. But she also wants to try an activity that gives her the most discount. The discounts are higher for a user that refers more friends or gets referred by a friend with high referral count. As a result Alice is interested in finding out the businesses recommended by her friends and the discounts obtained through them, within her vicinity. In addition, she is also interested in checking if there are discounts available for her favorite restaurant at a given location.

Scenario 2. Alice and her friends are also interested in playing location-based games and having fun by exploring the city further. So they setup various tasks for friends to perform, such as running a few miles at the Gym, swimming certain laps, taking pictures at a place, or dining at a restaurant. They setup various points for each task, and give away prizes for the friends with most points. In order for Alice to learn about the tasks available near her, she needs to query an application to find out all tasks from friends near her and the points associated with them.

The scenarios above, while fictitious, are not far from reality. Groupon and LivingSocial are some example companies that are leading the thriving business of local activities. SCVNGR [6] offers similar services as location-based games. But none of these services provide any location privacy to users: all the locations visited by the users are known to these services and to its administrators.

Our goal is to build a system that caters to these scenarios and enables users to query for friends' data based on locations, while preserving their location privacy. We want to support: a) *point query* to query for data associated with a particular location, b) *circular range query* to query for data associated with all locations in a certain range (around the user), and c) *nearest-neighbor query* to query for data associated with locations nearest to a given location. Finally, while it is also useful to query for data that belongs to non-friends in certain scenarios, we leave such extensions for future.

2.2 System Requirements

The target scenarios above bring out the following key requirements from an ideal location-privacy service.

- Strong location privacy: The servers processing the data (and the administrators of these servers) should not be able to learn the history of locations that a user has visited.
- Location and user unlinkability: The servers hosting the services should not be able to *link* if two records belong to the same user, or if a given record belongs to a given user, or if a given record corresponds to a certain real-world location.
- Location data privacy: The servers should not be able to view the content of data stored at a location.
- Flexibility to support point, circular range, and nearest-neighbor queries on location data.
- Efficiency in terms of computation, bandwidth, and latency, to operate on mobile devices.

The need for each of these requirements becomes more clear when we describe the related work and their limitations in more detail in the next section. In our proposed system, LocX, we aim to achieve all these requirements.

3 Related Work

Prior work on privacy in general location-based services (LBS). There are mainly three categories of proposals on providing location privacy in general LBSs that do not specifically target social applications. First is spatial and temporal cloaking [11], [12], [13], [22], [15], wherein approximate location and time is sent to the server instead of the exact values. The intuition here is that this prevents accurate identification of the locations of the users, or hides the user among k other users (called k-anonymity [12], [13], [22]), and thus improves privacy. This approach, however, hurts the accuracy and timeliness of the responses from the server, and most importantly, there are several simple attacks on these mechanisms [23], [24], [25], [26] that can still break user privacy. Pseudonyms and silent times [27], [14] are other mechanisms to achieve cloaking, where in device identifiers are changed frequently, and data is not transmitted for long periods at regular intervals. This, however, severely hurts functionality and disconnects users. The key difference between these approaches and our work is that they rely on trusted intermediaries, or trusted servers, and reveal approximate real-world location to the servers in plain-text. In LocX, we do not trust any intermediaries or servers. On the positive side, these approaches are more general and, hence, can apply to many location-based services, while LocX focuses mainly on the emerging geo-social applications.

The second category is location transformation, which uses transformed location coordinates to preserve user location privacy. One subtle issue in processing nearest-neighbor queries with this approach is to accurately find all the real neighbors. Blind evaluation using Hilbert Curves [21], unfortunately, can only find approximate neighbors. In order to find real neighbors, previous work either keeps the proximity of transformed locations to actual locations and incrementally processes nearest-neighbor queries [28], or requires trusted third parties to perform location transformation between clients and LBSA servers [29]. In contrast, LocX does not trust any third party and the transformed locations are not related to actual locations. However, our system is still able to determine the actual neighbors, and is resistant against attacks based on monitoring continuous queries [30], [31].

The third category of work relies on Private Information Retrieval (PIR) [16] to provide strong location privacy. Its performance, although improved by using special hardwares [17], is still much worse than all the other approaches, thus it is unclear at present if this approach can be applied in real LBSs.

Prior work on privacy in geo-social services For certain types of geo-social services, such as buddy tracking services to test if a friend is nearby, some recent proposals achieve provable location privacy [18], [19] using expensive cryptographic techniques such as secure two party computation. In contrast, LocX only uses inexpensive symmetric encryption and pseudorandom number generators. The closest work to LocX is Longitude [32], [33], which also transforms locations coordinates to prevent disclosure to the servers. However, in Longitude, the secrets for transformation are maintained between every pair of friends in order to allow users to selectively disclose locations to friends. As in, Longitude can let a user reveal her location to only a subset of her friends. In contrast, LocX has a simpler threat model where all friends can access a user's information and hence the number of secrets that users have to maintain is only one per user. LocX can still achieve location and user unlinkability. In addition, LocX can provide more versatile geo-social services, such as location based social recommendations, reminders, and others, than just buddy tracking as in the above prior work.

Anonymous communication systems. These systems, including Tor [34], provide anonymity to users during network activity. One might ask, then, *why using Tor to anonymously route data to LBSA servers is not sufficient?* This approach seems to provide privacy as the server only sees location data but not the identity of the user behind that data. However, recent research has revealed that hiding the identity of the users alone is not sufficient to protect location privacy. Even if Tor is used, it is possible for an attacker with access to the location data to violate our *privacy* and *unlinkability* requirements. For example, using anonymized GPS traces collected by the servers, it has been shown that users' home and office locations, and even user identity can be derived [23], [24], [25], [26]. LocX defends against such attacks and meets all our requirements.

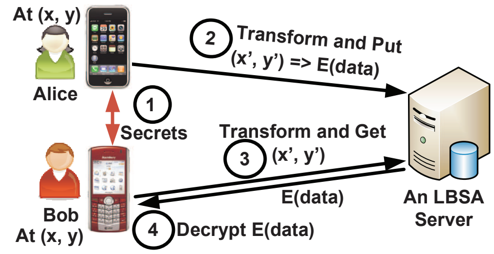


Fig. 1. A basic design. In this design, 1) Alice and Bob exchange their secrets, 2) Alice stores her review of the restaurant (at (x, y)) on the server under transformed coordinates, 3) Bob later visits the restaurant and queries for the reviews on transformed coordinates, and 4) decrypts the reviews obtained.

Systems on untrusted servers. In the context of databases, recent systems proposed running database queries on encrypted data (stored on untrusted servers), using heavy-weight homomorphic [35] or asymmetric encryption [36] schemes. These approaches are suitable for spatial data outsourcing or data mining scenarios where the data is static and is owned by limited number of users. But they are less suitable for LBSAs, where the data is dynamic and personal, and thus cannot be encrypted under a single secret key.

In the context of location and social applications, Persona [37] and Adeona [38] also relied on encrypting all data stored on untrusted servers to protect user privacy. Persona focused on privacy in online social networks, and Adeona focused on privacy in device tracking systems where there is no data sharing among users. Applying Persona's mechanisms to LBSAs directly would encrypt all location coordinates, making LBSAs unable to process nearest-neighbor queries. But if location is not encrypted, attacks using anonymized GPS traces, mentioned above, can succeed, making Persona insufficient to protect location privacy. Similarly, Adeona is useful for a user to retrieve her own data, but not the data from her friends. Our contributions complement these systems. Some techniques in these papers can help LocX as well, e.g. Persona's approach to partition data shared with friends into fine-grained groups, and Adeona's hardware-assisted approaches to speed up crypto processing.

4 System Design

In this section, we describe the design of LocX in detail.

4.1 Terminology and Attacker Model

Terminology. *Location coordinates* refer to the longitude, latitude pairs associated with real-world locations. A pair of coordinates is returned from a GPS, and is used to associate data with a location. *Location data or location information* refers to such data associated with a location. For example, when reviews (and referral point details) are written for a given restaurant, the reviews are the location data associated with the restaurant's location coordinates.

System and Attacker Model. In this paper, we assume that the companies that provide LBSA services manage the servers. Users store their data on the servers to obtain the service. The companies are responsible for reliably storing this data, and providing access to all the data a user should have access to. The companies can get incentives via displaying ads, or charging users some usage fees. In our attacker model, we assume that the attacker has access to the LBSA servers. This attacker could be an employee of the company running the service or an outsider that compromises the servers. The attacker might even be an oppressive regime or a government that obtains data from the providers via subpoenas. As a result, in our model, the attacker can access all the data stored on the servers, and can also monitor which user device is accessing which pieces of information on the servers. Our goal is to design a system that preserves the location privacy of users in this setting. We assume that the attacker does not perform any attacks on the consistency or integrity of data on the servers, but aims only to learn users' location information. Finally, like all prior social systems [39], [40], [41], [37], we assume that the friends of a user are trusted and *do not collude* with the servers in breaking the user's privacy.

4.2 A Basic Design

To clarify the need for each component in LocX, we start the design description with a basic, simple design.

As listed in our requirements, the server should support different types of queries (point, circular range and nearest-neighbor queries) on location data. For the server to be able to do this, we need to reveal the location coordinates in plain text. But doing so would allow the malicious server to break a user's location privacy.

To resolve this problem, we propose the idea of *coordinate transformation*. Each user u in the system chooses a set of secrets that they reveal only to their friends. These secrets include a rotation angle θ_u, a shift b_u, and a symmetric key $symm_u$. The users exchange their secrets via interactions when friends meet in person, or via a separate trusted channel, such as email, phone etc. The secret angle and shift are used by the users to transform all the location coordinates they share with the servers. Similarly, the secret symmetric key is used to encrypt all the location data they store on the servers. These secrets are known only to the friends, and hence only the friends can retrieve and decrypt the data.

For example, when a user u wants to store a review r for a restaurant at (x, y), she would use her secrets to transform (x, y) to (x', y') and store encrypted review $E(r)$ on the server. When a friend v wants to retrieve u's review for the restaurant at (x, y), she would again transform (x, y) using u's secret (previously shared with v), retrieve $E(r)$, and then decrypt it using u's symmetric key to obtain r. Similarly, v would transform (x, y) according to each of her friends' secrets, obtain their reviews, and read them. We only focus on point queries for now. Figure 1 depicts this basic design.

A limitation. This basic design has one important limitation: the server can uniquely identify the client devices (for *e.g.*, using the IP address). Using this, the server can associate different transformed coordinates to the same user (using the IP). Sufficient number of such associations can break the transformations (as we show in Section 5). So maintaining unlinkability between different queries is critical.

One approach to resolve this limitation is to route all queries through an anonymous routing system like Tor [34]. But simply routing the data through Tor all the time will be inefficient. Especially in the context of recent LBSAs, that adds larger multimedia files (pictures and videos) at each location. So we need to improve this basic design to be both secure and efficient.

4.3 Overview of LocX

LocX builds on top of the basic design, and introduces two new mechanisms to overcome its limitations. First, in LocX, we split the mapping between the location and its data into two pairs: a mapping from the transformed *location to an encrypted index* (called **L2I**), and a mapping from the *index to the encrypted location data* (called **I2D**). This splitting helps in making our system efficient. Second, users store and retrieve the L2Is via *untrusted proxies*. This redirection of data via proxies, together with splitting, significantly improves privacy in LocX. For efficiency, I2Ds are not proxied, yet privacy is preserved (as explained later).

Decoupling a location from its data. In today's systems, location data $data_{(x,y)}$ corresponding to the real-world location (x, y) is stored under (x, y) on the server. But in LocX, the location (x, y) is first transformed to (x', y'), and the location data is encrypted into $E(data_{(x,y)})$. Then the transformed location is decoupled from the encrypted data using a random index i via two servers as follows: *1)* an L2I = $[(x', y'), E(i)]$, which stores $E(i)$ under the location coordinate (x', y'), and *2)* an I2D = $[i, E(data_{(x,y)})]$, which stores the encrypted location data $E(data_{(x,y)})$ under the random index i. The index is generated using the user's secret random number generator. We refer to the server storing L2Is as the *index server* and the server storing I2D as the *data server*. We describe these two as separate servers for simplicity, but in reality they can be on the same server, and our privacy properties still hold. This separation of location information into two components (L2I and I2D) helps us continue to efficiently run different types of location queries on L2Is and retrieve only relevant I2Ds.

The key interfaces used by the applications to store and retrieve data on the LocX servers are listed in Table 1. Figure 2 depicts the design of LocX.

Proxying L2Is for location privacy. Users store their L2Is on the index server via *untrusted proxies*. These proxies can be any of the following: PlanetLab nodes, corporate NATs

API Call	Purpose of the Call
putL2I ((x', y'), E(i))	Put L2I of (x, y) on the IS.
getL2I ((x', y'))	Get the L2I of (x, y) from the IS.
putD2I (i, E(data))	Put I2D of (x, y) on the DS.
getD2I (i)	Get I2D of (x, y) from the DS.

TABLE 1
The index server (IS) and data server (DS) APIs and their functions in LocX.

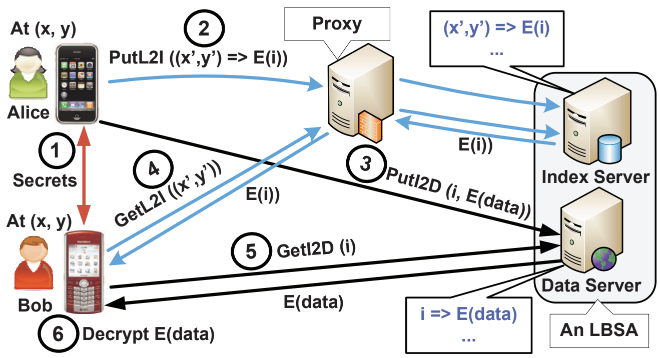


Fig. 2. Design of LocX. 1) Alice and Bob exchange their secrets, 2) Alice generates and L2I and I2D from her review of the restaurant (at (x, y)), and stores the L2I on the index server via a proxy. 3) She then stores the I2D on the data server directly, 4) Bob later visits the restaurant and fetches for L2Is from his friends by sending the transformed coordinates via a proxy, 5) he decrypts the L2I obtained and then queries for the corresponding I2D, 6) finally Bob decrypts Alice's review.

and email servers in a user's work places, a user's home and office desktops or laptops, or Tor [34] nodes. We only need a one-hop indirection between the user and the index server. These diverse types of proxies provide tremendous flexibility in proxying L2Is, thus a user can store her L2Is via different proxies without restricting herself to a single proxy. Furthermore, compromising these proxies by an attacker does not break users' location privacy, as (a) the proxies also only see transformed location coordinates and hence do not learn the users' real locations, and (b) due to the noise added to L2Is (described later). To simplify the description, for now, we assume that the proxies are non-malicious and do not collude with the index server. But we will later describe our solution in detail to even defend against colluding, malicious proxies.

With this high-level overview, we now describe our solution to store and query data on the servers in detail. We also explain the challenges we faced, and the tradeoffs we made in making our solution secure and efficient.

4.4 Privacy Preserving Data Storage

When a user generates the location data corresponding to a location (x, y), she uses her secrets to decouple it into a L2I and an I2D. Now we describe how they are stored on the index and the data servers respectively.

Storing L2I on the index server. First consider storing L2I on the index server. To perform this, the user transforms her real-world coordinate (x, y) to a virtual coordinate (x', y') using her secret rotation angle θ_u and secret shift b_u: $(x', y') \leftarrow (cos\theta_u x - sin\theta_u y + b_u, sin\theta_u x + cos\theta_u y + b_u)$. This transformation preserves the distances between points[1], so circular range and nearest neighbor queries for a friend's location data can be processed in the same way on transformed coordinates as on real-world coordinates. Then the user generates a random index (i) using her random number generator and encrypts it with her symmetric key to obtain $(E_{symm_u}(i))$. The user then stores this L2I, $[(x', y'), E_{symm_u}(i)]$, at the transformed coordinate on the index server via a proxy. The L2I is small in size and is application independent, as it always contains the coordinates and an encrypted random index. Thus the overhead due to proxying is very small (quantified in Section 6).

Storing I2Ds on the data server. The user can directly store I2Ds (location data) on the data server. This is both secure and efficient. *1)* This is secure because the data server only sees the index stored by the user and the corresponding encrypted blob of data. In the worst case, the data server can link all the different indices to the same user device, and then link these indices to the retrieving user's device. But this only reveals that one user is interested in another user's data, but not any information about the location of the users, or the content of the I2Ds, or the real-world sites to which the data in the encrypted blob corresponds to. *2)* The content of I2D is application dependent. For example, a location-based video or photo sharing service might share multiple MBs of data at each location. Since this data is not proxied, LocX still maintains the efficiency of today's systems.

Intuition behind privacy. Due to the coordinate transformation, the index server does not see the real-world coordinate of the user. Because of the proxy, the index server cannot link the different L2Is stored on the index server to the same user. The index server has a single coordinate space in which it stores all the data from all the users. These are the reasons behind the privacy in LocX. To break a user's privacy, a malicious index server will have to *break two steps:* a) learn the transformation secrets of the user, and b) link a request to the corresponding user (otherwise, the attacker does not know which transformation secret to apply to a request). These two steps significantly raise the bar for attacks.

4.5 Privacy Preserving Data Retrieval

Retrieving location data from the server in LocX is a more challenging problem. In particular, we need to *a)* maintain location privacy, and *b)* ensure that the retrieval is efficient.

Consider the following simple design for data retrieval. A user takes the location coordinate she is interested in, transforms it according to all her friends' secrets, and sends a query to the server containing all the transformed locations via a proxy. If a user has f friends, and is at a location (x, y), she sends a query with points $((x'_1, y'_1), (x'_2, y'_2), ..., (x'_f, y'_f))$ to the server, where (x'_i, y'_i) is the transformation of (x, y) with friend i's secret. The index server then fetches all the L2Is at the locations in the query and returns them. The user then

1. Given any two real-world points (x_1, y_1), (x_2, y_2), it is easy to see that the distance between their corresponding virtual coordinates $\sqrt{(x'_2 - x'_1)^2 + (y'_2 - y'_1)^2} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

decrypts all the returned L2Is, and queries the data server for the I2Ds she cares about. There might be collisions on the indices generated by different users. However, as the data in I2D is encrypted using shared symmetric keys, collisions do not lead to unauthorized data access.

This design has two major problems. First, this approach to query the server easily breaks a user's privacy. Just by knowing that all the transformed points sent by a user correspond to the same real-world coordinate, the server can construct and solve a set of equations to derive the real-world location of the user (proven in Section 5). To prevent this derivation, if the user were to query for each friend's transformed coordinate separately, then it would increase the total time (and the # of RPCs) to retrieve the results, hurting the performance. Thus we need a secure and efficient approach *to retrieve L2Is* from the index server. Second, since the server sends all the points stored at a transformed coordinate (x_i', y_i') in the query (irrespective of who stored data there), the user may get many L2Is from non-friends who happen to store data at location (x_i', y_i'). Since the user does not know the source, she will have to attempt to decrypt *all L2Is returned* in response to location (x_i', y_i') with friend i's symmetric key. This wastes significant amount of computation cycles on the user's device. Thus, we need an efficient and secure mechanism *to identify the L2Is* that are from friends, and to quickly reject L2Is from non-friends. We next describe our solutions to these two problems.

Privacy while querying the index server. In order to prevent attacks while querying the index server, we propose that users add noise to the query. Noise in our solution is a few (N) additional, randomly selected points, $((x1_1', y1_1'), (x1_2', y1_2'), ..., (x1_N', y1_N'))$, added to each query sent to the index server. Of course, the noise added has to be minimal for efficiency. We show through analysis (in § 5) that adding only a few additional random points prevents privacy attacks, and the server will not be able to derive the real location of the user. In addition, the user can easily filter out the L2Is corresponding to the noise. Note that the noise in LocX does not affect the accuracy of the locations unlike prior systems [11].

Adding noise, coupled with routing the index server queries via proxies (just like the way they were stored), provides strong location privacy during querying. The queries only contain a list of points in the transformed coordinate space, without any user identifier or actual location information. Due to proxying, the server cannot identify the client. And finally the noise prevents derivation of user's location based on transformed coordinate. Putting noise and proxying schemes together, the server cannot link multiple different queries to the same user. We will later prove that this unlinkability preserves the users' secrets, and also show that this approach is resilient against collusion between the proxies and the index server.

Securely and efficiently identifying L2Is of friends. In the simple design for data retrieval described above, we query for a set of points in the transformed coordinates and decrypt *all* the returned results. This provides strong privacy as the server does not learn which of the returned L2Is are relevant to the user, but decrypting all the results increases the overhead on the client's device.

If, on the other hand, we provide some information to the server to filter out the L2Is that are irrelevant for a user before sending them, it provides efficiency, but breaks privacy. For example, suppose each user attaches an anonymized ID to each L2I. Then, a user can submit a list of IDs she cares about and some additional IDs for noise. This allows the server to send only the L2Is at a point that fall into the set of IDs specified by the user. Even decryption would be efficient, as the user would know the right key to use for each L2I. Unfortunately, these IDs would enable the server to link different L2Is, and this can lead to privacy leaks. For instance, the index server could perform "fingerprinting attacks," by leveraging the distance preserving property of our transformations. In these attacks, the server takes "fingerprints" of popular destinations (e.g. airports in major cities), and uses the distance between these destinations as fingerprints. It then matches these fingerprints with the locations corresponding to a particular user identified by the ID, and then derives the transformation secret of the user. This would then reveal all the real-world locations of that user, which could help identify the user behind the ID.

Fundamentally, there is a tradeoff between efficiency and privacy. Revealing more information to the server leads to efficiency, but hurts privacy, and vice versa. Exploring the design spectrum to balance these two properties leads to the following possible set of choices.

1) *No tags.* The basic design where no user-specific tag is attached to L2Is, and the user simply queries and decrypts all L2Is in the results for a location. This approach provides high privacy, but low performance.
2) *User ID tags.* The prior design where the server filters the L2Is in the response using the anonymized ID tags that the users attach with each L2I. This approach provides high performance, but low privacy.
3) *Keyed hash tags.* In this approach, each user u has a secret text T_u that she shares with her friends. The user u generates a new random string S_j for each new L2I she stores, and tags it with $< S_j, H(T_u.S_j) >$, where $H()$ is a hash function such as SHA1. So the L2I now contains $< (x', y'), E(i), S_j, H(T_u, S_j) >$. When a friend of u wants to query for a location (x, y), she transforms her location with u's secret to obtain (x', y'), and sends this point in a query. Then the index server sends all L2Is at (x', y') without any filtering. Upon receipt of the L2Is, the client user appends S_j in an L2I to T_u, and then compares the hash to check if it is indeed from user u. It would decrypt the L2I only if this hash check is passed. A similar check is performed on each L2I. Because of the fact that hashing is nearly two orders of magnitude (from our tests) faster than symmetric key decryption, this approach is significantly more efficient than *no tags* in terms of processing time on the user's device, while providing the same, strong privacy. We use HMACs [42] with proven security guarantees for implementing this.
4) *Random tags.* In this approach, each user u has another secret random number generator ($rgen_u$) that she shares with her friends. The user generates a new random num-

ber r_j from $rgen_u$ and attaches this tag to every new L2I she stores. The L2I now contains $< (x', y'), E(i), r_j >$. When a friend of u transforms her location (x, y) with u's secret to obtain (x', y') and sends this point in a query, the index server sends all L2Is at (x', y') without any filtering. Upon receipt of the L2Is, the friend checks if the random tag, r_j, in an L2I is within the set of random numbers generated by $rgen_u$. The friend only decrypts the L2Is whose tags are in this set. Since the membership check is faster than hashing (by about two orders of magnitude in our tests), this approach is more efficient than key-based hash tags, but requires some additional state. Specifically, the users need to exchange, with their friends, the maximum number of random tags (from their $rgen$) they have used so far in tagging L2Is. This helps them build the set of tags for checking L2Is. Thus this approach provides both high privacy and high efficiency. The idea of using random tags for fast decryption is also used in prior work on preserving privacy in accessing WiFi networks [43].

Both *keyed hash tags* and *random tags* nicely balance privacy and performance. We did construct several other mechanisms along similar lines to efficiently identify L2Is from friends while maintaining privacy, but we only discuss and evaluate these two due to space limitations. Fundamentally, all these mechanisms attach some additional tags to the L2Is, which can only be usefully interpreted by the friends. Since the server cannot link different L2Is from the same user, these mechanisms provide strong location privacy.

Querying the data server and decrypting location data. After obtaining the L2Is from the index server corresponding to a point (x', y'), say transformed with friend u's secrets, the client user identifies the L2Is from u (using the tags), and then decrypts the returned L2Is with u's symmetric key. Then the user directly queries the data server for the I2Ds corresponding to all the decrypted indices she cares about in a batch: $(i_1, i_2, ...)$. She then obtains the I2Ds from the data server, decrypts them using the symmetric key of the friend whose key was used to decrypt the corresponding index. And then the user consumes the data as per the application. There is no need for a proxy in this step as the index and the encrypted data on the data server cannot link a user to her location. Since the decrypted index is sent to the data server, it cannot even be linked to an encrypted index on the index server.

Supporting circular range and nearest-neighbor queries. The description so far was for point queries, where a user fetches data at a given location coordinate. These steps naturally extend to support more complex queries like circular range and nearest-neighbor queries. The key change necessary is for the index server to return data *around* a query point instead of returning data *at* a query point (as was done so far). Since our location transformation is distance preserving, building an R-tree [44], [45] on the L2Is input by the users can support both circular range and nearest-neighbor queries. Finally, the user should mention the type of the query she wants to run, while querying the index server. The rest of the steps in querying remain the same.

One issue in processing a nearest-neighbor query by querying at different transformed coordinates separately is that the index server will return each friend's nearest location data instead of nearest location data taken based on all friends' location data. As a result, additional answers that are not necessarily needed by users might be included. While our focus is not to explicitly remove those extra answers, one way to remove them is to specify a query range along with the query; another way is to let the users filter out such data after decryption.

5 Privacy Analysis

5.1 Intuition Behind Privacy in LocX

Here we describe the intuition behind LocX's privacy, and how it meets all of our requirements.

Defending against an attacker with access to data on the servers. The data stored on both servers do not reveal any information about their locations to the attacker. The L2Is on the index server contain transformed coordinates and the data on the data server are all encrypted. As a result, an attacker with access to just the data on these servers cannot de-anonymize the data to associate users with their locations.

Location privacy during server access. Even the attacker with access to monitor both servers cannot link accesses to the index and the data server because the indices stored on the index server are encrypted, but the indices are not encrypted on the data server. Only the users know how to decrypt the encrypted indices. Without the decryption keys, the attacker cannot link these records to figure out even the transformed location of the users accessing the servers.

Location data unlinkability. The I2Ds are encrypted, and the users access them only via indices. Hence users cannot be linked to any locations. The indices stored or accessed by a user are random numbers. The data server can link together the indices accessed by the same user, but this does not help the servers link the user to any locations. Finally, the users store and retrieve L2Is on the index server via proxies, so servers cannot link different transformed locations to the same user. Together, these provide location unlinkability.

5.2 Privacy During Location Data Access

Here we present a theoretical analysis of the privacy properties during data access in LocX. When a user accesses her friends' data by transforming her own location to different points in the transformed space and sending them in a query, a malicious index server learns the different, transformed coordinates that map to the same, real-world location (which is the user's current location). The question is whether an attacker could use this information to derive the user's real-world location. Here, we discuss the fundamental constraints we need to preserve in LocX to prevent the server from succeeding in such attacks.

Constraints in querying the index server. Assume first that the users directly access the index server, without any proxies. Each user has a secret angle, θ, and a secret shift, b, to transform her location coordinates. Suppose a user has n friends and she issues m location queries. In each of the m locations, (x_j, y_j), the user searches for n_j $(n_j \leq n, 1 \leq j \leq m)$

friends' information. Let us assume that all friends' information is queried at all m locations, and let us also assume the worst case scenario where the friends' transformed points are queried in the same order. Consider that the index server is malicious and sees the transformed coordinates of the user's friends, (x_{ij}, y_{ij}), in all m queries. The attacker (index server) then builds $(2n_1 + 2n_2 + ... + 2n_m)$ equations as follows (2 equations for each requested friend at one location) in order to solve $2m$ unknown real coordinates (x_j, y_j) and $2n$ unknown friends' secrets (θ_i, b_i), where $1 \leq j \leq m, 1 \leq i \leq n$.

$$\begin{cases} \cos\theta_i \cdot x_1 - \sin\theta_i \cdot y_1 + b_i & = & x_{i1} \\ \sin\theta_i \cdot x_1 + \cos\theta_i \cdot y_1 + b_i & = & y_{i1} \\ ... & = & ... \\ \cos\theta_i \cdot x_m - \sin\theta_i \cdot y_m + b_i & = & x_{im} \\ \sin\theta_i \cdot x_m + \cos\theta_i \cdot y_m + b_i & = & y_{im} \end{cases} \quad (1)$$

The total # of unknown variables is $2m + 2n$. For the attacker to solve all the unknowns, the following must hold:

$$2n_1 + 2n_2 + ... + 2n_m \geq 2m + 2n \quad (2)$$

$$\Rightarrow n_1 + n_2 + ... + n_m \geq m + n \quad (3)$$

So to protect the users' locations and friends' secrets from being inferred by the attacker, the reverse of Formula (3) must hold:

$$n_1 + n_2 + ... + n_m < m + n \quad (4)$$

If the users query all n friends' data at each location, $n_j = n$, a stronger version of Formula (4) must hold:

$$mn < m + n \quad (5)$$

We consider two special cases that satisfy Formula (5).

1) $m = 1$, meaning that the transformed coordinates of friends should be only observed in one location. In other words, the index server should not link multiple queries to the same user. This can be achieved by using proxies to anonymize user identities and ensure that the index server cannot link different user requests to the same user.
2) $n = 1$, meaning that the user is limited to access only one, different friend's data at each of the m locations. In other words, the functionality the user obtains from the applications is limited only to the data from an unreasonably low number of friends, in all the locations.

For the general cases of $m > 1, n > 1$, we decide to exploit the first case for our design, since we do not want to limit users (and hence to hurt functionality) as in the second case. By routing queries through proxies, we can easily satisfy Formula (5) since the index server cannot link different requests to the same user, as long as the proxies do not collude with the index server. Thus, we have proved that the unlinkability of queries due to proxies preserves users' privacy in LocX.

Impact of malicious proxies. We assumed in the previous analysis that all proxies are benign. However, a proxy may be malicious and collude with the index server, which would then violates the unlinkability of queries and hence violates users' location privacy. Therefore, multiple proxies are needed, and we need to control the number of queries any given proxy can see. Based on Formula (4), the upper bound on the average number of friends' data a user can request at a given location through a single proxy is

$$\frac{n_1 + n_2 + ... + n_m}{m} < \frac{m+n}{m} = 1 + \frac{n}{m} \quad (6)$$

In a worse case, more than one proxy may be malicious, and they may collude with the index server. Given the number of colluding proxies, k, we have to further limit the average number of friends' requests that a user can send per location via one proxy to: $(1+\frac{n}{m})/k = \frac{1}{k} + \frac{n}{mk}$. This number, however, becomes impractically small. We resolve this limitation by adding noise to queries that users send via proxies.

Improving privacy using noisy queries. Now we derive the amount of noise to add per query. Following Formula (6), if k proxies are colluding, together they can see $min\{k \cdot (1+\frac{n}{m}), n\}$ friend requests from the same user at one location (n is the maximum number of friend requests of their interest for one location query), which violates Formula (4) and hence the user's location privacy. To make sure the colluding proxies cannot break Equation (1), we need to increase the number of unknowns on the right side of Formula (4). This is achieved by generating "dummy" friend requests based on false secrets (θ', b'). The user uses these false secrets to generate false location points, inserts the random points along with the user's legitimate transformed points and routes them via the proxies. The colluding proxies may then attempt to solve the equation without knowing which requests are real and which ones are fake. But since the solution to the equation is then based on dummy random points, the attacker will *not have the right secrets* for the user's friends.

Let the minimum number of such noisy points be n', and the user asks for $(n + n')$ friends' data in m location queries. Then we should have $min\{k \cdot (1 + \frac{n}{m}), n\} = 1 + \frac{n+n'}{m}$, from which we get $n' = min\{(k-1) \cdot (m+n), mn - m - n\}$. For each of the m locations, the *minimum* amount of noise that the user needs to generate on an average is

$$\frac{n'}{m} = min\{(k-1) \cdot (1 + \frac{n}{m}), n - 1 - \frac{n}{m}\} \quad (7)$$

Note that *1)* the overhead due to noise is proportional to the amount of collusion in the system when the number of colluding proxies k is $\leq \frac{mn}{m+n}$. The amount of noise does not increase beyond $n - 1 - \frac{n}{m}$ when $k > \frac{mn}{m+n}$, as all the requests sent out by a user are learned by the attackers by now. And *2)* the noise added above is an average value. We just need to ensure that the noise added over m points averages to this value – the noise does not have to be the same in each query. We show in our evaluation the exact amount of noise added in our setup for real values of parameters; and our results show that the overhead due to the noise is very low.

Finally, we stress that even if noise is not added, the worst that the attacker can do is to break a single user's location privacy – but not the location privacy of all her friends. Moreover, even if users do not generate enough noisy friend requests as the number specified in Formula (7), and the attackers or malicious proxies are able to solve Equation (1), it is still not easy for them to build the correct association between a real friend and a pair of secrets obtained from the solution, since there are $\binom{n+n'}{n} \cdot n!$ possible associations. Hence even in this worst case, the user's friends' secrets are still kept secure. In this case, only the user's current location is revealed to the attacker (from the solution to the Equation (1)), hence only this user's location privacy is violated. This privacy is also not violated arbitrarily long, but only so long as the proxies continue to collude and associate the requests coming to the index server to the same victim user.

5.3 Other Attacks and Defenses

We now discuss other possible attacks the servers can perform, in addition to the attacks described before, and our proposed solutions to ward off these attacks.

Query linking attacks by the index server. The index server might attempt to link the queries from the same user using some query "fingerprints." For instance, the server might guess that all queries with 199 points in them (one per friend) belong to the same user – assuming that it is uncommon to have many users that use the same proxy having 199 friends. Fortunately, our extension of adding noise to the requests helps here. Since the number of noisy points added varies per query, the server cannot perform such attacks.

Fingerprinting using cookies in incoming connections. We assume that the proxies or the clients scrub the outgoing connections, using tools such as Privoxy [46], to remove all user-identification information from the connection. This assumption is common to all anonymity-preserving systems, including Tor [34]. Thus such attacks do not work on LocX.

Localization-based attacks. As the users connect to the data server directly, it can attempt to learn users' location using their IPs. Fortunately, the location from localization technologies is at the granularity of tens of miles [47]. To prevent these attacks, accessing the server via proxies helps, but this reduces the efficiency of the system. Recently proposed [48] mechanisms can also help us in reducing the localization accuracy of the server and even defeating these attacks.

Timing attacks by the index server. The index server may attempt to link different requests that arrive at the server to the same user or query using timing information. For example, the server can say that all requests for I2Ds within a second belong to the same user, and hence all such I2Ds are related. Fortunately, we can leverage prior work on location privacy here [49], [11], [12], [13], [22]. By using techniques such as batching requests and randomly delaying requests to the server at the proxies or at the clients, and by combining them with noisy queries described before, we can deter these attacks.

Periodically refreshing a user's secrets. So far we described LocX as if only one pair of secrets (θ, b) is used by a user to protect her data. But we can easily extend this to allow users to use time-varying secrets. For *e.g.*, Alice may use (θ, b) to protect her data from the year 2010, and (θ', b') (generated using a pseudorandom number generator) to protect 2011's data. The time period for secret refresh can also be configured by the user. The user could then share new secrets with selected friends for better security (as described next).

Attacks due to stolen or compromised user devices and colluding friends. An attacker with access to a user's secrets, obtained by compromising or stealing her device or by colluding with one of her friends, obviously, has access to all her data and her friends' data. Unfortunately, this is a natural problem shared by other prior social systems that have relied on friends for performing certain tasks [41], [39], [40], [37]. That said, we believe that the damage from this problem can be limited. First, a colluding user can only leak her friends' data to the server and not more. So obtaining network-wide visibility for an attacker will require a large number of colluding users, which is hard. Second, using attributed-based encryption (ABE) [50], similar to the approach in Persona [37], can help limit the damage of this attack. With ABE the attacker will have to get many colluding friends (and access all attributes) to even obtain the data of even a single user. Finally, the users can easily revoke the keys of a friend suspected to be colluding or compromised, and periodically refreshing the keys (like discussed above) of all friends forces friends to re-request keys, thus providing an opportunity for users to "prune" their network to only the trusted friends (and also limiting future damage from already leaked keys).

Attacks using external information. Attackers can mount several attacks on targeted users using information learned about them from outside our system. For *e.g.*, Bob, an employee of a restaurant, might know Alice's home address and know that it takes 10 minutes for her to come from her home to the restaurant. Knowing two locations of Alice (home and restaurant) and the time window when transformations of these locations are stored on the server, Bob might collude with the server to try to figure out Alice's secrets.

While defending against all such attacks based on external information is extremely challenging, and is outside the scope of this paper, we offer our intuitions as to why such attacks are especially difficult against LocX. First, this attack can work only on those users whose information is precisely known by the attacker. The number of such users is usually very limited. Second, our defenses against timing attacks can significantly increase the time window the attacker has to process. And the attacker will have to process all the points uploaded to the system in that time window, which can be extremely large in a system with many users. Third, even if successful, the attacker will have to keep running this attack due to time-varying secrets of the users. Finally, just learning a user's secret does not compromise her privacy. The attacker still has to break the unlinkability of every (future) request sent by this user (by colluding with proxies). That is, even when the secrets of a user (θ, b) are broken, the attacker still needs to link future requests to that user.

Map matching attacks. The attacker might attempt to connect the points in the virtual coordinate space, to construct paths taken by a user, and then to map them back to the paths in the real-world map. Doing so would enable the attacker to identify real-world paths traversed by a user. However, such attacks are impractical for the attacker to mount due to the following reasons. First, isolating the points that belong to a given user is very hard. The virtual coordinate space is shared,

Measures	LocX	L2D	K-Anonymity
Client Processing Time (ms)	0.0045	0.0	0.0
Query Completion Time (ms)	0.009	0.0014	0.004
Data Communication Size (Bytes)	140.1	85	169

TABLE 2
Measures of Location Puts

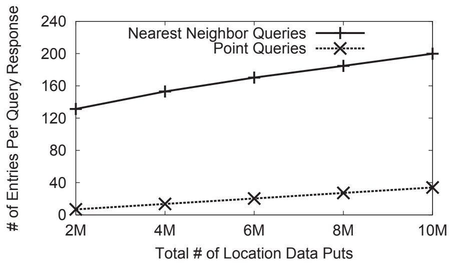


Fig. 3. Effect of varying the number of puts on query response sizes in synthetic data.

and all users' points overlap in this space. As a result, a set of points in a region of the space can lead to an extremely large number of total paths. Due to unlinkability of points, the attacker would not know which path among these is the path taken by the user. Second, due to our defenses against timing attacks, the points in a path may not appear in the right order at the server. So even if the user's points are isolated by the attacker, they will lead to a wrong path fingerprint. Finally, mapping a path fingerprint to the right path in the world's map is not trivial. (Figure 9 in evaluation validates this argument.)

Denial-of-Service attacks by malicious users. To prevent DoS attacks on the server behind the cloak of anonymity, we can leverage existing research [51], where *tokens* are used to verify that the *put*s (store operations) are from legitimate users and hence rate-limit malicious puts.

Summary. To break a user's location privacy in LocX, the attackers need to surpass two steps: a) learn the secrets of the user, every time she changes it, and b) correctly identify every request sent by the user. From our analysis above, doing so is very expensive for the attacker, and hence, LocX significantly improves location privacy over prior work.

6 Evaluation

Our evaluation focuses on answering the following key questions. *1)* What is the overhead of a *put* in LocX? *2)* What is the overhead of retrieving point and nearest-neighbor queries in LocX compared to today's systems? And how does it vary when more data is retrieved per query? *3)* How does the overhead vary when more noise is added to each query? *4)* How does the overhead from L2Is and I2Ds change when larger size of data is stored per *put*? *5)* Finally, how does LocX perform on mobile devices?

6.1 Implementation and Setup

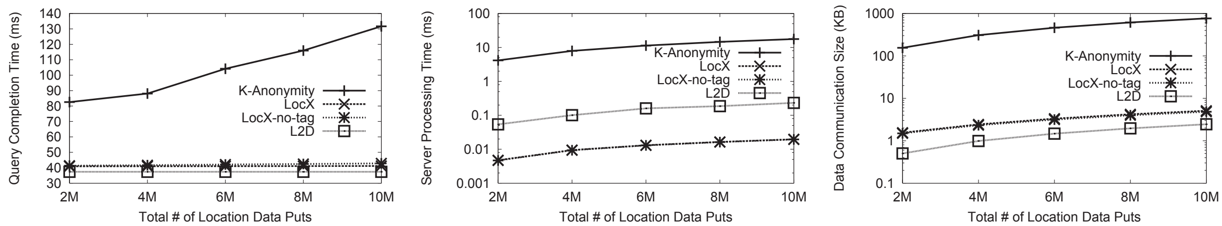
We implemented LocX in Java. We used AES with 128 bits keys for encryption and decryption. The implementation of nearest-neighbor queries was based on the R$*$-tree package from HKUST [45]. We configured each user to cache 1000 random number tags from each of her friends.

We measured LocX's performance on both desktops and on Motorola Droid mobile phones. The index and data servers were run on the same Dell PowerEdge server equipped with Quad Core Xeon L5410 2.33Ghz CPU, 24GB RAM and 64 bit Federal Core 8 kernels. Clients were run on another machine with the same configuration. We used the same code base for both desktop and mobile tests. But we had to modify the code slightly for Android OS to deal with some missing libraries. In addition, we had to make certain optimizations to limit the memory usage to under 16MBs for LocX process in Android.

Workload. We used both synthetic and real-world LBSA workload datasets for our tests. The synthetic dataset with default parameters was created following empirical observation on popular geo-social sites such as FourSquare: First, we partitioned a two dimensional space into 100 cells, each of which is a city. In each city, we randomly generated 100 pairs of location coordinates. Then we assigned 1000 resident clients to each city. Each client had 100–1000 friends following a power law distribution with $\alpha = 1.5$ [52], among whom 70% friends were from the same city as the client and 30% were from other cities. Each client did 20 location puts, among which 70% puts were at locations in the client's resident city and 30% were at locations in other cities. Each location put message was randomly generated consisting of maximum 140 bytes, following the tweets in Twitter. As a result, each city had 20K location puts on average, and the total number of location puts was 2M. After all the puts, each client submits a point query and a nearest-neighbor query with 70% probability of being within the client's resident city and 30% probability of being in other cities. Each nearest-neighbor query requests for 10 nearest locations (we only evaluate nearest-neighbor queries, as we found in our preliminary tests that the performance of circular range queries to be similar to that of nearest neighbor queries). We set noise to a fixed 10 points per query for now, and study the impact of noise later.

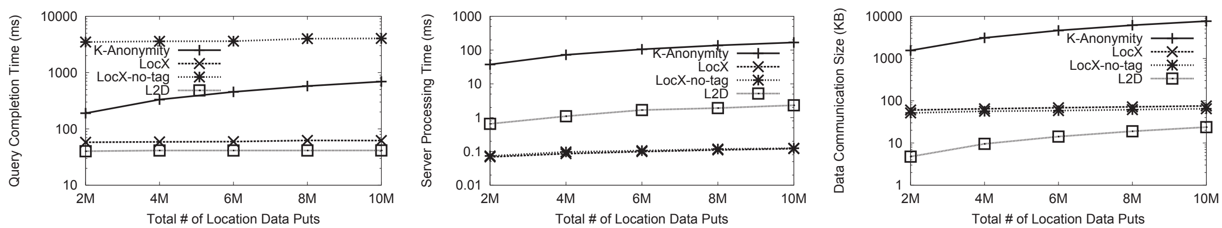
We crawled www.brightkite.com for real LBSA traces. We crawled using BrightKite's public APIs, at a rate slower than the rate specified in the API Terms of Use. Due to the slow rate, we distributed the crawling tasks to 20 machines, and crawled for about a month starting from August 20th, 2010. Starting with an initial seed of users, we crawled each user's profile, friends list, and check-in data. The crawled data in total had 25,314 users, 123,438 unique GPS coordinates with 259,775 check-ins by users. While using this data for experiments, we treated each check-in as a location put, and let each user query from one of her check-in locations. Since check-in messages were not available for us to crawl, we generated random messages of varying sizes.

Experiment setup. To evaluate the overhead that our approach is adding to today's LBSAs with no privacy, we compared LocX with random tags, referred to as *LocX*, with an implementation of a today's service that has social network on the server and directly maps a location to its data, referred to as *L2D*. In L2D, data is in plain-text, thus no encryption or decryption is needed. We measured the communication costs



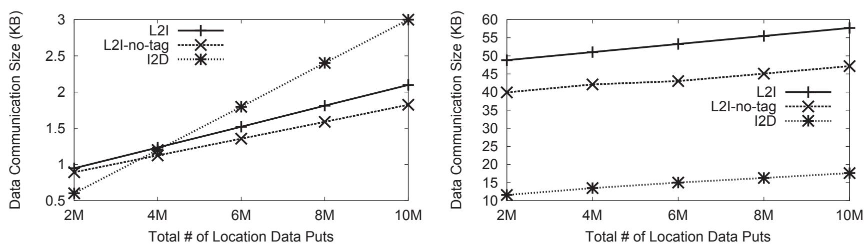
(a) Query Completion Time (b) Server Processing Time (c) Data Communication Size

Fig. 4. The various costs of running point queries, while varying the number of location puts in synthetic data.



(a) Query Completion Time (b) Server Processing Time (c) Data Communication Size

Fig. 5. The various costs of running nearest-neighbor queries, while varying the number of location puts in synthetic data.



(a) Point Queries (b) Nearest Neighbor Queries

Fig. 6. Breaking down the communication overhead from L2Is and I2Ds, when the number of puts is increased.

between clients and servers, the client processing time, the query completion time (including network latency), and the server processing time. To evaluate the performance trade-offs of the design choices we have discussed, we also compared LocX with random tags against LocX with no tags, which we call *LocX-no-tag*. Since these two different designs result in differences in processing L2Is, we specifically measured the communication cost between clients and the index server for L2I and the communication cost between clients and the data server for I2D.

Comparison. We also compare LocX to a recent k-Anonymity-based system [12], where a trusted third-party anonymizer stands between the clients and servers to anonymize queries and to filter query responses. The anonymizer knows the userId and the user's friends' information in order to filter responses. Since the anonymizer hides a user's query in a larger area of k other users, we can expect its data communication size and server time (including the anonymizer time) to be significant, in comparison to LocX, as shown in detailed next. We set $k = 10$ in all our experiments.

6.2 Experimental Results

We report results from our tests on desktop computers first, and present experimental results on mobiles later.

Performance of a location put. We present the cost of a single location put in synthetic dataset in Table 2. A put in today's system (L2D) costs no processing time on clients as there is no crypto operation. But we can see that a put in LocX with encryption and additional index data only slightly increases the overhead, which is not even observable by users. The average message size was 84.5 in L2D, but it was increased to 140 in LocX. k-Anonymity, however, has even higher size due to the information regarding the cloaked spacial region in the message.

Query performance with increase in the # of puts. Next we compared the performance of LocX (with random tags), LocX with no tags, k-Anonymity, and L2D for point queries and kNN queries. On synthetic dataset, we varied the number of location puts per client from 20 to 100, while fixing the amount of noise in a query to default 10 and message size to default maximum 140. Total number of clients was fixed at 100K. As location puts per client increases, the total data size increases, thus more data needs to be processed and the sizes of query answers increase. Figure 3 shows the increase in query answer sizes. Obviously, the response to a kNN query contains more data than a point query (by more than 6 times).

From Figures 4(a) and 5(a), we see that processing a query in LocX takes is comparable to that of L2D, in a LAN setting. However, the other two approaches – k-Anonymity and 'Locx-no-tag' – perform poorly. k-Anonymity has higher overhead as the entire cloaked spacial region is included in the responses, which leads to increase in the query completion time, and server processing time or load (shown in Figures 4(b) and 5(b)). In 'LocX-no-tag', a client cannot differentiate between friends' and non-friends' messages, so the client tries to decrypt every single message received, which leads to costly computation and time to completion. This problem becomes particularly worse while processing nearest-neighbor queries, as shown in 5(a). The server time of LocX is actually better than L2D due to the fact that the application logic is moved to the clients and server simply needs to do lookups. The communication cost of LocX is no more than 3 times the communication cost of L2D for point queries and no more than 7 times the communication cost of L2D for nearest-neighbor queries, as shown in Figures 4(c) and 5(c) respectively.

We also measured the client processing times. LocX, as expected, pays a slight processing cost on the client side in decrypting indices and location messages. But we find that this increase in overhead is actually negligible. Due to space limitation, we leave out the graphs for synthetic data but later present the results on real data set in Figure 10. The results are similar in both cases.

Individual overhead from L2I and I2D. Now we look into the overhead from L2I and I2D separately. Overhead from L2I in the setting where no tags are attached is referred to as 'L2I-no-tag'. We see in Figure 6(a) that as the number of puts increases, more data is returned as answers, and the communication cost of I2D increases more than that of L2I for point queries. But in the case of nearest-neighbor queries, since a lot of data needs to be filtered in L2I phase, more data is transmitted for L2Is. In contrast, only qualified answers are transmitted in I2D phase. As a result, the communication cost of L2I is more than that of I2D (see Figure 6(b)).

Varying put message sizes. We next increased the put message size from 140 to 700, while fixing the other parameters (20 puts per client). We expected only the communication cost of I2D to increase but the cost of L2I to remain the same in this test. Figure 7 confirms this for point queries, and we observed similar behavior for nearest-neighbor queries (no graph shown due to space constraints). Clearly, as the message size increases, larger sizes of data is transmitted as answers, thus the cost of I2D gradually dominates that of L2I.

Varying the amount of noise in queries. We next varied the amount of noise added per query from 10 to 50, while setting the other parameters to default. Figure 8 shows that increasing the noise only increases the communication overhead from L2I, and this increase in overhead is quite small. There is no increase in I2D overhead due to noise. Also note that noise does not increase the computation time on client devices, as clients can reject responses to noisy points and not even attempt to decrypt them. The trend for kNN queries is similar, but the graph is left out due to lack of space.

Distribution of Transformed Coordinates. Figure 9 compares the distribution of 2 million real-world locations after user-specific transformations to understand how the points are distributed. Clearly, after transformation, points are evenly distributed in the virtual space, as can be seen from the CDF plot, which is practically a vertical line near 0. Thus the virtual coordinate density is negligible compared to real-world coordinates. This is the main reason, we believe, why reverse engineering the transformations and performing Map-Matching attacks is hard.

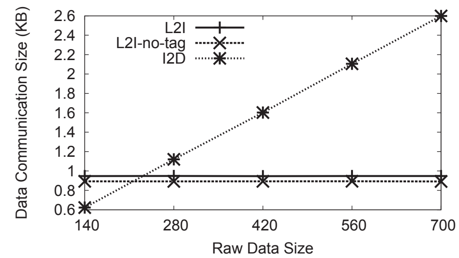


Fig. 7. Increase in the data (I2D) transfer size when the message size per location data put is increased in synthetic data.

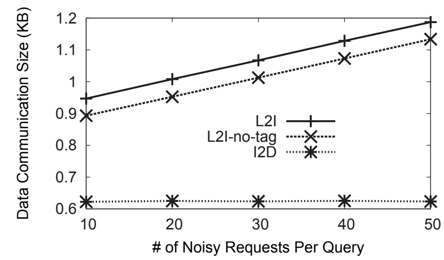


Fig. 8. Increase in the L2I communication overhead due to increase in the noise, for point queries in synthetic data.

Experiments with real-world BrightKite datasets. Since we were not able to crawl the messages in check-ins, we generated messages of size varying from 140 to 700 bytes, and then used the check-in locations to put this data on the server. We set the noise in the queries to default value 10. This real-world data had a lot fewer check-ins compared to our synthetic data, and hence the number of results returned in query responses was also smaller. The average answer size for a point query and a nearest-neighbor query were around 0.92 and 36.5 respectively. We learned from this test that the performance trend of LocX with real data is similar to that on synthetic data. Figure 10 shows that LocX does not incur too much processing overhead on real data either. Increasing the message size increases the processing time only slightly due to decryption of larger sized data. Due to similar trends, we leave out the other graphs on point and kNN queries.

We also notice that LocX with no tags consumes more processing than LocX with tags; the majority of time here is spent in trying different friend keys for decrypting each L2I. This clearly shows that tags are necessary to boost the performance of LocX, with only a slight more communication overhead.

Next we used this real-world data to get a realistic estimate of the amount of noise added by LocX according to Formula 7. We set n to 7.17, the average # of friends in the BrightKite dataset, and m to 10.95, the average # of check-ins per user. The number of noisy points a user with this n and m has to add

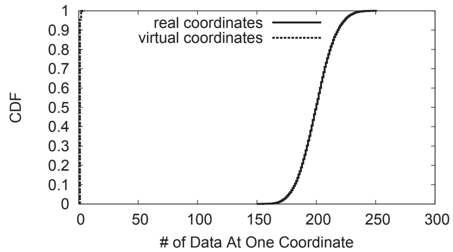


Fig. 9. Density distribution of location coordinates in the real-world and the transformed world with user-specific keys.

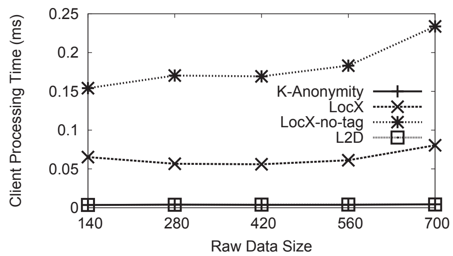


Fig. 10. The increase in the processing overhead for point queries in BriteKite dataset, for increase in put message size.

while using LocX with the increase in the number of colluding proxies k is shown in Figure 11. The noise increases up to a certain value of k and then remains constant, as expected (explained in Section 5.2). In addition, we see that even the maximum noise added (5.53) is less than the value of n. Noise of 5.53 location coordinates translates to about 22 bytes per query. If the average message size is 500 bytes, then this overhead is about 4%, which we think is reasonable.

Overhead of running LocX on Motorola Droid. We ported LocX to Android, and ran the experiments under synthetic data on Motorola Droids. We observed similar trends in our tests as the results reported before (in Figures 4 and 5). As a result, we do not present new graphs. The key difference, however, was that the client processing time is much slower on Droids due to low resources. In the default setting with 20 location puts per client and one point query per client (described in § 6.1), the average client processing time on Droids was about 10 times slower than on the Dell server. But even after this slow down, the query completion time on Droids were below .2 seconds for point queries, and all kNN queries were answered in below a second. We measured the power consumption on Droids and noticed that the phone can process about 40K point queries before the battery was completely consumed.

Summary. We find in our evaluation that LocX can run on today's mobile devices with low computation and communication cost and still provide strong location privacy.

7 Building Applications Using LocX

Here we sketch how to build LBSAs using LocX. We demonstrate the usage of our APIs by building three applications. In today's systems that provide these services, the data is entrusted to the server in plain-text, which performs the computations in the application logic. But since we do not

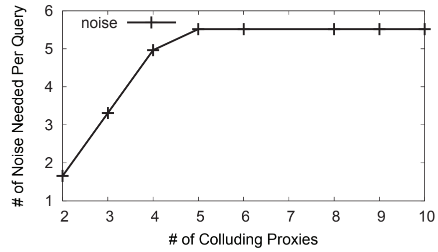


Fig. 11. Ideal amount of noise necessary to protect users in BrightKite, with increase in the number of malicious proxies.

trust the server in LocX, the application logic that computes on the plain-text location data is moved to the client.

Location-based reminders. This application users place reminders for friends at specific locations (for *e.g.* reminder to buy milk near a grocery store), and when the friends are at that location, an alert is generated on their device. To build this application in our model, a user bundles all the details about the reminder, such as the reminder text and time, encrypts the whole bundle and generates a corresponding I2D. Then the user transforms the reminder location based on the friend's secret and generates a corresponding L2I. These pieces are stored on the servers with a *putL2I* and a *putI2D* calls. Each user periodically runs a neighborhood query for data from her friends. First the user takes her current location, transforms it according to her secret, runs a neighborhood query, and fetches the L2Is and I2Ds, if any, using the *getL2I* and *getI2D* calls. Then the device decrypts and reminds the user as appropriate.

Location-based recommendations. This application aims to recommend nearby sites (restaurants, shopping malls, etc.) to users based on the reviews given to these sites by their friends. In our model, this application is built as follows. A user stores her reviews by generating a bundle containing all the information related to the review, such as the review text, rating, etc., encrypts the bundle using her symmetric key, and generates a L2I and I2D using the data. The locations of the sites are transformed, of course, while generating the L2Is. This information is then stored on the servers using the *putL2I* and *putI2D* calls. The application on each user's mobile downloads the data from her friends at the user's current location by running a neighborhood query. Then it decrypts the returned data, and plots the recommended sites on a map in the device. Thus, the application operates without even revealing users' location to the servers.

Friend locator. This application alerts a user whenever a friend is in the vicinity. When this application is built on LocX, users check-in at their current location periodically; then users check for friends in the vicinity by running a neighborhood query around their current location and decrypting check-ins from friends in recent times (*e.g.* last ten minutes). Despite using neighbor query, this approach to building friend locator is still efficient. Even a hotspot (*e.g.* a concert) in the real coordinate space is usually *not a hotspot* in the transformed coordinate space due to user-specific location transformations, and thus limits the amount of (irrelevant) data received and processed by a user.

8 Conclusions

This paper describes the design, prototype implementation, and evaluation of LocX, a system for building location-based social applications (LBSAs) while preserving user location privacy. LocX provides location privacy for users without injecting uncertainty or errors into the system, and does not rely on any trusted servers or components.

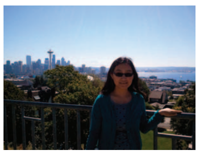
LocX takes a novel approach to provide location privacy while maintaining overall system efficiency, by leveraging the social data-sharing property of the target applications. In LocX, users efficiently *transform* all their locations shared with the server and encrypt all location data stored on the server using inexpensive symmetric keys. Only friends with the right keys can query and decrypt a user's data. We introduce several mechanisms to achieve both privacy and efficiency in this process, and analyze their privacy properties.

Using evaluation based on both synthetic and real-world LBSA traces, we find that LocX adds little computational and communication overhead to existing systems. Our LocX prototype runs efficiently even on resource constrained mobile phones. Overall, we believe that LocX takes a big step towards making location privacy practical for a large class of emerging geo-social applications.

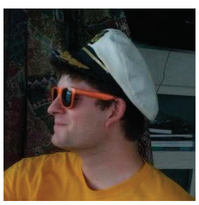
References

[1] M. Motani, V. Srinivasan, and P. S. Nuggehalli, "Peoplenet: engineering a wireless virtual social network," in *Proc. of MobiCom*, 2005.
[2] M. Hendrickson, "The state of location-based social networking," 2008.
[3] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *Proc. of SenSys*, 2008.
[4] G. Ananthanarayanan, V. N. Padmanabhan, L. Ravindranath, and C. A. Thekkath, "Combine: leveraging the power of wireless peers through collaborative downloading," in *Proc. of MobiSys*, 2007.
[5] M. Siegler, "Foodspotting is a location-based game that will make your mouth water," http://techcrunch.com/2010/03/04/foodspotting/.
[6] http://www.scvngr.com.
[7] B. Schilit, J. Hong, and M. Gruteser, "Wireless location privacy protection," *Computer*, vol. 36, no. 12, pp. 135–137, 2003.
[8] F. Grace, "Stalker Victims Should Check For GPS," Feb. 2003, www.cbsnews.com.
[9] DailyNews, "How cell phone helped cops nail key murder suspect secret 'pings' that gave bouncer away," Mar. 2006.
[10] "Police: Thieves robbed homes based on facebook, social media sites," WMUR News, September 2010, http://www.wmur.com/r/24943582/detail.html.
[11] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *Proc. of Mobisys*, 2003.
[12] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new casper: A privacy-aware location-based database server," in *ICDE*, 2007.
[13] B. Gedik and L. Liu, "Location privacy in mobile systems: A personalized anonymization model," in *Proc. of ICDCS*, 2005.
[14] T. Jiang, H. J. Wang, and Y.-C. Hu, "Preserving location privacy in wireless lans," in *Proc. of MobiSys*, 2007.
[15] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias, "Preventing location-based identity inference in anonymous spatial queries," *TKDE*, 2007.
[16] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: anonymizers are not necessary," in *SIGMOD Conference*, 2008.
[17] S. Papadopoulos, S. Bakiras, and D. Papadias, "Nearest neighbor search with strong location privacy," *PVLDB*, 2010.
[18] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh, "Location privacy via private proximity testing," in *Proc. of NDSS*, 2011.
[19] G. Zhong, I. Goldberg, and U. Hengartner, "Louis, lester and pierre: Three protocols for location privacy," in *Proc. of PET*, 2007.
[20] N. Daswani and D. Boneh, "Experimenting with electronic commerce on the palmpilot," in *Financial Cryptography*. Springer, 1999.
[21] A. Khoshgozaran and C. Shahabi, "Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy," in *Proc. of SSTD*, 2007.
[22] G. Ghinita, P. Kalnis, and S. Skiadopoulos, "Prive: anonymous location-based queries in distributed mobile systems," in *Proc. of WWW*, 2007.
[23] P. Golle and K. Partridge, "On the anonymity of home/work location pairs," in *Proc. of Pervasive Computing*, 2009.
[24] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady, "Enhancing security and privacy in traffic-monitoring systems," in *IEEE Pervasive Computing Magazine*, 2006.
[25] B. Hoh *et al.*, "Preserving privacy in gps traces via uncertainty-aware path cloaking," in *Proc. of CCS*, 2007.
[26] J. Krumm, "Inference attacks on location tracks," in *Proc. of Pervasive Computing*, 2007.
[27] A. Beresford and F. Stajano, "Mix zones: User privacy in location-aware services," in *Proc. of Pervasive Computing*, 2004.
[28] M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu, "Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services," in *Proc. of ICDE*, 2008.
[29] D. Lin, E. Bertino, R. Cheng, and S. Prabhakar, "Position transformation: a location privacy protection method for moving objects," in *Proc. of Security and Privacy in GIS and LBS*, 2008.
[30] C.-Y. Chow and M. F. Mokbel, "Enabling private continuous queries for revealed user locations," in *SSTD*, 2007, pp. 258–275.
[31] E. O. Turgay, T. B. Pedersen, Y. Saygin, E. Savas, and A. Levi, "Disclosure risks of distance preserving data transformations," in *Proc. of SSDBM*, 2008.
[32] S. Mascetti, C. Bettini, and D. Freni, "Longitude: Centralized privacy-preserving computation of users' proximity," in *Proc. of SDM*, 2009.
[33] S. Mascetti, C. Bettini, D. Freni, X. S. Wang, and S. Jajodia, "Privacy-aware proximity based services," in *Proc. of MDM*, 2009.
[34] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *USENIX Security Symposium*, 2004.
[35] H. Hu, J. Xu, C. Ren, and B. Choi, "Processing private queries over untrusted data cloud through privacy homomorphism," in *ICDE*, 2011.
[36] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *SIGMOD Conference*, 2009.
[37] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: An online social network with user defined privacy," in *Proc. of SIGCOMM*, 2009.
[38] T. Ristenpart, G. Maganis, A. Krishnamurthy, and T. Kohno, "Privacy-preserving location tracking of lost or stolen devices: Cryptographic techniques and replacing trusted third parties with DHTs," in *Proc. of USENIX Security Symposium*, 2008.
[39] A. Mislove, K. Gummadi, and P. Druschel, "Exploiting social networks for internet search," in *Proc. of HotNets*, 2006.
[40] A. Mislove, A. Post, P. Druschel, and K. Gummadi, "Ostra: Leveraging trust to thwart unwanted communication," in *Proc. of NSDI*, 2008.
[41] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson, "Privacy-preserving p2p data sharing with oneswarm," in *SIGCOMM*, 2010.
[42] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *CRYPTO96*.
[43] B. Greenstein, D. McCoy, J. Pang, T. Kohno, S. Seshan, and D. Wetherall, "Improving wireless privacy with an identifier-free link layer protocol," in *Proc. of MobiSys*, 2008.
[44] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proc. of SIGMOD*, 1984.
[45] D. P. group, "R-tree java implementation," http://www.rtreeportal.org/code/Rstar-java.zip.
[46] "Privoxy web proxy," http://www.privoxy.org/.
[47] B. Wong, I. Stoyanov, and E. Sirer, "Octant: A comprehensive framework for the geolocalization of Internet hosts," in *Proc. of NSDI*, 2007.
[48] P. Gill *et al.*, "Dude where's that IP? Circumventing Measurement-based IP Geolocation," in *USENIX Security Symposium*, 2010.
[49] J. Manweiler, R. Scudellari, and L. P. Cox, "Smile: Encounter-based trust for mobile social services," in *Proc. of CCS*, 2009.
[50] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *CCS*, 2006.
[51] K. P. N. Puttaswamy, R. Bhagwan, and V. N. Padmanabhan, "Anonygator: Anonymity and Integrity Preserving Data Aggregation," in *Proc. of Middleware*, 2010.
[52] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proc. of IMC*, Oct 2007.

Krishna Puttaswamy is currently a researcher in Bell Labs. He received M.S. and Ph.D. degrees in Computer Science from the University of California, Santa Barbara in 2010. Before that he received B.S. degree in Computer Science from the National Institute of Technology, Karnataka, India, in 2003. His research interests are in the areas of cloud computing, security and privacy.



Shiyuan Wang is a PhD Candidate in Computer Science at the University of California Santa Barbara. Her recent research interests are in data security and privacy. She is a recipient of UCSB Dissertation Fellowship.



Troy Steinbauer graduated with his Masters of Science in Computer Science at the University of California, Santa Barbara in 2011 and is currently a Senior Software Engineer at Zynga. While attending UCSB, Troy was a Research Assistent for the CURRENT Lab under Professor Ben Zhao.

Divyakant Agrawal is a Professor of Computer Science at University of California, Santa Barbara. His research expertise is in the areas of database systems, distributed computing, data warehousing, and large-scale information systems. He has served in the program committees of many leading conferences and served as the Program Chair of ACM SIGMOD'2010 and the General Chair/Co-Chair of ACM SIGPATIAL'2009, 2010, and 2011. Prof. Agrawal is a Fellow of both IEEE and ACM.

Amr El Abbadi is a Professor of Computer Science at University of California, Santa Barbara. His research interests lie in the area of scalable database and distributed systems. Prof. El Abbadi is a Fellow of the ACM. He was Vice Chair of ICDCS 1999, Vice Chair for ICDE 2002, the Americas Program Chair for VLDB 2000, the Program Co-Chair for SIGSPATIAL 2010, and the Program Co-Chair for ACM SoCC 2011.

Christopher Kruegel is an associate professor and holder of the Eugene Aas Chair in Computer Science at the University of California, Santa Barbara. He's also involved in the International Secure Systems Lab. His research interests include most aspects of computer security, with an emphasis on malware analysis, Web security, network security, and vulnerability analysis.

Ben Y. Zhao is an Associate Professor of Computer Science at University of California at Santa Barbara. Before UCSB, he completed his M.S. and Ph.D. degrees in Computer Science at U.C. Berkeley (2000, 2004), and his B.S. degree from Yale University (Branford, 1997). His research spans the areas of networking, security and privacy, distributed systems, simulation and modeling. His recent interests are in online social networks, cognitive radios and wireless data centers. He is a recipient of the National Science Foundation's CAREER award, MIT Tech Review's TR-35 Award and ComputerWorld's Top 40 Technology Innovators.