Preserving Privacy in Location-based Mobile Social Applications

Krishna P. N. Puttaswamy and Ben Y. Zhao
Computer Science Department, U. C. Santa Barbara

ABSTRACT

Location-based social applications (LBSAs) rely on the location coordinates of the users to provide services. Today, smartphones using these applications act as simple clients and send out user locations to untrusted third-party servers. These servers have the application logic to provide the service, and in the process collect large amounts of user location information over time. This design, however, is shown to be susceptible to large-scale user privacy compromises even if several location cloaking techniques are employed. In this position paper, we argue that the LBSAs should adapt an approach where the untrusted third-party servers are treated simply as encrypted data stores, and the application functionality be moved to the client devices. The location coordinates are encrypted, when shared, and can be decrypted only by the users that the data is intended for. This approach significantly improves user location privacy. We argue that this approach not only improves privacy, but it is also flexible enough to support a wide variety of location-based applications used today. In this paper, we identify the key building blocks necessary to construct the applications in this approach, give examples of using the building blocks by constructing several applications, and outline the privacy properties provided by this approach. We believe our approach provides a practical alternative design for LBSAs that is deployable today.

1. INTRODUCTION

With the proliferation of the Internet-enabled smartphones, location-based mobile social applications (LBSAs) have seen wide-spread adoption. These applications empower mobile users with the knowledge of their vicinity, which significantly improves user productivity in a variety of contexts ranging from work and personal life to health and travel. For example, these applications enable users to meet with friends in the surroundings [18,20,21], select restaurants and stores that have good reviews from friends [13], help select routes based on traffic information, place reminders for friends [27], download content faster [2] (in collaboration with friends' devices), among others. LBSAs are used by millions of users today [1], and the tremendous penetration of mobile devices will only increase this number. While the benefits from these applications make a compelling case for users to share their location, location privacy is a primary concern in doing so.

HotMobile'10, February 22–23, 2010, Annapolis, Maryland

Location information in the wrong hands severely hurts users. Real world examples exist where mobile users are harmed by stalkers economically [23], physically [11], and legally [7], by tracking users with such location information. Recent studies have shown that trivial algorithms can expose sensitive locations of users such as home and office locations even using anonymized GPS traces collected by location-based services [10, 14]. Such attacks can also identify users' visits to hospitals, clubs, or other embarrassing locations [23]. Further, it is possible to discover real identities of users (in addition to pseudonyms, username, etc.) using reverse white page lookups [16] in collaboration with anonymized GPS traces. Advanced attacks allow attackers to infer users' mode of transport (bus, car, foot), and predict the route taken by users [9], etc. Finally, these attacks are successful even on applications that sporadically send location information to servers [17], and despite several existing defenses based on spatial and temporal cloaking of user location [12].

We observe that the key reason that makes the location-based social applications (LBSAs) vulnerable to such large-scale privacy loss is that they all inherently depend on user location coordinates, which users entrust to applications running on untrusted third-party servers in plain-text. These servers consume this location and provide application-specific services to the users. However, the location coordinates reveal sensitive location information about users, and the untrusted servers can easily leak this data in large amounts due to software bugs, operator errors or due to active attacks, thus compromising location privacy of thousands of users *en masse*. The presence of numerous untrusted servers in the wild, offering different services, further increases the risks of losing location data.

In this paper, we propose a design for building LBSAs that provides a low-cost, practical, and deployable alternative to existing design all the while providing strong user location privacy. The key insight behind this design is to treat the server as a simple encrypted data store, and move the application functionality to the client smartphones. All the location information shared is encrypted and the lack of plain location information on the storage server improves user privacy. This approach easily works on today's smartphones because the servers running LBSAs today provide their service by running simple operations such as certain database or hashtable lookups, performing simple computations on the location data, and sending the results to be displayed on the clients. For example, in a nearby restaurant review application, the server takes the user location, finds restaurants that are in the vicinity of the user's location, queries the reviews of these restaurants, and sends the results back to the users for display. In our proposed approach, the data storage and lookup operations happen on encrypted data

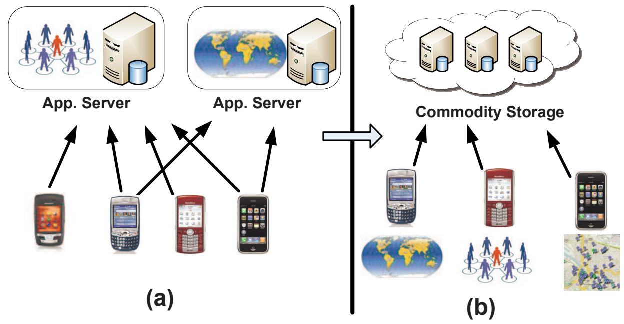

Figure 1: Comparison of the existing and proposed architectures of location-based mobile social applications. (a) In today's architecture, several untrusted third-party servers obtain user location data and offer different services to the users. (b) In our proposed architecture, the functionality is moved to the client devices, and a simple shared storage with a narrow interface is used to build a wide variety of applications.

but still remain on the storage server. The clients receive the encrypted results, decrypt and display the results to the users. The clients only incur an additional cost of decrypting the received content, and performing simple calculations on the decrypted data. By using light-weight cryptographic schemes, we can easily move the functionality to the smartphones and provide services while preserving privacy.

In our approach, the users build an offline social network with their friends by exchanging cryptographic keys and storing these keys on their mobile devices. Whenever users need to exchange certain application-specific data, they do so by always encrypting it with the keys already exchanged with their friends. The applications running on the client devices consume this encrypted data, and deliver the functionality that the users need. Figure 1 provides a comparison of the existing and the proposed architectures.

To understand if this approach is flexible to support a variety of LBSAs, this paper is geared towards identifying the building blocks and the minimal storage server interface necessary in this approach. This is in fact the primary contribution of this paper. We have identified two simple building blocks (called proofs) and a narrow storage server interface. The two proofs are: friendship proofs and transaction proofs. Friendship proofs cryptographically attest the social connection (or friendship) between two users, and similarly, transaction proofs cryptographically attest certain data generated by a user. Using these proofs, any user in the network can verify if a piece of data was generated by a friend, and if so decrypt the data. But no other user other than a friend will be able to see the contents. Finally, the interface exposed by the storage server is narrow enough that we can reason about the privacy guarantees, and yet they are flexible enough to build several LBSAs. As a result, a single storage server can support many different LBSAs.

The rest of the paper is structured as follows. We first describe several motivating applications in Section 2 that highlight different usage scenarios and properties of these LBSAs. In Section 3, we describe our goals, assumptions, and the threat model used in this paper. Next, in Section 4, we present the two key building blocks, and the interface the server exposes. We also present the usage of these building blocks by sketching the implementation of our motivating applications (in Section 2). We then present an initial analysis of the privacy guarantees provided by our approach in Section 5. Finally, we survey the related work in Section 6 and conclude.

2. MOTIVATING APPLICATIONS

Here we describe several motivating LBSAs and look at the operations performed by the applications. These applications are chosen such that they stress on different scenarios under which user location information is used. We highlight these unique properties of the applications, and also investigate the possibility to move the functionality to the client devices. Later, in Section 4.4, we describe how to exactly implement these applications using our proposed building blocks and the storage server interfaces.

2.1 Example Applications

Collaborative Content Downloading. Alice is sitting in a restaurant and wants to download a large file from the web. Alice's mobile phone has both WLAN and WWAN interfaces, but she has Internet connectivity only on WWAN interface (GPRS, 3G, etc.), which takes a long time to download the file. Using mobile social network, however, Alice can easily solve this problem. Alice's mobile phone can talk to other mobile phones in the restaurant, check if any one-hop or two-hop friends are within the WLAN range, and automatically download parts of the file in parallel using the WWAN bandwidth of the friends.

Many such collaborative bandwidth-sharing systems have been proposed before [2], but incentives, trust, and security remain as main problems in such systems. A peer might download illegal content from Alice's mobile phone causing her legal trouble or simply a peer can free-ride on her bandwidth. By selecting trusted friends, several such problems can be averted. This bandwidth-sharing scenario can similarly extends to sharing WiFi access points with friends in need of Internet access, sharing compute cycles with other mobile peers to help them process compute-intensive tasks (decrypting and decoding a real-time video [25]), etc.

Social Recommendations. Several location-based recommendation systems can be built using the information available in the social circle. Suppose that Alice finished her dinner at a restaurant, and wants to recommend this restaurant or certain items, or just wants to convey her opinion on the restaurant to her friends. Alice can convey her opinions by leaving her opinion at an application server. When her friends are around this restaurant, and are deciding on a restaurant for dinner, their mobile phones can obtain Alice's recommendations and help them choose the restaurant. Along similar lines, this scenario may be extended to provide recommendations for other shopping sites and for items within a site.

Local Businesses. It is very common to use mobile devices to gain information about the businesses in a larger neighborhood around a user's location. Here, the user may first want to gain an overview of the businesses in the vicinity, and then later zoom in on a few selected sites and view their friends' opinions about these sites. Similarly, users may want to leave their opinion on a set of sites in the vicinity or the entire neighborhood. The same functionalities are also expected in the scenario where a user is at home and is interested in knowing the businesses in a specific neighborhood.

Location-based Reminders. Users can leave reminders for their friends at interested locations using this applications. Whenever the users that the reminders are intended for are near the location of the reminder, their mobile phone alerts the users about the reminders. For example, users may leave reminders for friends on updated location/time of a party, leave shopping list reminders for friends (family members, roommates, or for themselves) near grocery stores. Similarly, users can keep track of and analyze their exercise/running habits over a period of time by leaving reminders on their running tracks.

Friend Locator. Users interested in keeping track of the current location of their friends or interested in knowing if any of their friends are in the vicinity so that they can meet up, can use this application. Several such applications exist today[1], and users share their location information with the servers to do this. The server computes the distance between the users and alerts people when their friends are within a certain distance.

2.2 Properties of the Example Applications

The applications above highlight different aspects of LBSAs. In the collaborative downloading application, finding trusted devices among the nearby mobiles is critical. In the recommendations application, finding information about a specific site is the focus, but finding information about a larger vicinity of the user is the focus in the local businesses application. The reminders application first downloads data and later fires events when user visits specific sites, thus focusing on dynamically associating events with a user's location. Finally, the friend locator application stresses both on the location and on the social dimensions of the users. As a result, they all provide a good mix of the different types of LBSAs used today.

Operations Invoked by the Example Applications. Also notice that the operations in the above applications that manipulate the location information are very few and are of low cost. For example, the reminder and the friend locator applications just need to compare two location coordinates and check if they are within a certain distance. Similarly, most applications above just need to organize the data associated with locations in the order of the locations by just comparing them. As a result, these operations can be easily moved to the mobile smartphones, and yet maintain the full functionality of the applications.

3. GOALS, SYSTEM AND THREAT MODEL

Here we describe the goals of our design, the scenarios our system targets, and the threat model we consider in this paper.

3.1 System Goals

We aim to achieve the following key goals in our design. (a) Our design should *preserve location privacy* of the users while the users use the applications. To preserve privacy, all data shared in our design is encrypted, and only the user's friends will be able to decrypt a user's (location) data. (b) Our design should be *flexible*, to support a variety of location-based social applications. We demonstrate this flexibility by using our proposed design to sketch the implementation of several different types of location-based applications. (c) We aim to keep the design *simple and practical* to spur its adaption. We leverage widely-used symmetric cryptography to keep the overhead on the mobiles low, and we expose simple hashtable-like interfaces to make using our system easy for programmers. Finally, (d) our design should have *low deployment overhead* and be deployable today. To ease the deployment, we keep our design in line with the storage and computation services provided by existing cloud computing facilities. We describe how our design can easily leverage these cloud services to build scalable LBSAs quickly.

3.2 System Model

Since location-based applications typically involve simple lookup operations combined with limited computation, as mentioned before, we believe that moving the functionality of these applications to the mobile smartphones is practical. For example, iPhone 3G comes with a 412MHz processor and 512MB of RAM, which is sufficient to perform the computation of all common LBSAs. While the responsibility of decrypting and consuming friends' data

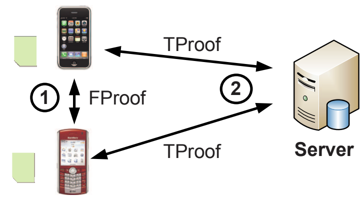

Figure 2: Depiction of proof exchanges. (1) Friends exchange friendship proofs and store them on their devices. (2) Users create and store transaction proofs on the server that are later retrieved by their friends.

is on the smartphones, the server-side is still responsible for storing users' data, backing them up, and serving data to users in an available manner. Since much of these storage tasks are common to several application servers, we observe that they can be easily offloaded to cloud storage providers such as Amazon Simple Storage Service[2] by the application servers potentially running on Amazon EC2[3]. Given that the resource cost on these providers is quite low, and the fact that the applications only need to pay for only the resources they use, this provides a low-cost alternative. Thus we assume for the rest of the paper that the storage is on these cloud providers. Furthermore, the application owners can recover the deployment cost they incur either directly from the LBSA users (fees), or by presenting ads to the users while serving data to them.

3.3 Threat Model

In order to understand the worst case guarantees provided by our system, we assume a strong attacker model in this paper. We assume that the third-party storage server is untrusted. As a result, we investigate the user privacy lost even when the data stored on the server is leaked to an attacker. In practice, however, this is a very strong assumption about the attacker's capability. During our privacy analysis, in Section 5, we also consider an even stronger attacker with power to compromise and monitor the storage server for extended period of time, present our solutions against this attacker and the associated privacy vs. performance tradeoffs.

We assume that the users do not collude with the storage server to break other users' privacy. This assumption fits a social networked system given that users mainly consume and provide data from and to their friends that they trust. In addition, we assume that each user has a user-generated public-private key pair (private key is kept secret, while the public key is shared), and that the users' devices have a localization technology (GPS, for instance) that can tell them their location in terms of longitude-latitude values. The users' mobile devices are trusted, and if a user's mobile device is compromised, we assume that the user detects it and notifies her friends of the device compromise and invalidates the old public key.

4. BUILDING BLOCKS AND THEIR USAGE

Now we present the details of the key building blocks in our design (proofs), the interface the storage server needs to expose to support the types of applications described before, and finally present the usage of these building blocks by sketching the construction of the motivating applications in Section 2.

4.1 Friendship Proof

Friendship proof (FProof) is a cryptographic attestation that a user A gives to her friend B ($\mathrm{FProof}_{A \rightarrow B}$). The user B can store

[1] www.loopt.com

[2] http://aws.amazon.com/s3/

[3] http://aws.amazon.com/ec2/

API Call	Purpose of the Call
putFriendInfo(friendId, value)	Put some data about *friendId*
getFriendInfo(friendId)	Get data about *friendId*
putLocationInfo((x, y), value)	Put data about the location (x, y)
getLocationInfo((x, y))	Get data about the location (x, y)

Table 1: The storage server APIs and their functions.

this attestation on her device and cryptographically prove to any other participant in the system that A is her friend. These proofs are unidirectional, and A should obtain a similar proof from B ($\text{FProof}_{B\rightarrow A}$). The users store all their proofs from their friends in their mobile device and carry it around to benefit from the proofs, as shown in Figure 2.

Given the sensitive nature of the proofs, and untrusted nature of the storage servers, proofs are exchanged via a secure channel between friends. One example is to exchange the proofs when two friends meet each other. Friends' mobile devices can communicate via a wireless interface and exchange these proofs using a cryptographically secure handshake. Secure email is another example channel for exchange.

The proof that an user A gives to an user B is construed by putting together certain pieces of information, and signing the hash of the content with A's private key. The information consists of: A's public key, B's public key, time of issue, and A's symmetric session key. In short, a friendship proof (FProof) is constructed as follows: Let,
Content $=<\text{PubKey}_A$, PubKey_B, SKey_A, timestamp$>$. Then,
$\text{FProof}_{A\rightarrow B} =<$Content,$\text{PrivKey}_A$(Hash(Content))$>$.

The session key (SKey_A) that A puts in the proof is a symmetric key that A uses to encrypt all the data she stores on the storage server (see transaction proofs next). A gives the same session key to all her friends, and hence any data generated by A can be decrypted by all her friends. Using a symmetric key to encrypt all the data in the system makes the system very efficient, even on mobile devices.

The friendship proof described here can be extended to attest friends multiple hops away as follows. Suppose A exchanges proofs with a direct friend C, after exchanging proofs with another friend B. During that time, A can also share the proof $\text{FProof}_{B\rightarrow A}$ with C. C can then use the two proofs, $\text{FProof}_{B\rightarrow A}$ and $\text{FProof}_{A\rightarrow C}$, to prove their two-hop friendship to B and obtain a new proof from B to attest to their two-hop friendship. This way users can leverage the data generated by two-hop friends in the social network, in addition to the data from the direct friends.

4.2 Transaction Proof

Transaction proof cryptographically attests that a piece of information belongs to a user. This proof includes the user's message (msg) for her friends. This message could be simply the user's current location coordinate, or a user's opinion about the dinner she had at a restaurant, or any message that she thinks might be helpful to her friends. The contents of the message is application-dependent, and the message (msg) is encrypted with the user's session key when it is stored on the storage server. Users generate and leave these transaction proofs on the server, as shown in Figure 2, and later any of their friends can access them. A transaction proof (TProof) from a user A is constructed as follows: Let,
Content $=<\text{Sess}_A$(PubKey_A, timestamp, msg)$>$. Then,
$\text{TProof}_A =<$Content, PrivKey_A(Hash(Content))$>$.

4.3 Interfaces Exposed by the Storage Server

Table 1 lists the interfaces exposed by the storage server. We argue that these function calls are flexible to support a wide variety of LBSAs. Next we describe these functions in detail.

FriendInfo get and put Calls. These calls enable users to share application-specific data in encrypted form (as transaction proofs) with their friends. The key *friendId* is the public key of the user that is putting the data, and the puts are authenticated by the storage server. Anyone in the network that knows a user's public key can get the contents from that storage server. However, since only the friends know the public key of a user, and the session key necessary to decrypt the transaction proofs obtained from a *getFriendInfo*, all non-friend users essentially get data that they cannot understand.

LocationInfo get and put Calls. These calls are used for sharing information about a site in the geographic location (x, y) – the longitude-latitude value obtained from a user's GPS. The site could be anything from a restaurant to a store or a school/department. The data is stored as a transaction proof generated by the user. When the user stores the proof, the exact value of (x, y) is also stored within the proof. The server stores the data corresponding to a given (x, y) in a bucket with a key that is approximately close (say within a block radius from the given value). A new bucket is created if one such bucket does not exist. Storing the data at approximated value of (x, y) is useful for several reasons: (a) To account for the localization errors inherent in several positioning systems, and (b) To also enable applications that query for data related to an entire neighborhood (similar to range queries in databases). Similarly, the get call returns all the information stored in the bucket closest to the input (x, y) value. When multiple buckets are close to the user, data from all of them are returned. The server can use several approaches to approximate the coordinates, such as pre-dividing a region into blocks and assigning one bucket per block, etc.

Decrypting the data obtained from the *getLocationInfo* call is different from that of the *getFriendInfo* call. In the version of the call in Table 1, the user that obtains the data will have to try to decrypt all the proofs she obtains with all her friends' keys; valid decryption can be tested by comparing the signature stored in the proof. Section 5 discusses the privacy vs. performance tradeoffs in the design of this call in more detail.

Finally, we implement several optimizations to the calls in the Table 1: (a) Each get call takes in an array of input values to lookup multiple keys at once to avoid multiple round-trip times. (b) A last access time parameter is included to tell the server to avoid sending duplicate proofs and send only the new proofs since the specified time. We do not show these optimizations in the Table to keep the description simple.

Summary. To summarize, friendship proofs are exchanged once between friends via a secure channel. The transaction proofs are stored on the storage server based on the application's needs, and the proofs contain application-specific data encrypted with the user's symmetric key. These applications store the data via one of the *put* calls, and fetch the appropriate data from the store via *get* calls. Applications then decrypt the received data and perform application-specific computation on the data before displaying the results to the users.

4.4 Building Applications with the Proofs

Here we demonstrate the usage of the proofs and the server interfaces by building the motivating apps. we described in Section 2.

In the collaborative downloading application, nearby users only need to use their friendship proofs to check their social proximity. All one-hop friends are known a priori, and any pair of users can easily check if they are two-hop friends by just intersecting their friends list. Once trusted one-hop or two-hop friends are discovered, the mobiles can continue to download files. To preserve the confidentiality of the users' social circles during two-hop friend discovery, privacy preserving matching techniques can be used [8].

In the recommendations and the local businesses applications, the mobiles need to use both the proofs. When a user wants to recommend a site, the user generates a transaction proof and stores it on the server using a *putLocationInfo* call with the site's coordinates. Later, a friend user in the vicinity can do a *getLocationInfo* call with the location's coordinates, decrypt the friends' recommendations out of all the proofs returned, and view the recommendation for the sites in the vicinity that she cares about. Since the *putLocationInfo* call stores the data in a bucket that is approximately close to the input (x, y), all proofs intended for the locations in the same vicinity end up in the same bucket. This enables users (at the intended location, or remotely browsing say from home) to obtain information about the vicinity of the specified location. Finally, the applications on the client devices can make additional calls by providing the longitude-latitude values that are close to the current location of a user to view a larger neighborhood when necessary.

In the reminders application, the user creating the reminder generates a transaction proof and stores it on the server via *putFriendInfo*. Later, her friends download the reminders via *getFriendInfo*. Only the friend user to whom the reminder is intended stores the reminder in the device and the rest of them discard it. The application on the mobile remembers this reminder and generates an alert for the receiver when she is near the reminder location.

Finally, to implement the friend locator, each user periodically stores her current location (in a transaction proof) via a *putFriendInfo* call, and each user's mobile periodically obtains the location of all friends via *getFriendInfo*, computes the distance between the friends on the mobile and plots it on a map.

5. PRIVACY ANALYSIS AND TRADEOFFS

In this section we describe the intuition behind the privacy guarantees provided by our design. We first describe the properties of the server interface, and then look at the impact of compromising several end-points in the system.

5.1 Server Interface Privacy and Tradeoffs

Now we look at the privacy and the performance properties of the interfaces in Table 1, and potential ways to strengthen them.

FriendInfo get and put Calls. These calls provide strong privacy guarantees. Only the friend users with appropriate keys can decrypt the data of a user. Since these calls are targeted for a specific user, they are also efficient – transmit exactly the data that is requested.

LocationInfo get and put Calls. There is a clear performance vs. privacy tradeoff in designing the *getLocationInfo* call. In the *getLocationInfo* interface in Table 1, the user needs to decrypt all the proofs she obtains from the call to see which ones belong to her friends. One potential way to improve the performance is to tag each proof stored via a *putLocationInfo* call with an Id (or public key) of the user that generated the proof. However, this tag clearly leaks location privacy of users in case the server's data leaks.

An approach to achieve both performance and privacy in this call is to tag the proofs with an userId that changes periodically in a known pattern (known only to friends). For example, users can exchange a salt in their friendship proofs and periodically change the tags associated with the new proofs stored via *putLocationInfo* using random numbers generated from this salt. This enables friends to query a specific user's data at a given location using these tags, but the server will not be able to relate the different tags to the same user. Thus, users do not lose privacy and yet the returned results can be filtered by users. Due to the involved nature of this solution, we did not present it in Table 1.

An additional dimension to further filter the returned results is by associating tags related to the locations themselves. Tags such as the site type – restaurants, gas station, etc. We leave details of this for future work.

5.2 Impact of Several Potential Attacks

Compromised Client. A compromised client can leak the location privacy of all her friends. Thus, informing friends of compromised key/device is critical. Revoking keys, unfortunately, is an expensive operation in several crypto systems. As a result, we recommend that the users protect access to their keys via passphrases to reduce the chance of key compromises even if the client device is compromised or lost.

Compromised Third-party Storage Server (Stronger Threat Model). If the data on the server is leaked to an attacker not in the social network, the user privacy is still preserved as no data can be decrypted by the attacker, and hence the attacker cannot associate an user with a location. However, if the attacker were to monitor the incoming connections on the compromised server passively, then the attacker might be able to associate the (x, y) locations in the *putLocationInfo* call with information gleamed from the connection [15] such as the client's browser, SSID, IP address, among others and hence potentially associate a location with an user. However, these attacks can be averted by scrubbing the connection off such information with privacy proxies such as Privoxy. In addition, using anonymous routing [5] can further enhance privacy against such attacks. But unfortunately, using heavy-weight anonymous routing techniques incur significant performance (bandwidth/latency) penalty on the clients. Given that certain reputed storage servers (such as Amazon S3) are less likely to be compromised *and* passively monitored for long time, we believe that heavy-weight techniques are not necessary for reputed storage servers by default.

DoS Attacks on the Server. Like any other shared storage server, our design is also vulnerable to DoS and pollution attacks. But existing mechanisms to combat these attacks also work for us.

6. RELATED WORK

Mechanisms to Preserve Location Privacy. A well-known mechanism used to improve user location privacy is spacial and temporal cloaking [12], wherein the location and time sent to the server is approximated instead of sending the real value. The intuition is that this prevents accurate identification of the location of the users, 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 [10, 14, 16] that can still break user privacy.

Pseudonyms and silent times [15] are other solutions proposed to improve location privacy. They argue for changing the device identifier frequently, and not transmitting signals for long periods at regular intervals, thus hurting functionality and disconnecting users from applications. The primary problem with majority of the solutions proposed to date is that they build on the model where the third-party servers are entrusted with location data in plain text. Compromising such a server easily leads to large-scale privacy leaks.

Novel Applications and Location Privacy. AnonySense [5] recently proposed using an anonymous routing overlay to preserve user location privacy. While this system supported a limited interface only to upload non-real-time data to the servers, several mechanisms used in this system are complementary to our work here.

Several researchers have looked at building specific mobile social applications while preserving user privacy. SmokeScreen [6] is designed to share presence information only with trusted users, and missed connections [19] looks at connecting users that shared a physical location without compromising their privacy at the server. We share the privacy goal with these papers, but we are attempting to build a framework for supporting a variety of applications.

The friendship proof construction is similar to that of location proofs proposed in [22], and the social attestations proposed in [28]. The primary difference is that social attestations are used for social access control of data in Lockr [28], and location proofs [22] are mainly focused on enabling novel class of applications.

Encryption to Improve Privacy. Similar to our approach, Persona [3] encrypts all data shared among users in a social network. While we focus on location privacy and using symmetric crypto for efficiency on mobile devices, Persona focuses on online social networks and on providing fine-grained privacy controls to users. Hence Persona uses attribute-based encryption which are significantly heavy-weight for mobile devices.

Decentralized Social Networks. VIS [4] argued for a decentralized approach to build mobile social applications using compute clouds. Similarly, PrPl [24] built a decentralized social network where users store their data in personal Butlers. PrPl uses a novel language called SociaLite to execute queries over this distributed network of Butlers. This language ensures that user access control policies are honored, and hence preserves user privacy. In these proposals, however, the cost of owning and operating this decentralized network is an hindrance to widespread deployment.

Social and Policy Changes. A recent paper [26] argued for social and policy changes in addition to technical changes to provide user privacy in mobile environments. It argued for a system architecture that includes strict access control policies, data visualization tools for users to understand the full impact of data shared on privacy, and allow for users to change the access control policies when they wish. Incorporating our mechanisms into this architecture would further improve user privacy.

7. FUTURE WORK AND CONCLUSIONS

We argued in this paper that location-based mobile social applications (LBSAs) need to take an approach where the application intelligence is moved to the clients and the servers simply act as rendezvous points to share encrypted data. We provide evidence to show that this approach is flexible enough to implement several widely-used applications, and yet preserves location privacy of the users – a property lacking in today's systems.

Our current design is tailored for location-based social applications. We can further extend and apply our design to other contexts, including non-social applications. This would improve the applicability of our work significantly: (i) Several more applications can adapt our approach, and (ii) Even users of social applications can benefit by using data from non-friends, which is especially useful for users with only a handful of friends. The key challenge in extending our approach to non-social applications is encryption key management. Key management is currently done by the users based on their offline social network, which is lacking in non-social applications. Thus we need to develop novel mechanisms for users to securely discover the keys used to encrypt the data on the server, without revealing the key to the server itself.

In addition to resolving the above-mentioned challenge, we are also working on implementing and deploying the applications described in this paper on mobile devices. We intend to report on our experience regarding the performance, energy-efficiency, overhead, and usability of our proposed approach in the future.

8. REFERENCES

[1] Mobile LBS on the move, Oct. 2008. `http://www.emarketer.com/Article.aspx?R=1006609`.

[2] Ananthanarayanan, G., et al. Combine: leveraging the power of wireless peers through collaborative downloading. In *Proc. of MobiSys* (2007).

[3] Baden, R., Bender, A., Spring, N., Bhattacharjee, B., and Starin, D. Persona: An online social network with user defined privacy. In *Proc. of SIGCOMM* (2009).

[4] Cáceres, R., et al. Virtual individual servers as privacy-preserving proxies for mobile devices. In *Proc. of MobiHeld* (2009).

[5] Cornelius, C., et al. AnonySense: Privacy-aware people-centric sensing. In *Proc. of MobiSys* (2008).

[6] Cox, L. P., Dalton, A., and Marupadi, V. Smokescreen: Flexible privacy controls for presence-sharing. In *Proc. of MobiSys* (2007).

[7] DailyNews. How cell phone helped cops nail key murder suspect secret 'pings' that gave bouncer away, Mar. 2006.

[8] Freedman, M. J., and Nicolosi, A. Efficient private techniques for verifying social proximity. In *IPTPS* (2007).

[9] Froehlich, J., and Krumm, J. Route prediction from trip observations. *Society of Automotive Engineers* (2008).

[10] Golle, P., and Partridge, K. On the anonymity of home/work location pairs. In *Proc. of Pervasive* (2009).

[11] Grace, F. Stalker victims should check for gps, Feb. 2003.

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

[13] Hendrickson, M. The state of location-based social networking, Sept. 2008.

[14] Hoh, B., et al. Enhancing security and privacy in traffic-monitoring systems. In *IEEE Pervasive Computing Magazine* (2006).

[15] Jiang, T., Wang, H. J., and Hu, Y.-C. Preserving location privacy in wireless lans. In *Proc. of MobiSys* (2007).

[16] Krumm, J. Inference attacks on location tracks. In *Proc. of Pervasive* (2007).

[17] Krumm, J. A survey of computational location privacy. *Personal and Ubiquitous Computing* (2008).

[18] Li, K. A., et al. Peopletones: a system for the detection and notification of buddy proximity on mobile phones. In *Proc. of MobiSys* (2008).

[19] Manweiler, J., and others, R. We saw each other on the subway: Secure, anonymous proximity-based missed connections. In *Proc. of HotMobile* (2009).

[20] Miluzzo, E., et al. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proc of SenSys* (2008).

[21] Motani, M., Srinivasan, V., and Nuggehalli, P. S. Peoplenet: engineering a wireless virtual social network. In *Proc. of MobiCom* (2005).

[22] Saroiu, S., and Wolman, A. Enabling new mobile applications with location proofs. In *Hotmobile* (2009).

[23] Schilit, B., Hong, J., and Gruteser, M. Wireless location privacy protection.

[24] Seong, S.-W., et al. The architecture and implementation of a decentralized social networking platform. Tech. rep., Stanford, October 2009.

[25] Shen, G., Li, Y., and Zhang, Y. Mobius: enable together-viewing video experience across two mobile devices. In *Proc. of MobiSys* (2007).

[26] Shilton, K., et al. Designing the personal data stream: Enabling participatory privacy in mobile personal sensing. In *In TPRC* (2009).

[27] Sohn, T., et al. Place-its: A study of location-based reminders on mobile phones. In *Proc. of Ubicomp* (2005).

[28] Tootoonchian, A., Gollu, K. K., Saroiu, S., Ganjali, Y., and Wolman, A. Lockr: social access control for web 2.0. In *Proceedings of WOSN* (2008).