Longitude: a Privacy-preserving Location Sharing Protocol for Mobile Applications*

Changyu Dong, Naranker Dulay

Department of Computing
Imperial College London
{changyu.dong,n.dulay}@imperial.ac.uk

Abstract. Location sharing services are becoming increasingly popular. Although many location sharing services allow users to set up privacy policies to control who can access their location, the use made by service providers remains a source of concern. Ideally, location sharing providers and middleware should not be able to access users' location data without their consent. In this paper, we propose a new location sharing protocol called Longitude that eases privacy concerns by making it possible to share a user's location data blindly and allowing the user to control who can access her location, when and to what degree of precision. The underlying cryptographic algorithms are designed for GPS-enabled mobile phones. We describe and evaluate our implementation for the Nexus One Android mobile phone.

1 Introduction

Location sharing is an increasingly popular function of social-networking services, allowing users to share their location with family and friends. Examples include Google Latitude [1], Yahoo Fire Eagle [2], and Loopt [3]. Perhaps the biggest user concern about location sharing services is privacy. Many services allow the users to control who will have access to their location data, over what period of time, and to what degree of precision. However, for many users, the service providers are also a source of concern. Will not the location sharing service use location data to the detriment of the user?

Users' location data is normally saved by the service provider. Unfortunately, this allows providers to track, profile and target users [4, 5] as well as aggregrate the data and sell it to others. The typical approach to informing users is to provide a lengthy webpage that states what the service provider may do with the data. The webpage is usually written in a sufficiently obfuscated way to ensure that few users will bother reading it, and often to hide the fact that providers want to give themselves a high degree of access to the data. A related, but important concern, is that the service provider may be the target of network intrusions and untrustworthy insiders, as well as requests from law-enforcement agencies [6].

This paper describes a protocol called Longitude for location sharing that uses cryptography to limit service provider access to location data. It is aimed at providers on the

* This work was supported by UK EPSRC research grant EP/F023294/1 - PRiMMA: Privacy Rights Management for Mobile Applications.

Internet or middleware to provide location sharing blindly without the hassle of compliance to data protection and location data requests. It ensures that users are able to share their location but are not tracked. Note that the protocol is not a replacement for traditional location-services like Google maps that translate locations into maps.

Naively, a user (Alice) could encrypt her location before sending it to the location sharing provider (Luke), effectively protecting it from Luke or other adversaries. Alice would have to securely disseminate the key to her friends (Bob and Carol) and revoke it if she wanted to prevent access to any friend or if the key was disclosed. Rather than a common shared key, Alice could establish pair-wise secret keys with each of her friends or use asymmetric keys, both requiring a great deal more additional storage, computation and communication overheads. A more flexible approach is needed, particularly for resource-constrained mobile devices.

Longitude has the following characteristics:

1. Privacy preserving. Longitude enables location-sharing providers or middleware to disseminate user location data blindly. The data is specially encrypted. Alice can control which of her friends can see her location, at what times and to what degree of precision.
2. Simple key management: Alice only needs to keep her own key on her mobile device. She can remove any of her friends at any time without affecting other friends. The revocation process can be done by Alice without requiring any interactions with her friends.
3. Lightweight cryptography. Most of the computationally intensive cryptographic operations in Longitude are done by the service provider, not on the mobile device. Computation and battery life for mobile devices can be optimised further by precomputing cryptographic material when the mobile device is connected to a power source.
4. Constant communication overhead. Longitude's communication costs do not increase with the number of friends (receivers). No matter how many friends a user has, each piece of location data is encrypted and sent only once. Therefore, the overhead of data communication is minimised.

The paper is organised as follows: in Section 2, we summarise the related work; in Section 3, we discuss the system and security model as well as the initial assumptions; in Section 4, we present Longitude, how to fine control user privacy and issues related to user revocation; in Section 5, we explain the underlying cryptographic techniques; in Section 6, we describe and evaluate a prototype implementation of Longitude for Android phones; in Section 7, we conclude the paper and discuss our future plans.

2 Related Work

Location sharing services have attracted a lot of attention from industry [1–3], and the development of GPS-enabled mobile phones makes it easy to sense and share user location. According to [7], these services can be categorised into two types: (1) purpose-driven, in which the requester has a specific need for the users location, e.g. coordinating meetings, arranging transportation, sending reminders, and (2) social, in which location

information is shared simply because it is interesting or fun to do so. However, users are concerned about their privacy and according to [8], existing industry guidelines and implementations do not adequately address these concerns .

Previous research on location privacy has focused on anonymisation. For example, in [9], the authors describe a middleware system which delays and reorders messages from users within a mix zone to confuse an observer. In [10], a mechanism called cloaking is proposed that conceals a user within a group of k people. To achieve this, the accuracy of the disclosed location is reduced so that a user is indistinguishable from at least $k - 1$ other users. In [11], k-anonymity is achieved by an ad-hoc network formed by the user and surrounding neighbours, while [12] shows how to achieve k-anonymity in a distributed environment where there are multiple non-colluding servers. Anonymisation has a fundamental difference with location sharing. The goal of anonymisation is to prevent others from relating a location to a user; on the other hand, the goal of location sharing is to let authorised users know where a user is. Therefore, anonymisation is not directly applicable here.

Most existing location sharing services do offer the users some form of controls over their privacy. In [8], the authors examine 89 location sharing services and the most widely adopted privacy controls are white list, being invisible, blacklist, group-based permission and providing less detailed location. Several research projects in this area have tried to provide more expressive and effective policy-based privacy controls. For example, Locaccino [13] allows users to specify more fine grained policies based on temporal and spatial restrictions. The pawS system [14] allows a user to use P3P policies to define their location privacy settings and negotiate with the location service provider. The main drawback in all such approaches is that the users must trust the provider, its privileged employees and the security of the infrastructure. The user's privacy will also be compromised if the service provider is required to disclose the data to a law-enforcement agency.

In a location sharing services, the provider usually acts as a broker to disseminate the location information to the authorised receivers. In most of the cases the provider does not need to know the data content in order to provide this service. In [15] a system for sharing user location is described which provides protection from the provider. Users use pairwise symmetric key encryption or asymmetric key encryption to prevent the provider from learning their location. However, the user needs to store multiple keys. Moreover the user has to send multiple copies of the same data, each encrypted under a different key in order to let all her friends be able to get her location. The overheads of key management, computation and communication increase linearly with the number of friends.

Some work [16–19] has been done dealing with the problem of preserving privacy in proximity services. Proximity service is a sub-type of location sharing service which notifies and displays a friend's location if the friend is nearby. While in Longitude we consider the more general location sharing where a user can see a friend's location no matter the friend is near or far away from the user.

3 Models and Assumptions

3.1 Systems model

The Longitude protocol has the following parties: the location-sharing service provider and the set of users registered with the provider. We assume that each user has a GPS-enabled mobile phone that can sense the user's current location and send it to the provider. The provider stores the location and along with some user configuration data. Users define which other users are authorised to receive their location. Authorised receivers can be removed at any time by the user. Users can also define the precision of the location that will be seen by a particular receiver, e.g. accurate to 1km, 5km, 10km, 100km.

3.2 Security Model

We consider the service-provider to be honest-but-curious. That is, the service provider will follow the protocol correctly, but will try to find out as much secret information as possible. To simplify the presentation in the paper we assume that there are mechanisms in place which ensure integrity and availability of the stored data. We also assume that there is a proper authentication mechanism which allows the user to identify the service provider and their friends and vice versa. In addition, we assume that each user securely protects their cryptographic key on their mobile device. Since location data will be transmitted through public networks and wireless networks, we assume that it is possible that an unauthorised user can intercept the data.

4 Longitude Protocol

4.1 Overview

We first describe how the protocol works in general. The protocol is depicted in Figure 1. In the figure we only show two users, Alice and Bob.

The design of Longitude is based on proxy re-encryption [20]. In a proxy re-encryption scheme, a ciphertext encrypted by one key can be transformed by a proxy function into the corresponding ciphertext for another key without revealing any information about the keys and the plaintext. Applications of proxy re-encryption include access control systems [21] and searchable data encryption [22]. The details of the proxy re-encryption scheme used in Longitude will be presented in Section 5.

To share her location with Bob, Alice and Bob must first register with the location service provider (Luke). During registration, Alice and Bob also obtain public cryptographic parameters and generate a public/private key pair locally on their mobile devices. After registration, Bob can send a request to Alice asking her to allow him to see her location. The request can be done out of band without involving Luke. In the request, Bob provides a copy of his public key. If Alice agrees, she computes a re-encryption key using Bob's public key and her own private key (explained in detail in section 5). She also decides how accurate the location should be for Bob and generates a corresponding precision mask (explained in section 4.2). The re-encryption key

Fig. 1. Overview of Longitude Protocol

and the precision mask are sent to Luke, and act as an authorisation policy that allows Bob to retrieve Alice's location. Alice can now send encrypted location data to Luke. Bob's public key can also be discarded by Alice. Luke only stores a user's most recent location. The previous location is overwritten by a newly received location. When Bob wants to know where Alice is, he sends a request to Luke, who retrieves Alice's last encrypted location, applies the re-encryption key and policies defined by Alice then sends it to Bob. Bob can then decrypt the location received from Luke and process it as needed, e.g. to display Alice's location on a map.

4.2 Location Encryption and Location Granularity

Proxy re-encryption, though very efficient, is still too time-consuming to encrypt large volumes of data. To overcome this, in Longitude the actual data is encrypted by a more efficient hybrid encryption scheme, where a secure symmetric stream cipher is chosen to encrypt the location data under a random key and the random key is then encrypted using the proxy re-encryption scheme. The stream cipher also allows Luke to modify part of the ciphertext without rendering it undecryptable. In particular we can use this to allow Alice to define the granularity that her location is seen by different friends.

A location consists of a latitude and longitude. Both parts are represented in the format of decimal degrees. Obviously, a pair (51.49875, -0.17917) gives more accurate information about Alice's location than just (51.4, -0.1). In Longitude, we use this to allow Alice to define precision masks for each friend (see Figure 2). Before encryption, locations are encoded as a pair of fixed-length ASCII strings. Each String has 11 characters in the format of "siiiffffff" where "s" is for the sign, "iii" is for the integral part and "ffffff" is for the fractional part. For example 51.49875 is encoded as "+0514987500". When using a stream cipher to encrypt, the stream cipher generates a stream of random bits. The location strings are also converted into bits and XORed with the random bit stream. Precision masks govern how many digits will be released to friends. Each precision mask is a pair of integers from 0 to 11. Luke simply truncates the encrypted location to the length specified by the precision mask before returning it to a friend. The truncated encrypted location information can still be decrypted after that because the decryption is another XOR. The benefits of using precision masks are two fold: (1) Alice does not have to encrypt the same location at different precision levels for different friends (2) applying the precision mask does not require Luke to first decrypt the data, so Luke can do it blindly. An example of using precision masks is shown in Figure 2. In the example, point 1 (+051.4987500, -000.1791700) is Alice's actual location , while point 2 (+051.49, -000.17) and point 3 (+051.4, -000.1) are the displayed locations for two different precision masks (6,6) and (5,5), i.e. what would be sent to two different friends.

Alice can also specify time-based policies to further control her privacy. An example of such a policy could be "My co-workers should not see my location during weekends". The policies are specified by Alice as constraints and uploaded to Luke. The policies

Fig. 2. Applying precision mask to encrypted location

do not need to be encrypted because they contain no location data (although they might contain other sensitive information). Luke is responsible for checking and enforcing these policies when Alice's location is requested by her co-workers.

4.3 Friend Revocation

If Alice wishes, she can revoke Bob from accessing her location. In Longitude, revocation can be accomplished in two different ways.

The first is called weak revocation. In this case, Alice simply sends a request to Luke asking that Bob should not receive her location any more. Luke then removes the corresponding re-encryption key. Since Alice's key pair and Bob's key pair are generated independently, it is easy to prove that after the re-encryption key has been removed by Luke, Bob will not be able to decrypt any of subsequent location updates from Alice.

Weak revocation has low overhead and is secure if Luke and Bob do not collude. However, if Luke colludes with Bob and does not remove the re-encryption key, Bob will still be able to track Alice. To prevent collusion, Alice can use strong revocation by updating her keys. Updating only changes two components in her keys and leaves the other parts unchanged. Alice also updates the re-encryption keys for all friends except Bob. After Alice has done this, Bob's re-encryption key will not be able to decrypt future locations encrypted using Alice's new public key. Note this process does not require Alice to interact with any of her friends. The update can be done by Alice herself using existing information. If Alice is authorised to receive location updates from her friends, those friends do not need to be involved either. The re-encryption keys they generated for Alice are still valid because these keys are generated using an unchanged component in Alice's public key. The details of the key update algorithm can be found in Section 5.

5 Proxy Re-encryption

Fig. 3. The proxy re-encryption scheme

The proxy re-encryption scheme used in Longitude is adapted from [21]. The scheme has many desirable features, for example, the proxy function is unidirectional and the user only needs to store her own key. We extended the scheme with a new key structure, support for user revocation and redesigned re-encryption and decryption functions. Our scheme is also provably secure under the conventional Decisional Bilinear Diffie-Hellman (DBDH) assumption [23], while the security of the original scheme is based on

a special extension of the DBDH assumption. The proxy re-encryption scheme consists of 8 functions:

- The **Setup** funcition needs to be run once by the location service provider to initialise the service. It generates public parameters which will be used from then on. The provider does not need to keep any secret information after running this function.
- The **Keygen** function is run on the user's mobile device when the user registers. It also only needs to be run once.
- The **Encrypt** function is run on the user's device to encrypt the location data which is going to be sent to the provider.
- The **RekeyGen** function is run on the user's device to generate the re-encryption key for an authorised friend.
- The **ReEncrypt** function is run by the provider to transform location ciphertexts sent to friends.
- The **Decrypt** function is run on a friend's device to decrypt the locations received from the provider.
- The **KeyUpdate** function is run to update the user's key pair during strong revocation.
- The **ReKeyUpdate** function is run to update a re-encryption key during strong revocation.

5.1 Cryptographic Scheme

Our scheme is constructed on top of bilinear pairings. We briefly review bilinear pairings. We use the following notation:

- G_1 and G_2 are two cyclic groups of prime order q.
- g is a generator of G_1.
- e is a bilinear pairing $e : G_1 \times G_1 \rightarrow G_2$ which has the following properties:
  1. Bilinearity: for all $u, v \in G_1, a, b \in \mathbb{Z}_q$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
  2. Non-degeneracy: $e(g, g) \neq 1$.
  3. Computable: There exists an efficient algorithm to compute $e(u, v)$ for all $u, v \in G_1$.

We now describe the proxy re-encryption algorithm in detail. The encryption/decryption scheme is shown in Figure 3.

- **Setup**(k): Given the security parameter k, choose two groups G_1, G_2 of prime order q and a bilinear pairing $e : G_1 \times G_1 \rightarrow G_2$. Then choose a random generator $g \in G_1$. Finally set the public parameter $param = (G_1, G_2, e, g)$ for the system.
- **Keygen**($param$): User i chooses x_i, y_i, z_i uniformly randomly from $\mathbb{Z}_q$ and computes $h_{i1} = g^{y_i}, h_{i2} = g^{z_i}, Z_i = e(g^{x_i}, g^{z_i})$. The user's public key is $pk_i = (h_{i1}, h_{i2}, Z_i)$, the user's private key is $sk_i = (x_i, y_i)$.
- **Encrypt**($m, pk_i, param$): To encrypt a message (e.g. location) with the user i's public key, choose r_i uniformly randomly from $\mathbb{Z}_q$, and compute ciphertext $C = (g^{r_i}, m \cdot Z_i^{r_i})$.

– **RekeyGen**($sk_a, pk_b, param$): To generate a key which can transform a ciphertext encrypted with a user a's public key to a ciphertext which can be decrypted using another user b's private key, the user a chooses n uniformly randomly from $\mathbb{Z}_q$, and computes $rk_{a\to b} = (h_{b1}^n, g^n h_{a2}^{-x_a})$.
– **ReEncrypt**($C_a, rk_{a\to b}, param$): To transform a ciphertext encrypted with a's public key into a ciphertext which can be decrypted using b's private key, the provider computes:

$$\begin{aligned} c_1 &= e(g^{r_a}, h_{b1}^n), \\ c_2 &= m \cdot Z_a^{r_a} \cdot e(g^{r_a}, g^n h_{a2}^{-x_a}) \\ &= m \cdot e(g^{x_a}, g^{z_a})^{r_a} \cdot e(g^{r_a}, g^n) \cdot e(g^{r_a}, g^{-x_a z_a}) \\ &= m \cdot e(g,g)^{x_a z_a r_a} \cdot e(g^{r_a}, g^n) \cdot e(g,g)^{-x_a z_a r_a} \\ &= m \cdot e(g^{r_a}, g^n) \end{aligned}$$

The new ciphertext $C_b = (c_1, c_2)$.
– **Decrypt**(sk_i, C_i): The re-encrypted ciphtertext is decrypted as follows: $c_2 \cdot c_1^{-\frac{1}{y_i}} = m \cdot e(g^{r_i}, g^n) \cdot e(g^{r_i}, h_{i1}^n)^{-\frac{1}{y_i}} = m \cdot e(g,g)^{r_i n} \cdot e(g,g)^{-r_i y_i n \frac{1}{y_i}} = m$
– **KeyUpdate**($sk_i, pk_i, param$): The user only needs to change two components in the pair: the secret key will be changed from (x_a, y_a) to (x'_a, y_a) where x'_a is a random integer from Z_q and the public key will be changed from (h_{a1}, h_{a2}, Z_a) to $(h_{a1}, h_{a2}, Z_a^{\frac{x'_a}{x_a}})$.
– **ReKeyUpdate**($rk_{a\to b}, param$): For a re-encryption key $(h_{b1}^n, g^n h_{a2}^{-x_a})$, the user raises both of the values to the power of $\frac{x'_a}{x_a}$, where x'_a is the random integer generated in the KeyUpdate function. The new re-encryption key can be effectively written as $(h_{b1}^{n'}, g^{n'} h_{a2}^{-x'_a})$ where $n' = n \cdot \frac{x'_a}{x_a}$.

5.2 Security Against an Unauthorised User

Our scheme is semantically secure against an unauthorised user. The notion of secure against an unauthorised user is captured through the following game.

Game$_1$: The adversary $\mathcal{A}$ is an unauthorised user:
Game Setup: The challenger runs **Setup**(k) to generate the public parameter (G_1, G_2, e, g) given the security parameter k. It also uses **Keygen**($param$) to generate an arbitrary number of public/private key pairs $pk_i/sk_i = (h_{i1}, h_{i2}, Z_i)/(x_i, y_i)$. Then the challenger randomly choose a pair pk_a/sk_a.

The public parameter and all the public keys are given to $\mathcal{A}$.
Phase 1: $\mathcal{A}$ is given oracle access to **Encrypt**($\cdot, pk_i, param$). The adversary outputs a pair of message m_0, m_1 of the same length.
Challenge: The challenger randomly chooses $b \leftarrow \{0,1\}$ and then a ciphertext $C =$**Encrypt**($m_b, pk_a, param$) is returned to $\mathcal{A}$.
Phase 2: $\mathcal{A}$ continues to have oracle access to **Encrypt**($\cdot, pk_i, param$).
Guess: $\mathcal{A}$ outputs a bit b' and wins the game if $b' = b$.

Theorem 1. *The proxy encryption scheme is semantically secure against an unauthorised user, i.e. for all PPT adversaries $\mathcal{A}$, there exists a negligible function $negl$ such that:*

$$Pr[Succ_{\mathcal{A}}^{game_1}(k)] \leq \frac{1}{2} + negl(k)$$

The proof of Theorem 1 relies on the Decisional Bilinear Diffie-Hellman (DBDH) assumption [23] which is stated as follows: given $g, g^{\alpha}, g^{\beta}, g^{\gamma} \in G_1$ and $r \in G_2$, every probabilistic polynomial time adversary $\mathcal{A}$ has only a negligible probability in deciding whether $r = e(g,g)^{\alpha\beta\gamma}$ or not, i.e.:

$$Pr[\mathcal{A}(g, g^{\alpha}, g^{\beta}, g^{\gamma}, e(g,g)^{\alpha\beta\gamma}) = 1] - Pr[\mathcal{A}(g, g^{\alpha}, g^{\beta}, g^{\gamma}, e(g,g)^{\delta}) = 1] \leq negl(k)$$

Proof. Let's consider the following PPT adversary $\mathcal{A}'$ who attempts to solve the DBDH problem using $\mathcal{A}$ as a sub-routine. $\mathcal{A}'$ is given a tuple $(G_1, G_2, e, g, g^{\alpha}, g^{\beta}, g^{\gamma}, r)$ such that $g, g^{\alpha}, g^{\beta}, g^{\gamma} \in G_1$ and $r \in G_2$. $\mathcal{A}'$ does the following:

Game Setup: $\mathcal{A}'$ sets $param = (G_1, G_2, e, g)$. $\mathcal{A}'$ also chooses $y_a \in Z_q$ randomly and sets $pk_a = (g^{y_a}, g^{\gamma}, e(g^{\alpha}, g^{\gamma}))$. $\mathcal{A}'$ then chooses an arbitrary number of random integers $(x_i, y_i, z_i) \in Z_q$ and computes $pk_i = (g^{y_i}, g^{z_i}, e(g^{x_i}, g^{z_i}))$. The public parameters and all the public keys are given to $\mathcal{A}$.
Phase 1: Whenever $\mathcal{A}$ requires oracle access to **Encryt**$(\cdot, pk_a,\ param)$, $\mathcal{A}'$ chooses a random integer $r_a \in Z_q$ and encrypts the message using the corresponding public key as $(g^{ra}, m \cdot e(g^{\alpha}, g^{\gamma})^{r_a})$. At the end of phase 1, $\mathcal{A}$ outputs two messages m_0, m_1 of the same length.
Challenge: $\mathcal{A}'$ randomly chooses $b \leftarrow \{0,1\}$ and sends $(g^{\beta}, m \cdot r)$.
Phase 2: Whenever $\mathcal{A}$ requires oracle access to **Encryt**$(\cdot, pk_a,\ param)$, $\mathcal{A}'$ chooses a random integer $r_a \in Z_q$ and encrypts the message using the corresponding public key as $(g^{ra}, m \cdot e(g^{\alpha}, g^{\gamma})^{r_a})$.
Guess: $\mathcal{A}$ outputs a bit b'.

If $b' = b$, $\mathcal{A}'$ outputs 1, otherwise outputs 0. There are two cases:

Case 1: $r = e(g,g)^{\delta}$ for some random δ. In this case the probability of $b' = b$ is exactly $\frac{1}{2}$. So we have $Pr[\mathcal{A}'(g, g^{\alpha}, g^{\beta}, g^{\gamma}, e(g,g)^{\delta}) = 1] = \frac{1}{2}$.

Case 2: $r = e(g,g)^{\alpha\beta\gamma}$. In this case, $(g^{\beta}, m \cdot r)$ is a proper ciphertext for $\mathcal{A}$ and the probability of $b' = b$ is the same as $Succ_{\mathcal{A}}^{game_1}(k)$. So we have $Pr[\mathcal{A}'(g, g^{\alpha}, g^{\beta}, g^{\gamma}, e(g,g)^{\alpha\beta\gamma}) = 1] = Succ_{\mathcal{A}}^{game_1}(k)$.

Since the DBDH problem is hard, we have

$$Pr[\mathcal{A}'(g, g^{\alpha}, g^{\beta}, g^{\gamma}, e(g,g)^{\alpha\beta\gamma}) = 1] - Pr[\mathcal{A}'(g, g^{\alpha}, g^{\beta}, g^{\gamma}, e(g,g)^{\delta}) = 1] \leq negl(k)$$

After substitution, the above in-equation becomes $Succ_{\mathcal{A}}^{game_1}(k) - \frac{1}{2} \leq negl(k)$ and hence $Succ_{\mathcal{A}}^{game_1}(k) \leq \frac{1}{2} + negl(k)$.

5.3 Security Against the Proxy

Our scheme is also semantically secure against the proxy (provider). This notion is captured by **Game**$_2$ which differs from **Game**$_1$ only in the game setup step. In **Game**$_2$, the challenger also gives a set of re-encryption keys to the adversary.

Theorem 2. *The proxy encryption scheme is semantically secure against the proxy i.e. for all PPT adversaries $\mathcal{A}$, there exists a negligible function $negl$ such that:*

$$Pr[Succ_{\mathcal{A}}^{game_2}(k)] \leq \frac{1}{2} + negl(k)$$

Proof. The proof here is very similar to the proof of Theorem 1 except that in the **Game Setup** step, $\mathcal{A}'$ needs to generate a set of proxy keys and send them to $\mathcal{A}$. To generate a re-encrypt key $rk_{a\to b}$, $\mathcal{A}'$ chooses n, n' randomly from Z_q and set $rk_{a\to b} = (h_{b1}^n, g^{n'})$. Note that this re-encryption key is not correctly formed, but it has the same distribution as a correctly formed re-encryption key. Therefore $\mathcal{A}$ cannot distinguish this simulation from a real-world attack in which all values have the correct form. In other words, the view of $\mathcal{A}$ is indistinguishable from a real-world attack. The rest of the proof is the same as the previous proof.

6 Implementation and Evaluation

6.1 Implementation

Fig. 4. The architecture of the Prototype Application

We implemented Longitude in Java for testing and evaluation purposes. The architecture of a small application and location sharing service using Longitude is as shown in Figure 4.

The client side has three components: (1) a user interface which provides the basic functionality for displaying user locations visually and performing management and configuration tasks; (2) a location update service which runs in the background to sense user location, encrypt it and send it to the server on schedule; (3) a pre-computing service which runs in the background only when external power has been connected to the device (see Section 6.2). The client side runs on the Android platform [24].

The server side has persistent data storage for location data and user configurations including re-encryption keys, precision masks and time-based policies. A daemon runs on the server and receives updates and request from clients. It can run on any system with Java 1.1 or above.

We did not find any cryptographic library in Java which supports bilinear pairing, so we implemented our own pairing library[1]. The algorithm implemented for pairing computation was the BKLS algorithm in Jabobian coordinates as described in [25]. We built all the underlying algebraic structures such as finite fields and elliptic curves using the BigInteger class in standard Java. We used the AES implementation provided by SunJCE.

The security parameters are taken from [26]. Namely, G_1 is an order-q subgroup of a non-supersingular elliptic curve over a finite field F_p, where q is a 160-bit prime and

[1] Jpair: `http://sourceforge.net/projects/jpair/`

p is a 512-bit prime. G_2 is a subgroup of the finite field F_{p^2}. The overall security of this setting is roughly equivalent to 1024-bit RSA. We used AES-OFB [27, 28] as the stream cipher. A key length of 128-bit was used.

6.2 Optimisation

Performance is an important issue for mobile applications. To enable location sharing service, users need to run a client-side application using the protocol on their mobile device. The application typically needs to be run in the background to collect and update location data periodically. If the application consumes too much resource, it will slow down the foreground applications and will drain the battery.

Comparing to the location sharing services, the major performance overhead using Longitude comes from the cryptographic operations. To minimise the performance impact, Longitude is designed to distribute these operations between the mobile device and the server. To encrypt the location, 2 operations are needed on the mobile device: encryption of the location using the stream cipher and encryption of the random key using the **Encrypt** function of the proxy encryption scheme. To decrypt location ciphertext, 2 operations are needed on the mobile device: decryption of the random key using the **Decrypt** function and decryption of the location ciphertext using the stream cipher. Stream ciphers are usually very efficient [29] and their impact on performance is negligible. Although the proxy re-encryption scheme requires bilinear pairing operations which are computationally expensive, these operations are done on the server. The **Encrypt** and **Decrypt** functions which are performed on the user's mobile device require only group exponentiations and group multiplications. More precisely, the **Encrypt** function requires only 1 exponentiation in group G_1, 1 exponentiation and 1 multiplication in group G_2. The **Decrypt** function requires only 1 exponentiation and 1 multiplication in group G_2.

The **Encrypt** function is optimised further using the offline/online cryptography paradigm [30, 31]. The ciphertext produced by the function is in the form of $(g^{r_a}, m \cdot Z_a^{r_a})$ where m is the location plaintext, g and Z_a are components in the public key and r_a is a random integer. The function can be naturally divided into two phases: a pre-computing (offline) phase and a final-encryption (online) phase. The pre-computing phase can be performed when the mobile device is being charged and no foreground application is running. In this phase multiple $(g^{r_a}, Z_a^{r_a})$ pairs are computed and stored. In the final-encryption phase when the application needs to send a location update to the server, a pair which is pre-computed in the pre-computing phase is retrieved from local data storage and a multiplication is performed to assemble the final ciphertext $(g^{r_a}, m \cdot Z_a^{r_a})$. The used pair is then erased from the device. In this way we can significantly improve the performance and reduce the energy consumpition at the cost of some additional storage space, as we will see in Sections 6.3 and 6.4.

6.3 Performance Evaluation

The performance overhead of Longitude mainly comes from the cryptographic operations. Here we present our performance evaluation of these cryptographic operations

in terms of execution time. All the numbers are the average time in milisecond for 10 executions.

Operation	Time (ms)	Energy (mJ)
User Key Pair Generation	1693	945
Re-encryption Key Generation	1160	635
Public Key Encryption: Pre-computing Phase	427	245
Public Key Encryption: Final-encryption Phase	0.3	0.2
Stream Cipher Encryption	0.6	0.2
Public Key Decryption	32	10.5
Stream Cipher Decryption	1	0.7
Strong Revocation: User Key Update	94	14.6
Strong Revocation: Re-encryption Key Update	697	395

Table 1. Speed & energy consumption of Cryptographic operations on Nexus One Phone

The results of the client side tests are summarised in Table 1. The client runs on a Nexus One phone which has a 1GHZ Qualcomm QSD8250 CPU and 512 MB DRAM. From the table we see that the most time-consuming operation is the user key pair generation operation, which takes about 1.7 seconds. This should not be a problem because the user only runs it once when starting to use the service. Similarly, the other key generation and key update operations are slow but run only occasionally. The frequently used operations are encryption and decryption . The stream cipher encryption and decryption are very fast and can be done in 0.6 and 1 millisecond respectively. The public key decryption operation is much faster comparing to the public key encryption operation. As we can see, the optimisation we mentioned in section 6.2 can improve the performance significantly. The final-encryption phase is extremely fast, less than 1 ms.

The only cryptographic operation that runs at the service provider is the re-encryption operation. We measured this on a MacBook Pro laptop with an Intel Core2 Duo 2.5 GHZ CPU and 4 GB RAM. The operation takes 42 milliseconds.

6.4 Energy Consumption

We also measured the energy consumption of the client side cryptographic operations on the Nexus One. The measurement was done using PowerTutor [32]. The results are shown in Table 1 and given in Millijoules.

The capacity of the standard battery of Nexus One (1400mAh, 3.7V) is 18648 Joules. Therefore, 1000 full encryptions (including the pre-computing, the final-encryption and the stream encryption operations) will consume about 1.3% of the battery energy. If the pre-computing is done beforehand, then only the final-encryption and the stream encryption operations are needed for real-time encryption. In this case, 1000 encryptions will consume only 0.002% of the battery energy. The space overhead of storing 1000 precomputed values is about 200 KB. For decryption, 1000 decryptions (including the

public key decryption and the the stream decryption operation) will consume 0.06% of the battery energy.

An interesting question is how long can 1000 pre-computed values last? Will they run out before the next recharge? In most cases, no. Apparently, the more frequently the phone updates its location, the faster the stored values will be exhausted. However, GPS and wireless radio are energy consuming. Therefore, the more frequently the phone updates its location, the shorter the battery life is. For a heavy user who updates his location every minute, the battery usually lasts less than a day. While 1000 pre-computed values last 16.7 hours in this case. If the update frequency is 10 minutes, then the battery will last 2-3 days while 1000 pre-computed values will last about 7 days.

6.5 Communication Overhead

The location ciphertexts produced by the stream cipher have the same length as the location plaintexts. Therefore the communication overhead comes from the encrypted random stream cipher key. The ciphertext of a encrypted key consists of an elliptic curve point and an element in the field F_{p^2}. In our setting where p is 512-bits, the size of the ciphertext is about 1500 bits after point compression [33]. Further optimisation is possible by choosing elliptic curves with a larger embedding degree and by using compressed pairings [34].

6.6 Security Evaluation

In Section 5 we proved that Longitude's proxy re-encryption is semantically secure, which means that an adversary cannot get any information about the user's location by directly examining the ciphertext. However, there are three possible indirect attacks.

Since location data is sent through the Internet, an adversary may be able to infer the user's location given the user's IP address. Fortunately, this attack only allows the adversary to get an imprecise location, usually to the level of city or organisation. In addition, most mobile operators provide only a NATed Internet access, which means that an adversary will only see the gateway's IP address thus it is even harder for the adversary to infer the user's location. Therefore, in Longitude we did not implement any IP obfuscation mechanism. If needed, an external service such as Tor [35] could be used to provide anonymised communication.

If a query for a user's location is followed by a location-based query to another service provider, for example, a map-service, like Google Maps, then it's possible for the location sharing service to collude with the other service to correlate the two requests to discover a user's location. To counter this attack, the application would need to use offline data or perform requests to several *suitably* random locations.

Although precision masks allow users to be imprecise about their location, they do not prevent a recipient or intelligent software from inferring a more precise location, for example, by using background knowledge (user's home, workplace, favourite shops, previous locations). Depending on the circumstances and the intent of the user, Longitude mobile applications could generate precision masks more intelligently using viable but incorrect locations. However, even with cleverer concealment it's always possible that a recipient will learn the user's exact location and rightly or wrongly infer that the

user is deliberately concealing their exact location from them, leading to a loss of trust and perhaps the recipient reciprocating or taking some other action.

7 Conclusion and Future Work

In this paper, we presented a new privacy preserving location sharing protocol called Longitude. The most significant features of Longitude are that the location sharing provider only processes encrypted locations that it unable to decrypt, supports different granularities of locations for different receivers, and low key management, computation and communication overheads. In addition, Longitude's proxy re-encryption scheme is provably secure and the cryptographic functions optimised for mobile platforms. A prototype was implemented in Java on the Nexus One Android mobile phone and the CPU-time and energy consumption were evaluated.

One type of privacy policy which has proven to be useful in location sharing services are selective location-based policies. For example, Alice may, when at home, only want her families to be able to track her but not her friends. This type of policy can be easily implemented if the location sharing service provider has access to the user's location. But how could we support this type of policy is the provider only holds encrypted data? We plan to investigate this problem further, looking at schemes such as searchable encrypted data [22] and attributed-based encryption [36]. We would also like to explore how to provide more services upon encrypted data, as suggested in [37].

References

1. : Google Latitude. http://www.google.com/latitude
2. : Yahoo fire eagle. http://fireeagle.yahoo.net/
3. : Loopt. http://www.loopt.com/
4. Raphael, J.: Three Reasons Why I Won't Be Using Google Latitude. http://www.pcworld.com/article/158953 (2009)
5. Turoczy, R.: Google latitude: Ready to tell your friends (and google) where you are? http://www.readwriteweb.com/archives/google_latitude_location_aware.php (2009)
6. Gralla, P.: Privacy group asks ftc to investigate google. http://www.pcworld.com/businesscenter/article/161497/privacy_group_asks_ftc_to_investigate_google.html (2009)
7. Tang, K.P., Lin, J., Hong, J.I., Siewiorek, D.P., Sadeh, N.: Rethinking location sharing: Exploring the implications of social-driven vs. purpose-driven location sharing. In: UbiComp. (2010)
8. Tsai, J.Y., Kelley, P.G., Cranor, L.F., Sadeh, N.: Location-sharing technologies: Privacy risks and controls. http://cups.cs.cmu.edu/LBSprivacy/files/TsaiKelleyCranorSadeh_2009.pdf (2010)
9. Beresford, A., Stajano, F.: Location privacy in pervasive computing. Pervasive Computing, IEEE **2**(1) (jan-mar 2003) 46 – 55
10. Gruteser, M., Grunwald, D.: Anonymous usage of location-based services through spatial and temporal cloaking. In: MobiSys. (2003)

11. Hashem, T., Kulik, L.: Safeguarding location privacy in wireless ad-hoc networks. In: UbiComp'07: Proceedings of the 9th international conference on Ubiquitous computing, Berlin, Heidelberg, Springer-Verlag (2007) 372–390
12. Zhong, G., Hengartner, U.: A distributed k-anonymity protocol for location privacy. In: Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on. (9-13 2009) 1 –10
13. : Locaccino. http://www.locaccino.org
14. Langheinrich, M.: A privacy awareness system for ubiquitous computing environments. In: UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing, London, UK, Springer-Verlag (2002) 237–245
15. Freudiger, J., Neu, R., Hubaux, J.P.: Private sharing of user location over online social networks. In: 3rd Hot Topics in Privacy Enhancing Technologies (HotPETs 2010). (2010)
16. Ruppel, P., Treu, G., Küpper, A., Linnhoff-Popien, C.: Anonymous user tracking for location-based community services. In: LoCA. (2006) 116–133
17. Zhong, G., Goldberg, I., Hengartner, U.: Louis, lester and pierre: Three protocols for location privacy. In: Privacy Enhancing Technologies. (2007) 62–76
18. Mascetti, S., Freni, D., Bettini, C., Wang, X.S., Jajodia, S.: Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies. CoRR **abs/1007.0408** (2010)
19. Siksnys, L., Thomsen, J.R., Saltenis, S., Yiu, M.L.: Private and flexible proximity detection in mobile social networks. In: Mobile Data Management. (2010) 75–84
20. Blaze, M., Bleumer, G., Strauss, M.: Divertible protocols and atomic proxy cryptography. In: EUROCRYPT. (1998) 127–144
21. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans. Inf. Syst. Secur. **9**(1) (2006) 1–30
22. Dong, C., Russello, G., Dulay, N.: Shared and searchable encrypted data for untrusted servers. In: DBSec. (2008) 127–143
23. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. SIAM J. Comput. **32**(3) (2003) 586–615
24. : Android platform. http://www.android.com/
25. Chatterjee, S., Sarkar, P., Barua, R.: Efficient computation of tate pairing in projective coordinate over general characteristic fields. In: ICISC. (2004) 168–181
26. Scott, M.: Computing the tate pairing. In: CT-RSA. (2005) 293–304
27. NIST: NIST FIPS-197: Specification for the ADVANCED ENCRYPTION STANDARD
28. NIST: NIST SP 800-38A: Recommendation for Block Cipher Modes of Operation
29. Fournel, N., Minier, M., Ubéda, S.: Survey and benchmark of stream ciphers for wireless sensor networks. In: WISTP. (2007) 202–214
30. Even, S., Goldreich, O., Micali, S.: On-line/off-line digital schemes. In: CRYPTO. (1989) 263–275
31. Guo, F., Mu, Y., Chen, Z.: Identity-based online/offline encryption. In: Financial Cryptography. (2008) 247–261
32. Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R., Mao, Z.M., Yang, L.: Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: Proceedings of CODES+ISSS. (2010)
33. IEEE: IEEE P1363: Standard specifications for public key cryptography
34. Scott, M., Barreto, P.S.L.M.: Compressed pairings. In: CRYPTO. (2004) 140–156
35. : Tor. http://www.torproject.org/
36. Lewko, A.B., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In: EUROCRYPT. (2010) 62–91

37. Popa, R.A., Zeldovich, N., Balakrishnan, H.: Cryptdb: A practical encrypted relational dbms. Technical Report MIT-CSAIL-TR-2011-005, MIT (2011)