This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2514338, IEEE Transactions on Services Computing

Privacy-Preserving Location Sharing Services for Social Networks

Roman Schlegel, *Member, IEEE,* Chi-Yin Chow, *Member, IEEE,* Qiong Huang, *Member, IEEE,* and Duncan S. Wong, *Member, IEEE*

Abstract—A common functionality of many location-based social networking applications is a location sharing service that allows a group of friends to share their locations. With a potentially untrusted server, such a location sharing service may threaten the privacy of users. Existing solutions for *Privacy-Preserving Location Sharing Services* (PPLSS) require a trusted third party that has access to the exact location of all users in the system or rely on expensive algorithms or protocols in terms of computational or communication overhead. Other solutions can only provide approximate query answers. To overcome these limitations, we propose a new encryption notion, called *Order-Retrievable Encryption* (ORE), for PPLSS for social networking applications. The distinguishing characteristics of our PPLSS are that it (1) allows a group of friends to share their exact locations without the need of any third party or leaking any location information to any server or users outside the group, (2) achieves low computational and communication cost by allowing users to receive the exact location of their friends without requiring any direct communication between users or multiple rounds of communication between a user and a server, (3) provides efficient query processing by designing an index structure for our ORE scheme, (4) supports dynamic location updates, and (5) provides personalized privacy protection within a group of friends by specifying a maximum distance where a user is willing to be located by his/her friends. Experimental results show that the computational and communication cost of our PPLSS is much better than the state-of-the-art solution.

Index Terms—Location privacy, location sharing services, order-retrievable encryption, location-based social networking, spatio-temporal query processing

1 Introduction

Many location-based service providers today provide users with services related to their locations by making use of GPS-enabled mobile devices, wireless communication and spatial database management systems. A popular type of such services is for a user to search for points of interest in the vicinity (e.g., dining and shopping). Recently, location-based services have been combined with online social networks, where user-generated, geo-tagged information is shared among people who are part of a social network. A common functionality of many existing location-based social networking systems is *location sharing services* that allow users to discover the current location of their friends and notify the users when a friend is in the vicinity or within a certain distance, e.g., Facebook's Places [1], Foursquare [2], Google Plus [3], and Loopt [4].

Existing location-based social networking systems with location sharing services rely on a central server which receives location information from all users in the system. The problem with this approach is that the central server can generate a detailed movement profile of each user (e.g., the location, time and frequency of each place which has been visited by each user) and that raises privacy concerns [5]–[7]. Existing privacy-preserving location sharing schemes aim to protect the user location privacy against the central server, but they still allow the server to provide the user with the necessary services. However, in some existing schemes, the central server still knows the user's approximate location [8]–[11]. Other schemes require several messages to be exchanged not only between the user and the central server but also directly between the user and the user's friends [12], [13], increasing the communication cost and making those schemes less practical. Other schemes only return approximate results [14], making them less useful. Peer-to-peer (P2P) systems, where users' devices would directly communicate without an intermediate server are inherently difficult to realize in mobile phone networks as they typically make use of NAT (network address translation), restricting direct communication between devices in the process [15].

In this paper, we propose a new encryption notion, called *Order-Retrievable Encryption* (ORE); a new cryptographic protocol that realizes our *Privacy-Preserving Location Sharing Services* (PPLSS) for social networking systems. In particular, our ORE scheme enables users to browse their friends' *exact* locations within a certain distance without revealing any information about their locations to any other users or a social networking service provider. The framework of our PPLSS consists of a database server (which is maintained by the social networking service provider) and users. The users send their location information in encrypted form to the database server according to our ORE scheme. When a user wants to locate his/her friends in the vicinity, the user logs onto the social networking system, sends a *location query* (e.g., "*Q1: Send me the location of my friends within 2 km of my current location*") to the database server, and obtains the requested location information in encrypted form based on our ORE scheme. The user then recovers the actual location of his/her

- *R. Schlegel is with Corporate Research, ABB Switzerland Ltd., Baden-Dättwil, Switzerland. *C.-Y. Chow and *D. S. Wong are with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. *Q. Huang is with College of Informatics, South China Agricultural University, China. (*Authors are ordered alphabetically).*
E-mails: rs@ione.ch, chiychow@cityu.edu.hk, csqhuang-c@my.cityu.edu.hk, and duncan@cityu.edu.hk

friends from the encrypted information returned by the database server.

The key distinguishing characteristics of our PPLSS based on the proposed ORE scheme are: (1) *Secure location privacy.* PPLSS does not disclose any location information of its users to a central server or an eavesdropper, not even an approximate location, and does not require any third party. (2) *Low computational and communication cost.* It allows a user to receive the exact location information of his/her friends without requiring direct communication between users or multiple rounds of communication between a user and a server. (3) *Index structure.* We design an index structure for our ORE scheme to index encrypted locations of a group of friends to improve the efficiency of location query processing. (4) *Efficient data updates.* Our scheme supports highly dynamic location updates from individual users efficiently. (5) *Personalized privacy within a group of friends.* Each user is able to specify a maximum distance defining a personalized privacy region so that only those friends who are within the region can locate the user. The rationale behind such personalized privacy is that users may not want to share their locations with far-away friends as it might not be practical or necessary to share their location with friends at large distances.

Regarding security requirements, we consider the database server to be *honest-but-curious*, namely, the database server handles queries, stores data received from users and sends data to users who are making queries without tampering with the data. However, the database server also attempts to find out the location of users in the system. Note that the term "privacy-preserving" refers to the location privacy of users rather than keeping their identities private. In addition to the security analysis of our PPLSS, we also compare the performance of our scheme to that of the state-of-the-art cryptography-based scheme [16] through experiments. The results show that our PPLSS outperforms the work [16] in terms of both communication cost and query processing performance.

The rest of this paper is organized as follows: Section 2 gives an overview of our PPLSS and the ORE scheme. The details of our PPLSS are presented in Section 3. Section 4 extends the PPLSS to support personalized privacy. Section 5 discusses the security requirements of the ORE scheme and Section 6 gives an ORE construction which is implemented in our experiment and also provides its security analysis. Section 7 covers the security analysis of our PPLSS. Section 8 presents experimental results. Section 9 surveys related work and Section 10 concludes this paper.

2 Overview of PPLSS and ORE

In this section, we first describe the system model of our *Privacy-Preserving Location Sharing Services* (PPLSS) for social networking applications, and then give the definition of our *Order-Retrievable Encryption* (ORE) scheme.

2.1 System Model

Our PPLSS framework consists of a *database server* and a set of (mobile) *users*. The database server is maintained by a social networking service provider. Fig. 1 illustrates the PPLSS framework, in which each user sends his/her location in encrypted form according to our ORE scheme to the database server. When a user wants to query the exact location of his/her friends who are within a distance specified by the user, the user sends a *location query* in the form of a private location-based range query, like *Q1* given in Section 1, to the database server. The database server is equipped with a privacy-aware query processor that has the ability to provide an exact query answer for the user based on the user's encrypted location and his/her friends' encrypted locations without knowing any location information about the query and the users. Finally, the user decrypts the query answer and browses his/her friends' locations displayed on a road map. It is important to note that all user locations and location queries are encrypted using our ORE scheme (its definition will be described in Section 2.2) before they are sent to the database server.

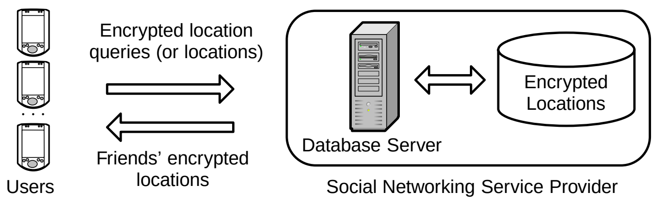

Encrypted location queries (or locations)
Friends' encrypted locations
Users
Encrypted Locations
Database Server
Social Networking Service Provider

Fig. 1. The System Model of PPLSS.

In PPLSS, we assume that the database server is *honest-but-curious*, i.e., it follows our designed protocol, but it attempts to infer the user's location. On the other hand, the user trusts his/her friends (i.e. other users in his/her friend list in the social network context). The user constructs a trusted group in which they share their locations through private location queries according to our ORE scheme. The security threat models and the security analysis of PPLSS will be given in Sections 6 and 7, respectively.

2.2 Order-Retrievable Encryption

As mentioned above, user locations (i.e., points) in the database server are always in encrypted form. When an "encrypted" query location of *Q1* for a group of friends is received by the database server, the database server should determine for any two friends' encrypted locations within the group which of them is closer to the encrypted query location. To achieve this, we use our proposed encryption notion ORE for geographical data.

An ORE scheme is a symmetric key encryption scheme with two additional functions: one is for generating *encrypted* query locations and the other one is for the database server to determine which one between two encrypted user locations is closer to an encrypted query location. The scheme is called ORE because the order of the encrypted user locations in terms of their distances from any given encrypted query location can be retrieved. Note that the actual distance information is not retrievable.

In the formal definition of ORE below, we assume that each distinct location in PPLSS can be represented uniquely using an element in a d-dimensional space and without loss of generality, suppose that $\mathcal{R}$ is the space of each dimension. One additional remark is that the ORE scheme defined below can be viewed as *a collection of one-way functions* [17] and this one-way function has the order retrievability property. In other words, our PPLSS framework does NOT need the decryption algorithm of the ORE scheme. Below are the details.

Definition 1. An Order-Retrievable Encryption (ORE) scheme consists of four probabilistic polynomial-time (PPT) algorithms.

- $SK_G \leftarrow \mathsf{KGen}(1^\lambda, \mathcal{R})$. The symmetric key generation algorithm KGen takes a security parameter $\lambda \in \mathbb{N}$ and the dimensional space $\mathcal{R}$ defined above, and outputs a symmetric key.
- $C \leftarrow \mathsf{Enc}(SK_G, P)$. The encryption algorithm takes SK_G and a d-dimensional point $P \in \mathcal{R}^d$, and outputs a ciphertext as an "encrypted" location.
- $\xi \leftarrow \mathsf{QGen}(SK_G, Q)$. The query generation algorithm takes SK_G and a query point $Q \in \mathcal{R}^d$, outputs an "encrypted" query location.
- $C_b \leftarrow \mathsf{Cmp}(\xi, C_0, C_1)$. The comparison algorithm takes an encrypted query location ξ and two encrypted locations C_0 and C_1, outputs C_b for $b \in \{0, 1\}$ if

$$\mathsf{dist}(Q, P_b) \leq \mathsf{dist}(Q, P_{1-b}) \tag{1}$$

where $\xi \leftarrow \mathsf{QGen}(SK_G, Q)$ and $C_i \leftarrow \mathsf{Enc}(SK_G, P_i)$ for $i = 0, 1$, and $\mathsf{dist}(P, Q)$ represents the actual distance between two locations P and Q. We stress that the function Cmp neither has SK_G as input nor has to output any further information about the evaluation of dist other than which of P_0 and P_1 is closer to Q.

We note that there are two distinct encryption algorithms, Enc and QGen. Our ORE scheme uses different encryption algorithms for locations used as query locations (QGen) and for locations encrypted to compare them to a given query locations (Enc), hence these two distinct algorithms.

Optionally, a decryption algorithm can be defined as $P/\perp \leftarrow \mathsf{Dec}(SK_G, C)$, which takes SK_G and a ciphertext C, and outputs a d-dimensional point $P \in \mathcal{R}^d$ or $\perp$ indicating the failure of decryption. We do not require the domain of ciphertexts C to be in any special form related to the plaintext space $\mathcal{R}^d$. Also note that our PPLSS framework does NOT need the decryption function Dec. The pair $(\mathsf{KGen}, \mathsf{Enc})$ can be viewed as a one-way function collection indexed by SK_G. The security parameter λ relates the security strength of the ORE scheme to the security level of a secure symmetric key encryption scheme. For example, $\lambda = 80$ refers to the 80-bit security level [18]. Section 5 discusses the security requirements of the ORE scheme, and Section 6 provides an ORE construction, based on a scheme proposed by Wong et al. [19], and its security analysis.

ORE vs. Order-Preserving Encryption. A related work called Order-Preserving Encryption (OPE) [20], [21] preserves the numerical ordering of the plaintexts in the ciphertexts. Formally, for any $A, B \subseteq \mathbb{N}$ with $|A| \leq |B|$, an encryption family $\mathcal{E} : \mathcal{K} \times A \rightarrow B$ is order preserving if for all $i, j \in A$, $E(k, i) > E(k, j)$ if and only if $i > j$, for any $k \in \mathcal{K}$, where $\mathcal{K}$ is the key space of the encryption family. OPE is different from ORE. The OPE maintains the order information in encryption while ORE destroys the order information so that given any two ciphertexts encrypted using ORE, the order information is not preserved. Instead, the ORE ciphertexts can be used with an auxiliary function called Cmp which gets an encrypted query location involved, and the function Cmp can tell which of the two ciphertexts contain a location which is closer to the query location, i.e., the ordering is with respect to the distance to a query location. Though OPE has been used for many other applications such as efficient range queries, indexing and query processing, OPE does not have the function of Cmp as we defined in ORE, and therefore, it is not known if it is possible to use OPE for constructing a privacy-preserving location sharing system. More discussions are given in Sec. 9.

3 PPLSS: Privacy-Preserving Location Sharing Services

In this section, we describe our PPLSS for social networking applications based on our *Order-Retrievable Encryption* (ORE) scheme. We will first present the ORE scheme for PPLSS, and then propose an index structure that makes use of the relative distance information provided by the ORE scheme to improve query processing efficiency.

3.1 The ORE Scheme

The main idea of our PPLSS is that a user or a *group initiator* registers with the system to create a *user group*. The group initiator then adds friends to the user group and creates a *shared group key* SK_G according to our ORE scheme and another *shared data key* SK_D for AES [22] encryption of location data (this is needed as our ORE scheme does not require a decryption function, and the actual location data exchanged between users is therefore AES-encrypted). It is important to note that users or group initiators are not required to register with their real identity. They can use pseudonyms as long as their friends are able to recognize them (friends can also communicate their pseudonyms out-of-band, e.g., through email). After the shared group key SK_G and the shared data key SK_D are securely delivered to all group members, each member periodically reports his/her encrypted location to the database server. When a user logs onto the system and wants to browse the location of his/her friends within a certain user-specified distance, the user issues a *location query* with an encrypted query location and an encrypted location marker to the database server. The database server is able to provide an exact answer for the user without knowing any location information of the user and his/her friends. In general, the ORE scheme involves seven major message exchanges for three operations among a group initiator u, the database server, and u's friends, as depicted in Fig. 2. The algorithms used in the ORE scheme are defined in Section 2.2, the symbols used in the ORE scheme are summarized in Table 1, and the three operations are explained in more detail below.

3.1.1 User Group Formation

A user u registers with the database server with an identity (or pseudonym) ID_u and creates a user group G that includes u, i.e., u is the *group initiator*. The group initiator can manage the members of G. Upon creation of G, a random group identifier ID_G (e.g., a random 128-bit string) is created. The group initiator can then invite users to join G, and invited users can either accept or decline to join G. The group initiator generates two random shared keys: a shared group key SK_G and a shared data key SK_D for G and sends the shared keys to each new group member through a secure channel. The establishment of the secure channel can be done using conventional two-party authenticated key establishment protocols [23], [24]. To enable the removal of users from a group, e.g., defriending, the shared keys (SK_G, SK_D) can be re-generated by the group initiator or any legitimate member in G. A user who is part of several groups will obtain the required keys for each group the user is

TABLE 1
Key symbols in the ORE or ORE-Index protocol.

Symbol	Description	Algorithm
Loc_u	Plaintext location of user u	-
Loc_{marker}	Plaintext query marker point for ORE	-
Loc_{R_i}	Plaintext location marker point of index ring R_i for ORE-Index	- -
(Loc_{min}, Loc_{max})	Plaintext query marker points for ORE-Index	-
$dist_u$	User-specified distance for a location query	-
$dist_{max}$	User-specified maximum distance for ORE-Index	-
$dist_{priv}$	User-specified privacy distance for a personalized privacy region	-
SK_G	Shared group key for ORE	ORE.KGen
SK_D	Shared data key for AES	AES.KGen
C	Encrypted user location using ORE	ORE.Enc
D	Encrypted user location using AES	AES.Enc
ξ	Encrypted query location or reference point using ORE	ORE.QGen
ψ	Encrypted location marker using ORE	ORE.Enc
κ	Encrypted privacy marker using ORE for personalized privacy region	ORE.Enc

a part of. Fig. 2 shows that this step involves Messages 1 to 3, where a group initiator u registers with the database server to create a group G (Message 1) and adds five members to G, i.e., $G = \{ID_u, ID_{m_1}, ID_{m_2}, \ldots, ID_{m_5}\}$ (Message 2). After u generates the shared keys (SK_G, SK_D) for G, u sends the keys directly to each member in G through a secure communication channel (Message 3).

3.1.2 User Location Update

After invited users agree to join the group G and receive the shared keys (SK_G, SK_D) from the group initiator u, each member m in G periodically sends $\langle ID_G, C_m, D_m\rangle$ to the database server, where ID_G is the group identity, C_m is the ORE encryption of m's location Loc_m, that is, $C_m \leftarrow \mathsf{ORE.Enc}(SK_G, Loc_m)$, and $D_m \leftarrow \mathsf{AES.Enc}(SK_D, Loc_m \| ID_m \| r_m)$ is the AES encryption of m's location, the identity ID_m (or pseudonym) of m in G, and a random number r_m. If m belongs to n different groups, i.e., $G_1, \ldots, G_n$, m will encrypt its location using ORE.Enc under each of $SK_{G_1}, \ldots, SK_{G_n}$ and AES.Enc under each of $SK_{D_1}, \ldots SK_{D_n}$ to generate C_{m_i} and D_{m_i}, respectively, for each group G_i, where $1 \leq i \leq n$, and send $\langle (ID_{G_1}, C_{m_1}, D_{m_1}), \ldots, (ID_{G_n}, C_{m_n}, D_{m_n})\rangle$ to the database server. The location sent to the database server is always encrypted, so at no point is the server able to determine the actual location of a user. Fig. 2 illustrates the two messages exchanged in this step, where the encrypted location of the group initiator u (Message 4) and that of other members in the group G (Message 5) are reported to the database server.

Notice that besides ORE, we also use AES to encrypt each group member's location. The purpose of including the AES encryption is to improve the communication efficiency of the next step, namely, location query processing. We will see in Section 6 that the ciphertext of ORE is about 40 times bigger in size than that of AES. Hence, we use an additional AES module for sending encrypted locations when answering a location query below. This is also the reason why the ORE scheme does not need the decryption algorithm (see Section 2.2).

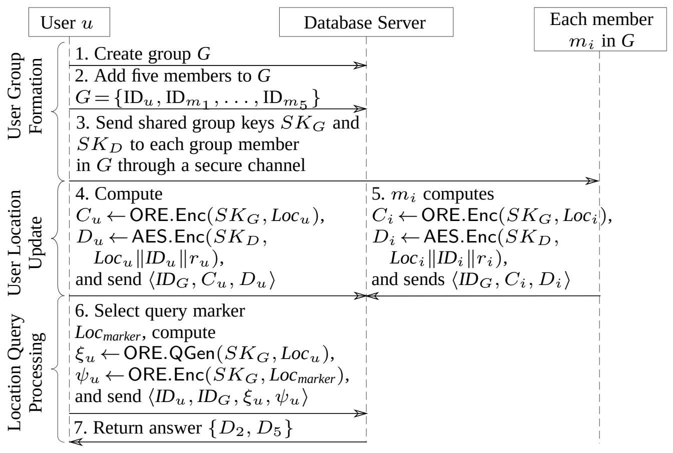

User u
Database Server
Each member m_i in G
User Group Formation
1. Create group G
2. Add five members to G $G = \{\mathrm{ID}_u, \mathrm{ID}_{m_1}, \ldots, \mathrm{ID}_{m_5}\}$
3. Send shared group keys SK_G and SK_D to each group member in G through a secure channel
User Location Update
4. Compute $C_u \leftarrow \mathsf{ORE.Enc}(SK_G, Loc_u)$, $D_u \leftarrow \mathsf{AES.Enc}(SK_D, Loc_u \| ID_u \| r_u)$, and send $\langle ID_G, C_u, D_u\rangle$
5. m_i computes $C_i \leftarrow \mathsf{ORE.Enc}(SK_G, Loc_i)$, $D_i \leftarrow \mathsf{AES.Enc}(SK_D, Loc_i \| ID_i \| r_i)$, and sends $\langle ID_G, C_i, D_i\rangle$
Location Query Processing
6. Select query marker Loc_{marker}, compute $\xi_u \leftarrow \mathsf{ORE.QGen}(SK_G, Loc_u)$, $\psi_u \leftarrow \mathsf{ORE.Enc}(SK_G, Loc_{marker})$, and send $\langle ID_u, ID_G, \xi_u, \psi_u\rangle$
7. Return answer $\{D_2, D_5\}$

Fig. 2. Message flows in the ORE Scheme for PPLSS.

3.1.3 Location Query Processing

Location query generation (by user). If a user u logs onto the system and wants to view the exact location of his/her friends within a user-specified distance $dist_u$, u generates a *location query* by (1) encrypting its current location Loc_u using the *query encryption* ORE.QGen under SK_G to generate an encrypted *query location*, that is, $\xi_u \leftarrow \mathsf{ORE.QGen}(SK_G, Loc_u)$, (2) randomly selecting a *query marker* Loc_{marker} which is a point on the circle centered at Loc_u with a radius of $dist_u$, and (3) encrypting Loc_{marker} using the *location encryption* ORE.Enc under SK_G, that is, $\psi_u \leftarrow \mathsf{ORE.Enc}(SK_G, Loc_{marker})$. Then, u sends the location query $\langle ID_u, ID_G, \xi_u, \psi_u\rangle$ to the database server.

Query processing (by server). Given the location query from the user u, the database server first finds all the members in G with group identifier ID_G. Suppose there are n members in $G = \{m_1, m_2, \ldots, m_n\}$. The database server then performs a sequential scan of G by executing the comparison algorithm ORE.Cmp for the encrypted location of each member in G based on u's encrypted query location ξ_u and query marker ψ_u. If $C_{m_i} \leftarrow \mathsf{ORE.Cmp}(\xi_u, C_{m_i}, \psi_u)$ for some $1 \leq i \leq n$, it means that the actual location of the member m_i is located within the distance $dist_u$ of u; D_{m_i} is added to an answer set A (note that only the AES encryption of member m_i's location D_{m_i} is included, but C_{m_i} is not). After performing the comparison algorithm for each member in G, the answer A is sent to u. Finally, u uses AES.Dec to decrypt the location of each friend in A under the shared data key SK_D locally and u can browse their location information displayed on an underlying road map.

Fig. 3 depicts u's specified distance $dist_u$ and the exact locations of u and other members in u's group G. Loc_{marker} is a point on the circle centered at u with a radius of $dist_u$. After u generates a location query, the query is sent to the database server (Message 6 in Fig. 2). Our ORE scheme enables the database server to compute that there are two members m_2 and m_5 within distance $dist_u$ from u without letting the database server know any location information of u or any other member in G. In Message 7, the AES encrypted locations of m_2 and m_5, i.e., D_2 and D_5, respectively, are returned to u.

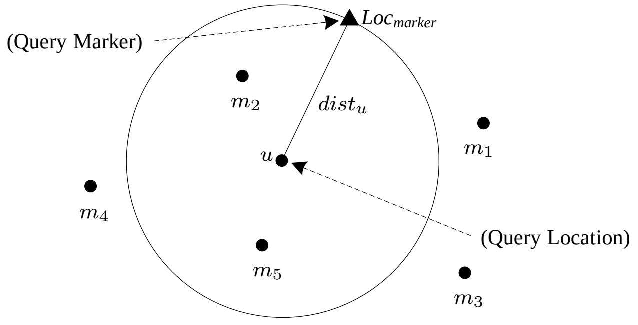


Fig. 3. A location query example.

3.2 ORE-Index: The ORE Scheme with an Index Structure

In a typical location-based social networking system (e.g., Google Latitude [25]), when a user u logs onto the system, a location query is automatically sent to the database server. Then, the database server periodically evaluates the query answer to allow u to keep track of the exact location of his/her friends within u's specified distance $dist_u$. The ORE scheme using the sequential scan proposed above takes $O(n)$ time to evaluate the location query answer, where n is the number of members in u's group G, because the database server has to perform the comparison algorithm $\mathsf{ORE.Cmp}$ for each member in G.

Although the number of friends of each user is usually not large[1], using the sequential scan approach to process a large number of location queries at a high frequency, i.e., with a small evaluation time interval, would put a considerable computational burden on the database server. In fact, many users would be subscribing to the location sharing service through their mobile devices[2], and the evaluation period for their location queries should therefore be as small as possible to provide more accurate location information of their friends. To avoid sequential scan of the members in G for each query evaluation, in the following we propose a tree-like index structure for managing the ORE encrypted data. We call the structure *ORE-Index* and it aims to improve the efficiency of processing location queries. Although index structures have previously been proposed for location-based queries [27], they rely on the assumption that the database server knows the location information of all users; they are not applicable to PPLSS where the database server does not have the location information of users or queries.

3.2.1 Basic Idea

When a user u generates an *initial* location query, u estimates a circular region in which it will be located for a certain time period, e.g., one hour. A simple way to calculate the radius of the region $dist_{max}$ is multiplying the time period by the maximum legal speed in the system area. More sophisticated ways can be used to compute $dist_{max}$, as the index structure does not have any assumption on how $dist_{max}$ is computed. The index structure is built based on a set of *encrypted location markers* and *an encrypted reference point* generated by u to index the members in u's group G. Once established, the index structure will be used to evaluate u's location queries. Note that the estimation of $dist_{max}$ will not affect the answer accuracy; it only influences the frequency of rebuilding the index, which becomes necessary whenever the required search area of u's location query is outside the region defined by $dist_{max}$. In general, our PPLSS using the ORE-Index scheme has two major phases, namely, index construction and location query processing.

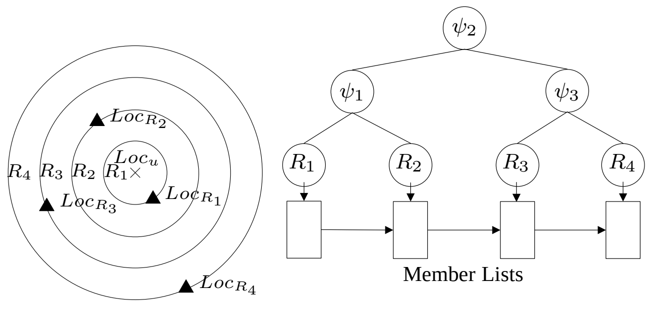
1. Average Facebook user has 130 friends [26].
2. There are more than 200 million active users currently accessing Facebook through their mobile devices [26].

3.2.2 Index Construction

Index construction request (by user). The generation of an index construction request requires a querying user u to determine a radius $dist_{max}$ for a circular area $\mathcal{A}$ in which u will be located for a certain time period. The database server only needs to build the requested index once and then u can reuse the index as long as the required search areas of u's subsequent location queries are within $\mathcal{A}$. $\mathcal{A}$ is then divided into N non-overlapping rings (or donut shapes), i.e., $R_1, R_2, \ldots, R_N$, (note that the innermost shape is in fact a circle). For each ring R_i, a point with the maximum distance to u's location Loc_u is randomly selected as a *location marker* Loc_{R_i}. Then, Loc_u is encrypted using the *query encryption* $\mathsf{ORE.QGen}$ as an *encrypted reference point* of an index, i.e., $\xi_u \leftarrow \mathsf{ORE.QGen}(SK_G, Loc_u)$, and each Loc_{R_i} is encrypted using the *location encryption* $\mathsf{ORE.Enc}$, i.e., $\psi_i \leftarrow \mathsf{ORE.Enc}(SK_G, Loc_{R_i})$ as the encrypted location of a node in the index. u sends an index construction request along with the encrypted reference point and the set of encrypted location markers to the database server, i.e., $\langle ID_u, ID_G, \xi_u, \psi_1, \psi_2, \ldots, \psi_N \rangle$.

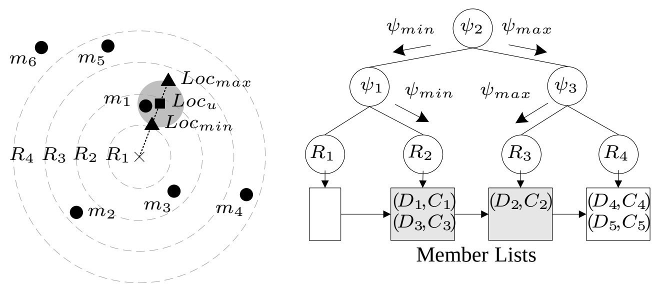
Notice that our scheme has no assumption about N and the width of each ring, i.e., the rings of an index can have different widths. In practice, N can be a user or system parameter. A larger N requires u to generate a larger set of encrypted location markers which incurs a higher cost of generating an index construction request. However, a larger set of encrypted location markers results in an index with more levels which improves query processing efficiency by reducing the number of false positives in a candidate answer. The detail of the query processing step will be discussed in Section 3.2.3. Thus, N can be adjusted as a trade-off between the cost of generating an index construction request on the client side and the query processing efficiency. For simplicity, we adjust N relative to u's specified search range $dist_u$ in a location query, i.e., $N = \lfloor dist_{max}/(dist_u \times \alpha) \rfloor$, where α ($\alpha > 0$) is a system parameter to tune the performance trade-off. A smaller α leads to an index with more levels, and vice versa.

Index construction (by server). When the database server receives the index construction request from the user u, ξ_u is used as the encrypted reference point $\xi_{\mathcal{I}_u}$ for an index structure $\mathcal{I}_u$ and the server uses a top-down approach to build $\mathcal{I}_u$ based on the (one-dimensional) relative distance information between the encrypted reference point and each encrypted location marker. Starting from the root node, the 1-st to N-th ring are split into two groups with respect to a key computed as ψ_i, where $i = \lfloor (1+N)/2 \rfloor$; hence, one group contains the 1-st to the i-th ring and the other group contains the $(i+1)$-st to the N-th ring. The root node keeps ψ_i as a key. Then, these two groups are recursively split until each leaf node contains only one ring. Each leaf node maintains a member list of the members in G located within the area of the corresponding ring. The database server also maintains a hash table, where an entry contains a member identity with a pointer to the leaf node whose member list contains the member. A singularly linked list is built on these member lists to facilitate range searches.



(a) Index request (b) Index structure

Fig. 4. Index construction.



(a) Location query (b) Query processing

Fig. 5. Location query processing.

Fig. 4 shows an index structure $\mathcal{I}_u$ for a querying user u, where u's location Loc_u (represented by a cross) is encrypted using ORE.QGen as an encrypted reference point ξ_u and u's specified region $\mathcal{A}$ is divided into four rings R_1 to R_4. On the outer boundary of each ring R_i ($1 \leq i \leq 4$), a point is randomly selected as a location marker Loc_{R_i} (represented by a triangle), which is encrypted using ORE.Enc as ψ_i. To construct $\mathcal{I}_u$, the key of the root node is ψ_2, as $\lfloor(1+4)/2\rfloor = 2$. The key of its left child is ψ_1, as $\lfloor(1+2)/2\rfloor = 1$. The key of its right child is ψ_3, as $\lfloor(3+4)/2\rfloor = 3$. Since each leaf node corresponds to one ring, the construction of $\mathcal{I}_u$ is complete (Fig. 4b).

Index maintenance (by server). Since $\mathcal{I}_u$ is constructed based on u's set of encrypted location markers, no split and merge operations are necessary on insertion and deletion, respectively. Three operations are required to maintain $\mathcal{I}_u$. (1) ***Insertion.*** To insert a member m with an encrypted location C_m into $\mathcal{I}_u$, we first check whether m is located in u's specified region $\mathcal{A}$. If $\mathsf{ORE.Cmp}(\xi_{\mathcal{I}_u}, \psi_N, C_m)$ returns ψ_N, m is outside $\mathcal{A}$ and m will not be inserted into $\mathcal{I}_u$. Otherwise, the server navigates $\mathcal{I}_u$ recursively from the root node to insert m. Starting from the root node with a key ψ_i, if $\mathsf{ORE.Cmp}(\xi_{\mathcal{I}_u}, \psi_i, C_m)$ returns C_m, we search its left subtree; otherwise, we search its right subtree. This procedure is repeated until a leaf node is reached where m's AES and ORE encrypted locations (D_m, C_m) are added to the member list. Then the hash table is updated accordingly. (2) ***Deletion.*** When a member m logs off from the system or moves outside the user's specified region $\mathcal{A}$, m will be deleted from $\mathcal{I}_u$. To do so, we access the hash table to find the leaf node whose member list contains m, and then remove m from the member list. (3) ***Update.*** When a member m moves from ring R_i to ring R_j, where $i \neq j$, we perform the deletion operation to remove m from R_i and then perform the insertion operation to insert m to R_j.

Fig. 5a shows that five members, m_1 to m_5, in user u's group G are within u's specified region $\mathcal{A}$. These five members are inserted into $\mathcal{I}_u$. For member m_1, since $\mathsf{ORE.Cmp}(\xi_{\mathcal{I}_u}, \psi_2, C_{m_1})$ returns C_{m_1}, we descend to the left child. Then, since $\mathsf{ORE.Cmp}(\xi_{\mathcal{I}_u}, \psi_1, C_{m_1})$ returns ψ_1, we descend to the right child. As a leaf node R_2 is reached, m_1's AES and ORE encrypted locations (D_1, C_1) are added to its member list. Similarly, other members m_2 to m_5 are inserted into $\mathcal{I}_u$ (Fig. 5b).

3.2.3 Location Query Processing

Location query generation (by user). A location query is periodically sent to the database server by the user u to keep track of the locations of his/her friends within u's specified distance $dist_u$. To generate a location query, u computes a required search area as a circle with a radius of $dist_u$ centered at its location Loc_u. Note that u can reuse the index built by the database server (i.e., no index construction request is needed) as long as the required search area is within the region $\mathcal{A}$ of the index. u finds two points Loc_{min} and Loc_{max} that are the closest and farthest points, respectively, within the required search area compared to the original query point which was encrypted as a reference point when its index construction request was generated. Then, Loc_{min} and Loc_{max} are encrypted by using the *location encryption* as $\psi_{min} \leftarrow \mathsf{ORE.Enc}(SK_G, Loc_{min})$ and $\psi_{max} \leftarrow \mathsf{ORE.Enc}(SK_G, Loc_{max})$, respectively, and Loc_u is encrypted using the *query encryption* as $\xi_u \leftarrow \mathsf{ORE.QGen}(SK_G, Loc_u)$. The location query $\langle ID_u, ID_G, \xi_u, \psi_{min}, \psi_{max}\rangle$ is sent to the database server.

Query processing (by server). When the database server receives a location query, it searches the index structure $\mathcal{I}_u$ of u for ψ_{min} and ψ_{max} with respect to its encrypted reference point. For ψ_{min}, the search starts from the root node with key ψ_i, if $\mathsf{ORE.Cmp}(\xi_{\mathcal{I}_u}, \psi_i, \psi_{min})$ returns ψ_{min}, we search its left subtree; otherwise, we search its right subtree. This process is repeated until a leaf node n_{min} is reached. Likewise, a leaf node n_{max} is found for ψ_{max}. The server then goes through the member list of every node between n_{min} and n_{max} (inclusive) and adds their members to a candidate answer set A. Since $\mathcal{I}_u$ only considers one-dimensional relative distance information, there may be some false positives in A. Thus, for each member m in A, the server performs a filtering step by using the comparison algorithm in the ORE scheme. If $\mathsf{ORE.Cmp}(\xi_u, \psi_{min}, C_m)$ returns ψ_{min}, m is removed from A. After that, a set of the AES encrypted locations of the members in A is returned to u as a query answer.

Fig. 5 depicts a location query of u with a location Loc_u in our running example, where the shaded circle indicates the required search area defined by $dist_u$. Loc_{min} and Loc_{max} are represented by triangles. Since the required search area is within the region $\mathcal{A}$ of the index (Fig. 4), u can reuse the index to process the location query without requesting the database server to construct a new index. The database server finds a leaf node n_{min} for the encrypted form of Loc_{min}, ψ_{min}, i.e., $n_{min} = R_2$, and a leaf node n_{max} for ψ_{max}, i.e., $n_{max} = R_3$. All the members in the member list of every leaf node from R_2 to R_3 are added to A, i.e., $A = \{(D_1, C_1), (D_2, C_2), (D_3, C_3)\}$. After removing false positives from A, one member remains in $A = \{(D_1, C_1)\}$ and a query answer $\{D_1\}$ is returned to u.

4 Personalized Privacy Regions

In this section, we further improve the privacy of individual users in our PPLSS using the ORE or ORE-Index scheme by allowing them to define their *personalized privacy regions*. In the ORE and ORE-Index schemes described in Sections 3.1 and 3.2, respectively, the querying user u is theoretically free to choose a location marker at a considerably larger distance than is practical (e.g., 1000 km). The database server would return all friends in u's group within that distance, allowing u to learn their location even though there is no practical need to know their location at such large distances. Personalized privacy regions are an extension to the PPLSS which help prevent this situation, by allowing individual users to specify a maximum distance $dist_{priv}$ up to which members of their groups are allowed to locate them. $dist_{priv}$ is chosen by each individual user and it is applied whenever other users of the group run location queries. This means that even when a user runs a location query with a location marker at a distance of 1000 km, if all other users of the group have defined a privacy region of 1 km, the query will only return people at most 1 km away from the location of the querying user (because for all other users their privacy region requirements would not be met).

The main idea of personalized privacy regions is exemplified in Fig. 6, where user Q runs a location query with friend y at location P_y as one of Q's group members. User y has defined a privacy region with a maximum distance $dist^y_{priv}$, which is represented by a circle centered at y with a radius of $dist^y_{priv}$, by randomly selecting a *privacy marker* Loc^y_{priv} on the boundary of y's privacy region. Consider that Q runs a location query with a range distance $dist_Q$ which includes y, but Q is outside the privacy region of y (Fig. 6a), the database server will *not* return the location of y in the answer set. If, on the other hand, Q is inside the privacy region of y (Fig. 6b), the database server will return the location of y as part of the answer to Q.

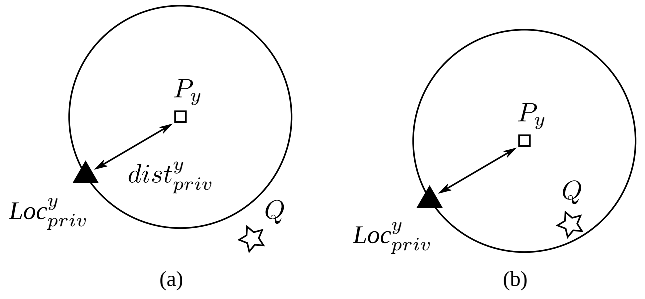


Fig. 6. In (a) the privacy region requirement of y is not satisfied, while in (b) it is satisfied.

In the following, we will describe our ORE and ORE-Index schemes for PPLSS when extended with personalized privacy regions.

4.1 Extension to the ORE Scheme

We will discuss how to extend our ORE scheme to support personalized privacy regions.

4.1.1 User Group Formation

The user group formation step is identical to the group formation for the ORE scheme in Section 3.1.1.

4.1.2 User Location Update

When having personalized privacy regions, the difference to the location update of the ORE scheme described in Section 3.1.2 is that a user not only updates its own location but also sends information about his/her privacy region. To this effect, each user decides on his/her personal privacy region by choosing a distance $dist_{priv}$. When doing a location update, a user u picks a *privacy marker* Loc^u_{priv} by randomly selecting a point on the circle with radius $dist^u_{priv}$ centered at u's location Loc_u. u then encrypts Loc^u_{priv} using the *location encryption* $\mathsf{ORE.Enc}$ under SK_G, i.e., $\kappa_u \leftarrow \mathsf{ORE.Enc}(SK_G, Loc^u_{priv})$. In addition to calculating $C_u \leftarrow \mathsf{ORE.Enc}(SK_G, Loc_u)$ and $D_u \leftarrow \mathsf{AES.Enc}(SK_D, Loc_u \| ID_u \| r_u)$ (where r_u is a random number), u encrypts its location Loc_u using the *query encryption* $\mathsf{ORE.QGen}$ under SK_G, i.e., $\xi_u \leftarrow \mathsf{ORE.QGen}(SK_G, Loc_u)$. u will then send the location update $\langle ID_G, C_u, D_u, \kappa_u, \xi_u \rangle$ to the database server.

Notice that if u belongs to multiple groups $G_1, \ldots, G_n$, u is able to specify a different $dist_{priv}$ for each group based on u's desired privacy requirements, e.g., u is willing to always disclose location information to his/her family, i.e., $dist_{priv} = \infty$, but only a small privacy region, e.g., $dist_{priv} = 1$ km, for his/her colleagues. When u generates a location update, u encrypts a privacy marker for each group G_i under the corresponding shared group key SK_{G_i} and sends the location update $\langle (ID_{G_1}, C_{u_1}, D_{u_1}, \kappa_{u_1}, \xi_{u_1}), \ldots, (ID_{G_n}, C_{u_n}, D_{u_n}, \kappa_{u_n}, \xi_{u_n}) \rangle$ to the database server.

4.1.3 Location Query Processing

When using personalized privacy regions, the location query processing is divided into two parts. In the first part, the database server checks which members in the group are within the distance specified by the querying user, as described in Section 3.1.3. In the second part, for each member m_i in an answer set A, the database server checks whether the querying user is within the privacy region of m_i. If this is not the case (i.e., the privacy requirement of m_i is not met), m_i is removed from the answer set A.

The generation of a location query is similar to the ORE scheme in Section 3.1.3. When requesting location sharing services, a user u will send a location query along with his/her encrypted location using the ORE scheme $\langle ID_u, ID_G, C_u, \xi_u, \psi_u \rangle$ to the database server, where $\xi_u \leftarrow \mathsf{ORE.QGen}(SK_G, Loc_u)$ and $\psi_u \leftarrow \mathsf{ORE.Enc}(SK_G, Loc^u_{marker})$. In the first part, for each member m_i of the group with identity ID_G except u, the database server runs the comparison algorithm $\mathsf{ORE.Cmp}(\xi_u, C_i, \psi_u)$. Whenever the comparison returns C_i, m_i is added to an answer set A. In the second part, for each member m_j in the answer set A, the database server runs the comparison algorithm again, this time for the privacy marker, by calculating $\mathsf{ORE.Cmp}(\xi_j, C_u, \kappa_j)$. Whenever the algorithm returns C_u, the querying user u is inside the privacy region of m_j, and thus, m_j remains in the answer set A. However, if the comparison returns κ_j, m_j is removed from A. Finally, a query answer that contains the AES encrypted location of each remaining member in A is returned to u.

Fig. 7 shows an example where a group of user u contains four friends m_1 to m_4 and u's specified distance $dist_u$ is represented by a dotted circle. The first part of the query processing uses the comparison algorithm $\mathsf{ORE.Cmp}$ to find that two members with C_2 and C_4 are within $dist_u$ of u, and thus, they are added to an answer set A, i.e., $\{m_2, m_4\}$. The second part of the query processing removes C_2 from A because u is outside the privacy region of m_2. Finally, m_4's AES encrypted location, $\{D_4\}$, is returned to u.

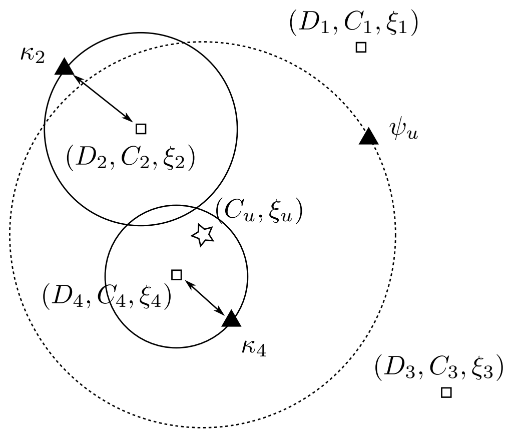


Fig. 7. The ORE scheme with personalized privacy regions.

4.2 Extension to the ORE-Index Scheme

The extension of the ORE-Index scheme to support personalized privacy regions is very similar to the ORE scheme. The only difference is that the database server first searches the index constructed for a querying user u to find a candidate answer set A. For each member m_i in A, if the ORE comparison algorithm $\mathsf{ORE.Cmp}$ indicates that (1) m_i is a false positive as in the original ORE-Index scheme (Section 3.2.3) or (2) the querying user is outside m_i's privacy region, m_i is removed from A. After that a set of the AES encrypted location of each remaining member in A constitutes a query answer returned to u.

5 Security Requirements of ORE

In addition to the correctness requirement below, a secure ORE encryption should satisfy two additional requirements, confidentiality of encrypted points and confidentiality of query points. We start with the correctness requirement as follows.

1. Correctness. For all shared group key $SK_G \leftarrow \mathsf{KGen}(1^\lambda, \mathcal{R})$ and $P_0, P_1, Q \in \mathcal{R}^d$, where P_0 and P_1 are two plaintext location points and Q is a plaintext query location point, we have $C_b \leftarrow \mathsf{Cmp}(\mathsf{QGen}(SK_G, Q), \mathsf{Enc}(SK_G, P_0), \mathsf{Enc}(SK_G, P_1))$ if and only if $\mathsf{dist}(Q, P_b) \leq \mathsf{dist}(Q, P_{1-b})$ where $b \in \{0, 1\}$ and $\mathsf{dist}(P, Q)$ represents the actual distance between two locations P and Q.

2. Data Confidentiality. We require that an adversary $\mathcal{A}$ should not be able to recover the plaintext points from their encrypted form. More precisely, there are two levels to consider for the data confidentiality requirement:

- **Level 1:** $\mathcal{A}$ observes a set of ciphertext points $\{C_i\}_{1\leq i\leq n}$ and also knows a set of plaintext points $\{P_j\}_{1\leq j\leq n}$, but does not know which ciphertext point in $\{C_i\}_{1\leq i\leq n}$ corresponds to which plaintext point in $\{P_j\}_{1\leq j\leq n}$. Now given a new ciphertext point C^*, $\mathcal{A}$ is to output the corresponding plaintext point $P^* \in \mathcal{R}^d$.
- **Level 2:** $\mathcal{A}$ knows the correspondence of plaintext points in $\{P_i\}_{1\leq i\leq n}$ and ciphertext points in $\{C_i\}_{1\leq i\leq n}$. Now given a new ciphertext point C^*, $\mathcal{A}$ is to output the corresponding plaintext point $P^* \in \mathcal{R}^d$.

The adversary $\mathcal{A}$ above can be considered as a malicious Social Network Service Provider (SNSP) which has access to the encrypted locations of users when the users update their locations. The malicious SNSP may also obtain the plaintext locations of some of the users.

3. Query Confidentiality. This requirement concerns the confidentiality of the query location, namely, an adversary $\mathcal{A}$ should not be able to find out a query location Q from its encrypted form $\xi \leftarrow \mathsf{QGen}(SK_G, Q)$. We may also consider the adversary as a malicious SNSP which tries to recover the location of a user making a query. In particular, given a set of ciphertext points $\{C_i\}_{1\leq i\leq n}$, a set of encrypted query locations $\{\xi_j\}_{1\leq j\leq \ell}$, and a challenging encrypted query location $\xi^* \leftarrow \mathsf{QGen}(SK_G, Q^*)$ for a randomly picked query Q^*, $\mathcal{A}$ is to find out Q^*.

In the next section (Section 6), we will describe how to construct a secure ORE scheme.

6 An ORE Construction

We now describe a construction of the *Order-Retrievable Encryption* (ORE) defined in Section 2.2. The construction is based on an encryption scheme recently proposed by Wong et al. [19]. We call their scheme the *WCKM encryption scheme*. The WCKM encryption scheme matches our definition of an ORE scheme. The rest of this section contains the following three aspects.

1) Review the WCKM basic encryption scheme according to the ORE definition given in Section 2.2
2) Describe a new attack showing that the basic scheme does not satisfy Level 1 of Data Confidentiality given in Section 5
3) Describe the final extended WCKM encryption scheme

Below is the review of the WCKM basic encryption scheme according to our ORE definition (Section 2.2):

Symmetric Key Generation. Suppose that all the points are in a d-dimensional space and $\mathcal{R}$ is the space of each dimension. Given a security parameter $\lambda \in \mathbb{N}$ and dimension space $\mathcal{R}$, KGen outputs a symmetric key SK_G as a randomly chosen *invertible* $(d+1) \times (d+1)$ matrix where each element is in $\mathcal{R}$. In the following, we assume that all elements in matrices and vectors are in $\mathcal{R}$, and $\mathcal{R}$ is of integers in a certain range, which will be defined in each concrete scheme.

Encryption. Given SK_G and a point P, which is a d-element vector $(p_1, p_2, \cdots, p_d) \in \mathcal{R}^d$, the encryption algorithm Enc prepares a $(d+1)$-element vector $\hat{P}$ as follows:

$$\hat{P} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_d \\ -0.5||P||^2 \end{pmatrix} \tag{2}$$

and calculates a ciphertext point $C = SK_G^T \hat{P}$, where $||P||$ represents the Euclidean norm of point P. Note that $||P||^2$ can be represented by $P \cdot P$ where $\cdot$ represents the scalar product.

Decryption. Given SK_G and a ciphertext point C which is a $(d+1)$-element vector $(c_1, c_2, \cdots, c_{d+1}) \in \mathcal{R}^{d+1}$, the decryption algorithm Dec recovers the original point P by computing

$$P = \pi_d SK_G^{T^{-1}} C \tag{3}$$

where $SK_G^{T^{-1}}$ is the inverse of SK_G^T and π_d removes the $(d+1)$-th dimension by setting $\pi_d = (I_d, 0)$ with I_d the d-dimensional identity matrix and "0" a column vector of zeros. π_d is thus a $d \times (d+1)$ matrix.

Query Generation. Given SK_G and a *query* point $Q = (q_1, q_2, \cdots, q_d) \in \mathcal{R}^d$, the query generation algorithm QGen picks a random $r > 0$ and creates a $(d+1)$-dimensional point $\hat{Q}$ as

$$\hat{Q} = r \begin{pmatrix} q_1 \\ \vdots \\ q_d \\ 1 \end{pmatrix} \quad (4)$$

and calculates a ciphertext query point $Y = SK_G^{-1}\hat{Q}$.

Comparison. Given two ciphertext location points C_0 and C_1 and one ciphertext query point Y, suppose that $C_i \leftarrow \mathsf{Enc}(SK_G, P_i)$ for $i = 0, 1$ and $Y \leftarrow \mathsf{QGen}(SK_G, Q)$ where $P_0, P_1, Q \in \mathcal{R}^d$. The comparison algorithm Cmp calculates the following to determine which ciphertext location point is closer to the encrypted point Y:

$$(C_0 - C_1) \cdot Y > 0 \quad (5)$$

If so, the output is set to C_0; otherwise, the output is set to C_1.

In the following, we analyze the security of the WCKM encryption scheme and show that besides the settings suggested in [19], additional conditions have to be introduced in order to ensure its security against Level 1 of Data Confidentiality requirement defined in Section 5.

6.1 Security Analysis

In [19], the authors described a bruteforce attack which entails a total of ${}_nP_{d+1} = \mathcal{O}(n^{d+1})$ trials of potential symmetric keys that an adversary needs to try if a set of n ciphertexts $\{C_i\}_{1\le i\le n}$ and plaintext points $\{P_j\}_{1\le j\le n}$ are given but the correspondence between the ciphertexts and plaintexts are not known. In each trial, the adversary performs no more than n decryptions. Hence as stated in [19], if $n = 10K$ and $d = 2$ (i.e. a 2-dimensional geographical data set), the adversary has to spend more than 310 years to test out all trial symmetric keys if the adversary can perform $1M$ decryptions per second. This bruteforce attack falls in the setting of Level 1 of Data Confidentiality given in Section 5.

We observe, however, that the setting $(n = 10K, d = 2)$ may not be secure enough for achieving 80-bit security (i.e. $\lambda = 80$) which is considered as the minimum security requirement for symmetric key security [18]. First of all, we can see that the number of trial symmetric keys is ${}_{10K}P_3 < 2^{40}$. The estimation given in [19] relies on the assumption that the adversary can perform at most $1M$ decryptions per second. This might be the case if the adversary can unleash the computational power of only a few machines. However, as finding the symmetric key SK_G will enable the adversary to access the entire database, there is a strong incentive to devote more resources to the cracking task. One example is to make use of a botnet which usually contains hundreds of thousands of nodes [28]. Some botnets even have more than one million computers that can be utilized by an adversary to launch a bruteforce attack. Suppose $100K$ computers in a botnet are involved in the cracking task and each of them can run $10K$ decryptions per second, then the time required for finding SK_G in the example above will be significantly reduced to just four months.

A New Bruteforce Attack. We propose a new bruteforce attacking technique which is different from the one described in [19], while it will be more effective in recovering the key SK_G when the value of d is small, as in the example above. For each row of $\pi_d SK_G^{T^{-1}}$ (in Equation 3), there are d+1 elements and each element is in $\mathcal{R}$. The scalar product of row i of $\pi_d SK_G^{T^{-1}}$ and the $(d+1)$-element vector C (in Equation 3) is the i-th element of the corresponding plaintext point P. A bruteforce attack can be launched which can find out the i-th row of $\pi_d SK_G^{T^{-1}}$. The bruteforce attack can be launched independently for each row of $\pi_d SK_G^{T^{-1}}$. Once all the d rows of $\pi_d SK_G^{T^{-1}}$ are found, the adversary is then able to decrypt all the other ciphertexts by following the decryption algorithm (in Equation 3).

Let $(e_{i,1}, e_{i,2}, \ldots, e_{i,d+1}) \in \mathcal{R}^{d+1}$ be the $d+1$ elements on i-th row of $\pi_d SK_G^{T^{-1}}$. For each trial sequence of $(e_{i,1}, e_{i,2}, \ldots, e_{i,d+1})$, the adversary performs a decryption for each ciphertext in $\{C_i\}_{1\le i\le n}$ and checks if the i-th element in the decrypted point is equal to the i-th element of any plaintext in $\{P_j\}_{1\le j\le n}$. This is carried out for all the n ciphertexts in $\{C_i\}_{1\le i\le n}$. If all the checks are passed, then the adversary finds the correct $(e_{i,1}, e_{i,2}, \ldots, e_{i,d+1})$.

The total number of trial values for $(e_{i,1}, e_{i,2}, \ldots, e_{i,d+1})$ is $|\mathcal{R}|^{d+1}$ for each row of $\pi_d SK_G^{T^{-1}}$. Since the bruteforce attack can be launched independently for each row, the total number of attempts that the adversary needs to try for finding the values of all the d rows of $\pi_d SK_G^{T^{-1}}$ is $d|\mathcal{R}|^{d+1}$. Depending on the cardinality of $\mathcal{R}$, the adversary may spend less effort to crack the system. Suppose that $d = 2$ and $\mathcal{R} = [-1K, 1K]$. Then the total number of possible candidates for SK_G is $2 \cdot 2000^3 \le 2^{34}$ which does not satisfy 80-bit symmetric key security.

6.2 The Final ORE Construction

To defend against the new bruteforce attack above, the dimension d of the scheme can be augmented, for example, by setting $d \ge 80$. In this way, even if SK_G is a binary matrix, the scheme can still provide at least 80-bit symmetric key security against the bruteforce attack above.

For Level 2 of Data Confidentiality (Section 5), dimension augmentation is not enough as the adversary knows the correspondence between the ciphertexts in $\{C_i\}_{1\le i\le n}$ and the plaintext points in $\{P_i\}_{1\le i\le n}$. Hence the adversary can recover SK_G after getting $d+1$ pairs of plaintext points P_i and their encrypted counterparts C_i. In the following, we review a technique called *secret splitting configuration* which was proposed in [19]. The technique can be used to achieve Level 2 of Data Confidentiality.

Instead of generating one transformation matrix, we now choose two matrices for the ORE scheme, e.g. SK_{G_0} and SK_{G_1}. For every extended location point p (i.e., a point augmented with random dimensions) to be encrypted, we split it into two parts p_a, p_b so that $p = p_a + p_b$. Note that for any query point q it holds that $p \cdot q = p_a \cdot q + p_b \cdot q$. We then encrypt p_a and p_b under SK_{G_0} and SK_{G_1}, respectively, e.g. $C_a \leftarrow SK_{G_0}^T p_a$ and $C_b \leftarrow SK_{G_1}^T p_b$. A query point q is also encrypted twice, namely, we compute $Y_a \leftarrow SK_{G_0}^{-1} q$ and $Y_b \leftarrow SK_{G_1}^{-1} q$. We then have $C_a \cdot Y_a + C_b \cdot Y_b = p_a \cdot q + p_b \cdot q = p \cdot q = C \cdot Y$. The same technique can also be applied to the query point. That is, we can choose to split a query point to two parts, e.g. $q = q_a + q_b$, and encrypt each part under the corresponding secret key.

However, as analyzed in [19], the split technique alone does not improve the security. Therefore, we consider the *secret splitting configuration*. Specifically, we choose a secret configuration, which is a vector of bits, e.g. $\vec{b} \leftarrow (b_1, \cdots, b_d)^T$ where

$b_i \in \{0, 1\}$ for $i = 1, 2, \cdots, d$. If $b_i = 1$, we split p_i (the i-th entry of a location point p) to two parts, e.g. $p_i = p_{a,i} + p_{b,i}$, and copy q_i (the i-th entry of a query point q) twice, e.g. $q_{a,i} = q_{b,i} = q_i$; otherwise, we split the q_i to two parts, e.g. $q_i = q_{a,i} + q_{b,i}$, and copy p_i twice. The configuration is secretly shared among all the users in the same group. We then have $\sum_{i=1}^{d}(p_{a,i}q_{a,i} + p_{b,i}q_{b,i}) = p \cdot q$. Since the configuration is unknown to the adversary and there are in total 2^d many possible choices, the enhanced scheme is 2^d more costly for the adversary to break than the original ORE scheme.

7 The Security Analysis of Our PPLSS Using ORE and ORE-Index

There are different security aspects to consider in our *Privacy-Preserving Location Sharing Services* (PPLSS) for social networking appliations. In the following, we start with a security model.

7.1 Security Model

In our security model, we consider the database server as an adversary which tries to locate one user in a group of n users, all of which are mutually friends with each other. The group is denoted as $G = \{u_1, u_2, \cdots, u_n\}$ where the secret keys shared by the group members are (SK_G, SK_D). The adversary (i.e. the database server) has access to data received from all the members in G. It can also collude with eavesdroppers and all other users in the system who are not in G. We say that the adversary is considered to have broken our PPLSS if the adversary is able to find out the location of any user in G solely from the data received from the n group members u_1 to u_n. We do not consider physical or side-channel attacks such as the adversary finding out a user's location through other means, for example, by tracking the cell towers that are communicating with the user. Once again, a privacy-preserving location sharing system is for protecting the location privacy of users. The adversary knows the identity (or pseudonym) of each user in the system.

We also assume that the database server is curious but honest. It might try to determine the locations of users as described above, but it will run the algorithms honestly without denying service to any user. We also assume that no user in G colludes with the adversary. A user possesses the secret keys for encryption and decryption and would thus be able to decrypt all location information from other users if he/she colluded with the database server. Therefore, users are assumed not to share the secret keys SK_G and SK_D with the server. However, the database server can have secret keys of all other users in the system who are not in G.

7.2 Location Privacy against Service Provider

In our PPLSS, all the points sent to the server by users in the system are encrypted either using our ORE scheme introduced in Section 2.2 (i.e., C_i, ξ_i, ψ_i and κ_i) or using AES [22] encryption (i.e., D_i). Because the encryption does not preserve distance, the server cannot gain any information from the encrypted points alone. Furthermore, the only operation possible on the encrypted points is relative distance comparisons, but without knowing the corresponding actual location of at least two points even distance comparisons do not reveal useful information. In the following, we consider the correlation of several types of encrypted points.

Query/Normal Points. In general, the database server can only run following distance comparisons:

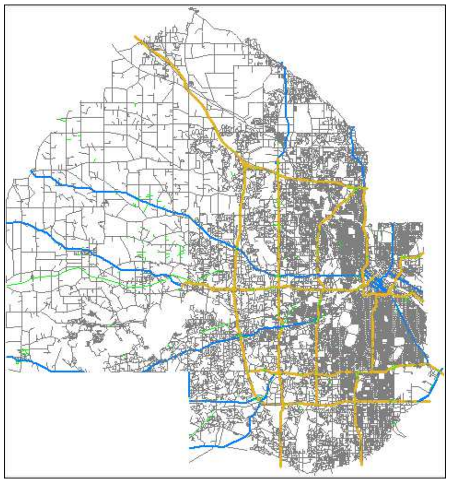


Fig. 8. The road network of Hennepin County, MN, USA.

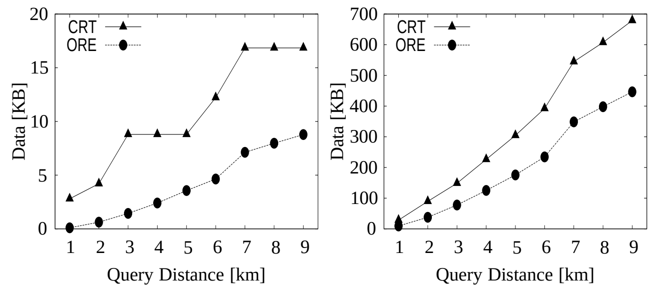
1) An encrypted location query point ξ and an encrypted query marker ψ or an encrypted user location point C_u
2) An encrypted user location point ξ_u (for the personalized privacy region scheme) and an encrypted privacy marker κ_u or an encrypted location query point ξ
3) An encrypted location query point ψ_{min} (or ψ_{max}) and an encrypted reference point $\xi_{\mathcal{I}_u}$ or the encrypted key ψ_i of a node in index $\mathcal{I}_u$

Because distances are not preserved for encrypted points, the database server can only run comparisons between points. For an arbitrary user at location Loc_u (given as C_u), the database server can only determine if Loc_u is closer to Q (given as ξ) than Loc_{marker} (given as ψ). The case of Loc_u farther away than Loc_{marker} does not reveal any additional information, and the case of Loc_u being closer than Loc_{marker} does not narrow down the possible region of a user either as the database server does not know the value of $dist$. However, in practice users tend to choose $dist$ in some predictable way, for example, 1 km for users who are walking. In this way, the database server may be able to tell whether a user x is in proximity of the querying user while another user, say user y is not. Nevertheless, the database server may only find out their *relative* proximity rather than their exact locations.

8 Experimental Results

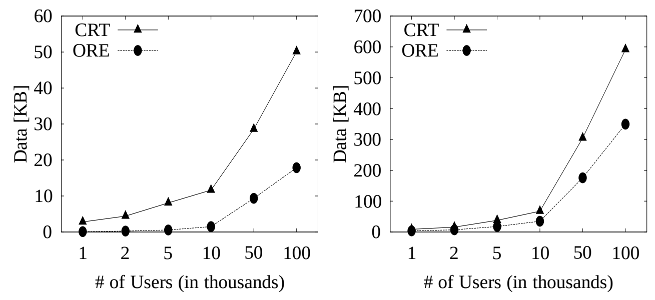
To evaluate the performance of our *Privacy-Preserving Location Sharing Services* (PPLSS) using the *Order-Retrievable Encryption* with the sequential scan (ORE) scheme and ORE with the proposed index structure (ORE-Index), and also to compare them to the state-of-the-art cryptography-based privacy-preserving query processing technique for spatial data, namely, the CRT scheme described in [16], we implemented a simulator in Java to run both our ORE and ORE-Index schemes and the CRT scheme [16]. CRT is an interactive protocol for location queries over spatial data, making use of R*-trees and cryptography-based transformations on location data to protect the privacy of the data. In all experiments, we generated a set of moving users on the road network of Hennepin County, Minnesota, USA, as illustrated in Fig. 8. The input road network is extracted from the Tiger/Line files that are publicly available[3]. The total area of the Hennepin County is

3. U.S. Census Bureau. Topologically Integrated Geographic Encoding and Referencing system (TIGER). http://www.census.gov/geo/www/tiger/.



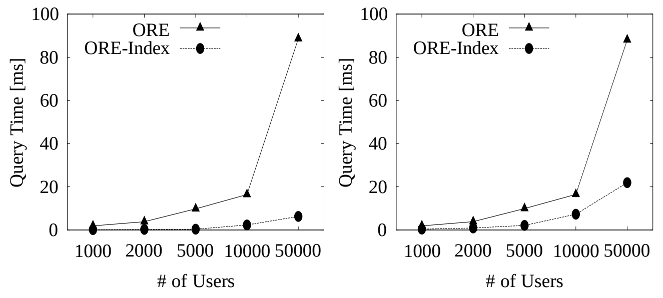
(a) Group size of 1,000 Users (b) Group size of 50,000 Users

Fig. 9. Communication cost of CRT and ORE ($dist$).



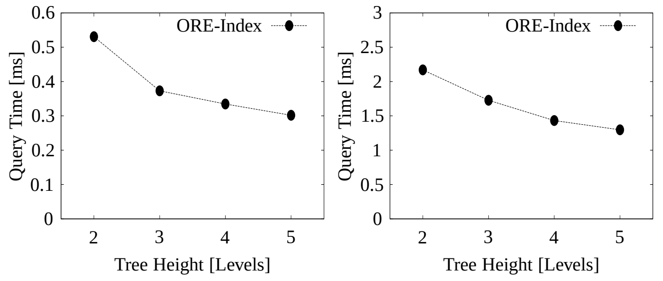
(a) $dist = 1$ km (b) $dist = 5$ km

Fig. 10. Communication cost of CRT and ORE (group size).



(a) $dist = 1$ km (b) $dist = 5$ km

Fig. 11. Query time of ORE and ORE-Index (group size).



(a) $dist = 1$ km (b) $dist = 5$ km

Fig. 12. ORE-Index scheme (index levels).

1,571 km^2. The road map has 57,020 edges and 42,135 vertices. Users are initially distributed among the vertices, and then move along the roads at speeds between 50 and 70 miles per hour. Unless mentioned otherwise, the default number of friends per user (the user group size) is 5,000, and users issue location queries with a query range distance $dist$ of 1 km and an index area with a radius $dist_{max}$ of 10 km. For the ORE-Index scheme, the default index height is three, i.e., an index contains eight rings. With the default $dist_{max}$ and index height, the default index ring width is 1.25 km, and hence, the default ratio of the query range distance $dist$ to the index ring width is $1/1.25 = 0.8$. All experiments were run on a machine with an Intel Core 2 Duo 3.16GHz CPU and 3.25GB of RAM.

8.1 Comparing ORE and CRT

We first compared our ORE scheme with the CRT scheme. Although several schemes were proposed in [16], CRT is the only one which offers the same strong privacy guarantees as our ORE scheme. Because both CRT and ORE target the mobile environment, we focused on comparing their communication cost. Our ORE scheme only returns the exact results to the user, while CRT requires the user to run several rounds (navigating an encrypted R-tree) and filter the returned results locally.

Comparing ORE and CRT when varying the range distance of location queries, i.e., $dist$, gives the results shown in Fig. 9a and 9b for group sizes of 1,000 users and 50,000 users, respectively. The result shows that our ORE scheme typically needs to transfer less than half the amount of data than CRT for small groups, and about two thirds of the amount of data as the group size grows larger. ORE is especially more efficient if the query range distance is comparatively small. Fig. 10a shows the comparison of the communication cost for $dist = 1$ km where up to a group size of 100,000 users the cost of ORE is a small fraction of the cost of CRT. The difference gets smaller as the query range distance increases to 5 km (Fig. 10b), but ORE still requires only half to two thirds of the data transmitted compared to CRT.

8.2 Comparing ORE and ORE-Index

The second experiment was designed to compare the efficiency of the ORE scheme with the ORE-Index scheme. Because both schemes return the exact result to the user, the amount of data transmitted is identical. We therefore focused the comparison on the query time, i.e., the processing time required by the database server to run a query. The result shown in Fig. 11a confirms that the ORE-Index scheme is indeed an order of magnitude more efficient in terms of query processing time than the ORE scheme for relatively small query range distances $dist$, i.e., 1 km. This is due to the fact that the ORE scheme always has to search sequentially through all users in a group, while the ORE-Index scheme only compares the users in the relevant rings of the index. For larger query distances, i.e., 5 km, ORE-Index still requires only half the processing time, or even less than half as the number of users increases (Fig. 11b).

8.3 Effect of Parameters of ORE-Index

The ORE-Index scheme has a number of parameters which influence its performance. We looked at the two most important parameters among them. The first parameter is the height of an index structure. If the area covered by an index, i.e., $dist_{max}$, remains constant, varying the index tree height means varying the width of the index rings. Increasing the height of the index results in thinner rings (with a smaller total area) and vice versa. Fig. 12 shows how the required query processing time for the database server varies with the index height from two to five levels, where the group size is 5,000 users. For smaller query range distances (Fig. 12a), increasing the height from two to three

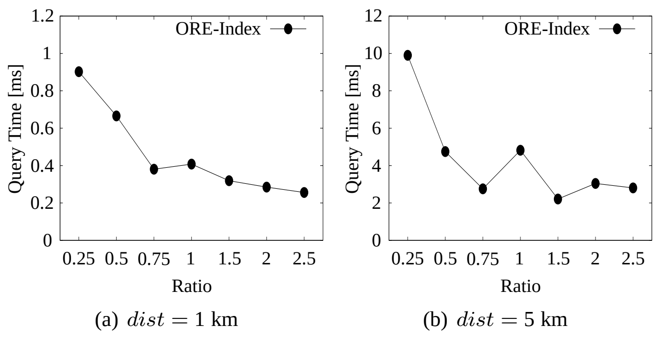


Fig. 13. ORE-Index scheme (the ratio of the query range distance to the index ring width).

(corresponding to going from three to seven rings) will yield significant improvement, while deeper trees result in somewhat smaller gains. For larger query range distances (Fig. 12b), the performance continues to improve as the index structure contains more levels.

Another important parameter of the ORE-Index scheme that influences the query processing time required by the database server is a ratio between the query range distance $dist$ and the ring width, as the ratio below one signifies that it is more likely that only one index ring has to be searched, while the ratio larger than one means that always at least two or even more index rings have to be searched. However, if the query range distance remains fixed, a smaller ratio results in a much larger area being covered by the index, resulting in more users per index ring.

Fig. 13 shows the evolution of the query processing time as the ratio varies from 0.25 to 2.5 with the same number of index rings, for 1 km and 5 km queries. For a ratio of 0.25 (meaning that the width of an index ring is 4 times the query range distance), the query processing time required by the database server is the highest, due to the fact that the index area is large, with each ring containing many users. As the ratio increases, the query processing time drops, most significantly until it reaches 0.75. Larger ratios only marginally decrease the query processing time. For larger query range distances (Fig. 13b) there is a special effect for a ratio of one, resulting in longer query processing time than a ratio of 0.75 or 1.5. On the other hand, increasing the ratio has as a consequence that the index has to be rebuilt more often because the total index area is proportionally smaller and a querying user will leave the index area of a previously built index sooner. 0.75 therefore seems to be an acceptable compromise between good query performance (in terms of query processing time) and the frequency with which the index has to be rebuilt.

9 Related Work

In this section, we survey the privacy-preserving techniques for conventional location-based services, spatial data outsourcing, and location sharing services.

Location-based services. The problem of user location privacy in location-based services has been addressed from several angles before. For example techniques such as *k-anonymity* or *location cloaking*, where the location of a user is expanded to include $k-1$ other users [8]–[11]. Another approach uses oblivious transfer or private information retrieval to allow a user to retrieve points of interest without the server knowing what was retrieved [29]–[31]. In conventional location-based services, the information held by the server (points of interest) is static, while the information held by the user (i.e., the user location) is dynamic. If location-based services are used for locating friends, on the other hand, then all information is dynamic, i.e., both the information held by the user (his/her own location) and the information held by the server (the location of all users). Privacy-preserving query processing schemes designed for conventional location-based services (such as store finders, etc.) are therefore usually not directly applicable to location-based services for locating friends, i.e., location sharing services for social networks.

Spatial data outsourcing. An order-preserving encryption scheme [20], [21] protects outsourcing data by using a bucket-based encryption E such that $E(x) < E(y)$ for every pair of values for which $x < y$. However, since the order-preserving encryption scheme can only protect data in simple numerical domains, it cannot easily be extended to protect spatial data. Another approach described in [32] for outsourcing data uses homomorphic encryption[4] to enable aggregate SQL queries over encrypted databases. The scope is very limited, though, focusing only on simple numerical domains and aggregate queries in SQL. Furthermore, the scheme has been shown to be insecure in [33].

For spatial data, one approach to preserve privacy in spatial datasets is to transform or perturb data in a way which still allows making meaningful operations on the transformed data. Both [34] and [35] suggest such kinds of distance-recoverable transformations, where the distance between points is preserved. Wong et al. showed in [19] that distance-recoverable or *general* scalar-product-preserving encryption schemes are not secure against certain attacks and in [36], Liu et al. demonstrated how the original data can be recovered in schemes such as [34] and [35]. In [19], Wong et al. introduced a scheme which is *asymmetric* scalar-product-preserving instead of *general* scalar-product-preserving, making it immune to such attacks.

A similar paper on outsourcing location data to an untrusted third party is by Yiu et al. [16]. Similar to Wong et al. [19] it transforms a database before outsourcing it to a service provider. Authorized users share a private key so they can send queries to the service provider, who can work on the transformed data to generate a response without learning any location information. Both those schemes [16], [19], however, are for outsourcing *static* data. For applications where the location of points is updated continuously, [16] for example would require the whole database to be re-transformed for each update, which is impractical.

Location sharing services. One paper proposing three different algorithms for a privacy-preserving location-based service for locating friends is by Zhong et al. [37]. Their algorithms use an additive homomorphic cryptosystem to perform secure multi-party computation. Their first scheme, Louis, allows two users to determine whether they are in proximity if and only if they are nearby, using a semi-trusted third party. Lester, the second scheme, does not need any third party and relies instead on letting a user solve a computational puzzle to determine whether another user is nearby. Each user determines the hardness of the puzzle and consequently, the amount of work is necessary for other users to find out whether they are in proximity. The third scheme, Pierre, makes use of a grid structure and encrypted grid coordinates to determine whether two users are in the same or in adjacent grid cells. There are other grid-based schemes, but they usually

4. Homomorphic encryption allows to perform addition and/or multiplication over ciphertexts such that it corresponds to the same operation over the plaintext, i.e., $\varepsilon(x) + \varepsilon(y) = \varepsilon(x + y)$, and/or $\varepsilon(x) \cdot \varepsilon(y) = \varepsilon(x \cdot y)$.

have the drawback that locations and proximity calculations are approximate because the distance between grid cells does not capture exactly the distance between users within those cells.

Another approach by Mascetti et al. in [14] uses three different protocols called *SP-Filtering*, *Hide&Seek* and *Hide&Crypt*. *SP-Filtering* computes the proximity between users with a certain degree of approximation. It requires a third party which does the computation. The third party compares so-called *granules*, which obfuscate the exact location of users to determine the approximate distance between them. If more precision is needed, *Hide&Seek* or *Hide&Crypt* is run as a second step. *Hide&Seek* starts a direct interaction between two users to get a more precise distance measurement. *Hide&Crypt* also requires direct interaction between users but uses secure computation to leak less information about the respective position of users. Nevertheless, the first step, *SP-Filtering*, will still leak the approximate location of each user to the third party.

In [13], Šikšnys et al. present an approach based on encrypted grid indices. Users share a list of grids with different levels (or resolutions). Each cell in a grid of a specific resolution can be mapped to a unique number through a one-to-one function such as AES. A server can then determine proximity by comparing these numbers, asking users to switch to a finer resolution if necessary. This requires several rounds of communication when two users are close, making it more expensive in terms of communication. A recent paper also by Šikšnys et al. [12] introduces Vicinity Locator which is similar to the Friend Locator in [13] but allows arbitrarily shaped regions of interest.

Another privacy-preserving location-sharing service proposed by Herrmann et al. [38] makes use of identity-based broadcast encryption (IBE) to realize a location-sharing service that affords location privacy with respect to the central server. One version of the scheme shares the location with friends irrespective of their relative location, leading to more data being transferred than necessary. An updated version maps locations to discrete regions to counteract this problem, but the mapping is approximate as it depends on the definition of the regions, while our scheme is exact in defining within which range to share locations. Furthermore, our scheme also provides personalized privacy regions, while their scheme has no such provisions.

Similarly, Freudiger et al. [39] also make use of broadcast encryption (albeit not identity-based) to distribute locations among friends, augmenting the system with dummy queries and caching of information required for localization to minimize leaking information through the geo-location process. In contrast, while our scheme is also cryptography-based, our scheme minimzes overhead by enabling the server to only send relevant locations as the response to a query of a user, and our scheme also provides privacy from overly curious friends.

To summarize, our PPLSS using the proposed ORE scheme can distinguish itself from existing solutions in that it (1) provides secure location privacy by not disclosing any location information about users and queries, not even approximate location information, to a database server, (2) does not require any third party, (3) achieves low communication and computational overhead by not requiring any direct communication between users or multiple-round communication between a user and a database server, (4) designs an index structure for our ORE scheme to improve query processing efficiency, (4) supports highly dynamic location updates from individual users efficiently, and (5) introduces a new privacy notion, called a personalized privacy region, to further improve user privacy within a group of friends.

10 Conclusion

In this paper, we introduce an *Order-Retrievable Encryption* (ORE) scheme; a new encryption notion for *Privacy-Preserving Location Sharing Services* (PPLSS) in social networking applications. ORE is designed to answer location queries that allow a user to view the exact location of his/her friends within a user-specified distance without revealing any location information about the user and his/her friends to the database server and any other users in the system. The distinguishing characteristics of ORE compared to existing algorithms are that ORE provides secure location privacy, achieves low communication and computational cost, and supports dynamic location updates. To improve query processing efficiency, we propose a tree-like index structure for our ORE scheme (ORE-Index) to facilitate range searches over the encrypted locations of a group of friends. In addition, a personalized privacy region scheme is proposed to further improve user privacy within a group of friends by enabling a user to specify a maximum distance up to which his/her friends are allowed to locate the user. We also perform experiments to evaluate ORE and ORE-Index and show that their performance is much better compared to the state-of-the-art cryptography-based technique designed for spatial queries.

References

[1] Facebook Places, “http://www.facebook.com/places/.”

[2] Foursquare, “http://www.foursquare.com.”

[3] Google Plus, “https://plus.google.com.”

[4] Loopt, “http://www.loopt.com.”

[5] L. Barkhuus, B. Brown, M. Bell, S. Sherwood, M. Hall, and M. Chalmers, “From awareness to repartee: Sharing location within social groups,” in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2008.

[6] E. Toch et al., “Empirical models of privacy in location sharing,” in *Proceedings of the ACM International Conference on Ubiquitous Computing*, 2010.

[7] S. Consolvo et al., “Location disclosure to social relations: Why, when, & what people want to share,” in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2005.

[8] C.-Y. Chow, M. F. Mokbel, and W. G. Aref, “Casper*: Query processing for location services without compromising privacy,” *ACM Transactions on Database Systems*, vol. 34, no. 4, pp. 1–48, 2009.

[9] M. Gruteser and D. Grunwald, “Anonymous usage of location-based services through spatial and temporal cloaking,” in *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services*, 2003.

[10] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, “The new casper: Query processing for location services withoutcompromising privacy,” in *Proceedings of the International Conference on Very Large Data Bases*, 2006.

[11] T. Wang and L. Liu, “Privacy-aware mobile services over road networks,” in *Proceedings of the International Conference on Very Large Data Bases*, 2009.

[12] L. Siksnys, J. R. Thomsen, S. Saltenis, and M. L. Yiu, “Private and flexible proximity detection in mobile social networks,” in *Proceedings of the International Conference on Mobile Data Management*, 2010.

[13] L. Siksnys, J. R. Thomsen, S. Saltenis, M. L. Yiu, and O. Andersen, “A location privacy aware friend locator,” in *Proceedings of the International Symposium on Spatial and Temporal Databases*, 2009.

[14] S. Mascetti, C. Bettini, and D. Freni, “Longitude: Centralized privacy-preserving computation of users’ proximity,” in *the International Workshop on Secure Data Management*, 2009.

[15] S. Triukose, S. Ardon, A. Mahanti, and A. Seth, “Geolocating ip addresses in cellular data networks,” in *Passive and Active Measurement*, ser. Lecture Notes in Computer Science, 2012, vol. 7192, pp. 158–167.

[16] M. L. Yiu, G. Ghinita, C. S. Jensen, and P. Kalnis, “Enabling search services on outsourced private spatial data,” *The International Journal on Very Large Data Bases*, vol. 19, no. 3, pp. 363–384, 2010.

[17] O. Goldreich, *Foundations of Cryptography, volume I, Basic Tools*. Cambridge University Press, 2007.
[18] B. Kaliski, "TWIRL and RSA key size," 2003, CryptoBytes Technical Newsletter, http://www.rsa.com/rsalabs/node.asp?id=2004.
[19] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proceedings of the ACM International Conference on Management of Data*, 2009.
[20] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order-preserving encryption for numeric data," in *Proceedings of the ACM International Conference on Management of Data*, 2004.
[21] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Eurocrypt*, 2009.
[22] "Specification for the advanced encryption standard (AES)," Federal Information Processing Standards Publication 197, 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.
[23] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003.
[24] IEEE, *P1363 - 2000: Standard Specifications For Public Key Cryptography*, 2000.
[25] Google Latitude, "http://www.google.com/latitude."
[26] Facebook Statistics, "http://www.facebook.com/press/info.php?statistics."
[27] S. Chen, C. S. Jensen, and D. Lin, "A benchmark for evaluating moving object indexes," *Proceedings of the International Conference on Very Large Data Bases*, 2008.
[28] P. Barford and V. Yegneswaran, *An Inside Look at Botnets*. Springer, 2007, pp. 171–191.
[29] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private queries in location based services: Anonymizers are not necessary," in *Proceedings of the ACM International Conference on Management of Data*, 2008.
[30] M. Kohlweiss et al., "Efficient oblivious augmented maps: Location-based services with a payment broker," in *Proceedings of the Privacy Enhancing Technologies Symposium*, 2007.
[31] R. Vishwanathan and Y. Huang, "A two-level protocol to answer private location-based queries," in *IEEE International Conferences on Intelligence and Security Informatics*, 2009.
[32] H. Hacigümüs, B. R. Iyer, and S. Mehrotra, "Efficient execution of aggregation queries over encrypted relational databases," in *Proceedings of the International Conference on Database Systems for Advanced Applications*, 2004.
[33] E. Mykletun and G. Tsudik, "Aggregation queries in the database-as-a-service model," in *Proceedings of the Annual IFIP Conference on Data and Applications Security*, 2006.
[34] K. Chen and L. Liu, "Privacy preserving data classification with rotation perturbation," in *Proceedings of the IEEE International Conference on Data Mining*, 2005.
[35] S. R. M. Oliveira and O. R. Zaane, "Achieving privacy preservation when sharing data for clustering," in *Proceedings of the SIAM International Conference on Data Mining*, 2004.
[36] K. Liu, C. Giannella, and H. Kargupta, "An attacker's view of distance preserving maps for privacy preserving data mining," in *Proceedings of the European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2006.
[37] G. Zhong, I. Goldberg, and U. Hengartner, "Louis, lester and pierre: Three protocols for location privacy," in *Proceedings of the Privacy Enhancing Technologies Symposium*, 2007.
[38] M. Herrmann, A. Rial, C. Diaz, and B. Preneel, "Practical privacy-preserving location-sharing based services with aggregate statistics," in *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks*, 2014, pp. 87–98.
[39] J. Freudiger, R. Neu, and J.-P. Hubaux, "Private sharing of user location over online social networks," in *HotPETs*, 2010.

Roman Schlegel has an MSc from EPFL in Switzerland in Communication Systems and a PhD in Computer Science from City University in Hong Kong. During his doctoral studies he also spent a year as a research assistant at Indiana University Bloomington in the US. After finishing his PhD he joined ABB Corporate Research as a research scientist for security in industrial control systems. His research interests include privacy, network security and applied cryptography.

Chi-Yin Chow received the M.S. and Ph.D. degrees from the University of Minnesota-Twin Cities in 2008 and 2010, respectively. He is currently an assistant professor in Department of Computer Science, City University of Hong Kong. His research interests include spatio-temporal data management and analysis, GIS, mobile computing, and location-based services. He was the co-organizer of ACM SIGSPATIAL MobiGIS 2012 and 2013.

Qiong Huang got his B.S. and M.S. degrees from Fudan University in 2003 and 2006 respectively, and got his PhD degree from City University of Hong Kong in 2010. Now he is a professor at South China Agricultural University. His research interests include cryptography and information security, in particular, cryptographic protocols design and analysis.

Duncan S. Wong received the B.Eng. degree from the University of Hong Kong in 1994, the M.Phil. degree from the Chinese University of Hong Kong in 1998, and the Ph.D. degree from Northeastern University, Boston, MA, in 2002. He is currently an associate professor in the Department of Computer Science at the City University of Hong Kong. His primary research interest is cryptography; in particular, cryptographic protocols, encryption and signature schemes, and anonymous systems.