Privacy-Preserving Ridesharing Recommendation in Geosocial Networks

Chengcheng Dai[1], Xingliang Yuan[1,2], and Cong Wang[1,2](✉)

[1] Department of Computer Science, City University of Hong Kong, Hong Kong, China
{cc.dai,xl.y}@my.cityu.edu.hk

[2] City University of Hong Kong Shenzhen Research Institute, Shenzhen, China
congwang@cityu.edu.hk

Abstract. Geosocial networks have received a lot of attentions recently and enabled many promising applications, especially the on-demand transportation services that are increasingly embraced by millions of mobile users. Despite the well understood benefits, such services also raise unique security and privacy issues that are currently not very well investigated. In this paper, we focus on the trending ridesharing recommendation service in geosocial networks, and propose a new privacy-preserving framework with salient features to both users and recommendation service providers. In particular, the proposed framework is able to recommend whether and where the users should wait to rideshare in given geosocial networks, while preserving user privacy. Meanwhile, it also protects the proprietary data of recommendation service providers from any unauthorised access, such as data breach incidents. These privacy-preserving features make the proposed framework especially suitable when the recommendation service backend is to be outsourced at public cloud for improved service scalability. On the technical front, we first use kernel density estimation to model destination distributions of taxi trips for each cluster of the underlying road network, denoted as cluster arrival patterns. Then we utilize searchable encryption to carefully protect all the proprietary data so as to allow authorised users to retrieve encrypted patterns with secure requests. Given retrieved patterns, the user can safely compute the potential of ridesharing by investigating the probabilities of possible destinations from ridesharing requirements. Experimental results show both the effectiveness of the proposed recommendation algorithm comparing to the naive "wait-at-where-you-are" strategy, and the efficiency of the utilized privacy-preserving techniques.

1 Introduction

With the proliferation of smartphones, geosocial networks are gaining increasing popularity for utilizing user-provided location data to match users with a place, event, or person relevant to their interests, and to enable further socialization activities based on such information [1]. Based on geosocial networks, many

H.T. Nguyen and V. Snasel (Eds.): CSoNet 2016, LNCS 9795, pp. 193–205, 2016.
DOI: 10.1007/978-3-319-42345-6_17

promising on-demand transportation services have been enabled and increasingly embraced by millions of mobile users. Among them, the trending ridesharing recommendation service has received lots of attentions, where users can submit their sources and destinations to find other users that match their trips. Ridesharing is particularly important for new on-demand transportation services such as Uber and DiDi. Besides saving money for users, ridesharing also brings potential to assuage traffic congestion and save energy consumption.

Despite the well understood benefits, such new services also raise unique security and privacy issues that are currently not very well investigated. Take the ridesharing recommendation service for example. Its recommendation mechanism depends on whether another passenger with the similar source and destination can appear in time. Thus, it would inevitably demand constant access to the users' current locations and/or their intended destinations. Such personal location data usually contain sensitive information, and should be always protected, as well recognised in the literature [11]. Besides, from the ridesharing service provider's perspective, the proprietary datasets, such as the recommendation algorithms, and valuable score functions learned from mass of data with data mining or machine learning techniques, are extremely valuable as well. They are crucial digital assets to the recommendation services and should also be strictly protected against any unauthorised access, especially given the rising concerns of recent data breach incidents [3]. Considering more and more emerging geosocial applications are directly hosted at commercial public cloud that are not necessarily within the trust domain of service providers, the security threats on unauthorised access of such proprietary information are further exacerbated.

In light of these observations, in this paper we propose a privacy-preserving framework for secure ridesharing recommendation with salient features to both users and recommendation service providers. In particular, the proposed framework is able to recommend whether and where the users should wait to rideshare in given geosocial networks, while preserving user privacy. The user privacy assurance hinges on the fact that all the data that leave from and arrive at the user's mobile devices are encrypted. Meanwhile, it also ensures that all the proprietary data from the recommendation service provider will always be encrypted during the service operations, and will never be exposed to any unauthorised users. These privacy-preserving features make the proposed framework resilient to data breach incidents at the service backend. They are also attractive when the recommendation service backend is to be outsourced at public cloud for improved service scalability, e.g., to satisfy the throughput and response time requirement for real-time queries.

In our framework, to deliver good ridesharing recommendation services, we exploit the fact that pick-ups and drop-offs of users' daily trips usually follow certain patterns. With the observation that user Alice may walk to some place nearby or change her destination to a Point-of-Interest (POI) nearby to increase her chance to rideshare, we investigate the probability for taxis to depart from somewhere near Alice's source towards somewhere near her destination. In particular, we first fragment the underlying road network into a number of road clusters, and then model destination distributions of taxi trips for each cluster, denoted as cluster arrival patterns, with kernel density estimation fusioned with departure probabilities for expected higher user satisfaction, as explained in Sect. 3. Based on these cluster arrival patterns, then we utilize off-the-shelf searchable encryption technique to carefully protect all the proprietary data so as to allow authorised users to retrieve encrypted patterns with secure requests. These patterns are always encrypted and stored on the cloud server while answering for authorised on-demand encrypted requests from mobile users.

The operation of our proposed framework starts from the client application on the user's smartphone. Given possible waiting places and destinations of a user, a secure query will be generated at the user client application, and then submitted to the cloud server. Subsequently, the server securely searches over encrypted patterns without decryption and returns encrypted result patterns. During this procedure, the privacy of both patterns and requested cluster ids (i.e., user source and destination information) are well-preserved. After decryption, the client application computes the ridesharing probability based on the patterns. If the potential to rideshare with others is not high enough for all nearby clusters, Alice is recommended to take a taxi directly. Otherwise, the client application highlights where to wait on the map for Alice. Thus in either cases, Alice can save either time or money.

The main contributions are summarized as follows: 1. We design a privacy-preserving recommendation framework to securely help users decide whether and where to wait for ridesharing. It also protects service provider's proprietary data from unauthorised users during operations. 2. Experimental results show the efficiency of the privacy-preserving techniques, and the effectiveness of the recommendation comparing to the naive "wait-at-where-you-are" strategy.

The rest of this paper is organized as follows. Section 2 states the system architecture, and Sect. 3 delineates the proposed privacy-preserving recommendation scheme. Section 4 gives the security analysis of the proposed scheme. Section 5 analyzes the performance. Section 6 discusses the related work. Finally, Sect. 7 concludes this paper.

2 System Model

As shown in Fig. 1, the architecture consists of three different parties: the *service provider*, the *user* and the *cloud server*. Service provider learns patterns with data mining or machine learning techniques, and encrypts these patterns before outsourcing them to the cloud. Users generate encrypted queries for certain patterns according to their ridesharing requests. Cloud server sends encrypted patterns to users in an "on-demand" manner. To enable search over encrypted pattern, searchable symmetric encryption (*SSE*) is utilized to securely index encrypted patterns. A secure pattern index will be uploaded as well.

Users. We consider authorised users with registration as prior work [5,18]. There is no malicious user that either shares her key with others or generates unnecessary queries to steal information from the server. As a client application on a

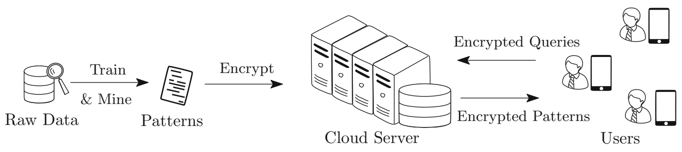

Fig. 1. Our proposed system model

user's smartphone, the city map is in the storage of the client application. When a user submits a query, the client application on the user's smartphone generates a secure search request to the cloud server. After receiving the encrypted patterns from the cloud server, the client application computes the ridesharing potential after decryption. If the user is recommended to rideshare, it will highlight the corresponding road for each recommended cluster on the map.

Besides, the user specifies her willingness in the preference setting of the client application, namely the maximum walking distance d_s to a new place from her source, the maximum walking distance d_e after she leaves the taxi to her own destination and maximum waiting time t_w at the new place for ridesharing.

Cloud Server. Sensitive patterns are encrypted and indexed before storing on a cloud server. The server is deployed in the cloud to provide the privacy-preserving recommendation service for a large number of real-time queries. In this paper, we consider a "honest-but-curious" cloud server, i.e., the server acts in an "honest" fashion, but is "curious" to infer and analyze the message flow to learn additional information on the user request and the pattern information.

Problem Definition. In our recommendation application, the user specifies her query as $Q = (ID, timestamp, l_s, l_d)$, where *id* is user id, *timestamp* is when the query is submitted, l_s and l_d are respectively the source and the destination of the user. Given a query, we compute the potential of ridesharing and where the user should wait based on ridesharing requirements. Alice can rideshare with another passenger Bob if (i) the source of Bob is within her maximum walking distance d_s from her source l_s (ii) the destination of Bob is somewhere within her maximum walking distance d_e from her destination l_d (iii) Bob submits his request within waiting time t_w. Recall that d_s, d_e and t_w are set as their ridesharing willingness in the client application. For example, Alice can increase her chance of ridesharing by increasing her waiting time t_w.

When a user submits a query, the client application generates a search request according to ridesharing conditions. The server returns encrypted patterns to the client application. To allow an authorised group of users to search through the patterns and prevent unauthorised access, the server cannot infer any sensitive information of patterns from the encrypted storage before search and can only learn the limited information about the requested patterns and the results.

3 Our Proposed Design

In this section, we discuss how to perform privacy-preserving ridesharing recommendation with a third-party cloud server. To initialize the service, the service provider distributes search request generation keys to authorised users. We here assume the authorisation between the client application and the user is appropriately done on the smartphone. We discuss how to learn patterns (discussed in Sect. 3.1) and make recommendation (discussed in Sect. 3.2) with machine learning techniques, and perform ridesharing recommendation in a privacy-preserving way (discussed in Sect. 3.3).

3.1 Learning Patterns

Road Segment Clustering. Since modeling destinations of trips based on single road is too dynamic, the underlying road network is grouped into road clusters by applying k-means to the mid-points of road segments [16]. The cluster of the mid-point is the cluster that the road segment belongs to, recorded as $c_i = \{r_1, \ldots, r_N\}$. A grid index structure is built on the underlying road network. Given a location (lon, lat), we can find out the road on which the location is located and further get the cluster it belongs to. We category trip records into groups according to which clusters their sources belong to. Each cluster makes use of the corresponding group of records as samples to derive the kernel density estimator about the probability for other passengers (or taxis) to depart from somewhere in the cluster and have a given location (lon, lat) as their destination.

Kernel Density Estimation. For each cluster, given a trip record (t_i, $source_i$, $destination_i$), we describe a training sample in the format $\mathbf{x_i} = (lon_i, lat_i, t_i)^T$, which is a 3-dimensional column vector with the longitude (lon_i) and latitude (lat_i) and the time (t_i), indicating a trip from somewhere in the cluster to $desination_i$ (lon_i, lat_i) happens at time t_i[1]. Intuitively ridesharing is related to when the query is submitted since traffic directions in modern cities depend on time. For example, traffics are likely to be from residencies to companies in the morning and right way around in the evening. Thus pick-up time t_i is taken into consideration.

Let $X_c = <\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n>$ be the samples for a certain cluster c that follows an unknown density p. As described in [13], its kernel density estimator over X_c for a new sample $\mathbf{x}$ is given by: $p(\mathbf{x}) = \frac{1}{nh^d}\sum_{i=1}^{n} K(\frac{\mathbf{x}-\mathbf{x}_i}{h})$. $K(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d}} e^{-\frac{1}{2}\|\mathbf{x}\|^2}$. h is a smoothing parameter and $K(.)$ is the widely used Gaussian kernel. With $d = 3$, we can obtain the probability to have a taxi that departs from a certain cluster c towards new location (lon, lat) at certain time as follows:

$$p(\mathbf{x}_{new}|X_c = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}) = \sum_{i=1}^{n} \frac{1}{n} \frac{1}{(\sqrt{2\pi}h)^3} e^{-\frac{1}{2h^2}\|\mathbf{x}_{new}-\mathbf{x}_i\|^2} \tag{1}$$

Equation 1 is equivalent to $\sum_{i}^{n} \frac{1}{n}\mathcal{N}(\mathbf{x}_{new}|\mathbf{x}_i, h^2\mathbf{I})$. The optimal smoothing parameter h [13] is $0.969n^{-\frac{1}{7}}\sqrt{\frac{1}{3}\boldsymbol{\sigma}^T\boldsymbol{\sigma}}$, where $\boldsymbol{\sigma}$ is the marginal standard deviation

[1] Instead of transforming the original pick-up time t_i into discrete values between 1 and 48 [6], we transform t_i to continuous values to keep more details about the time domain. Please refer to the experiment section for more details.

vector of values in X_c. Both samples $\mathbf{x}_1$, $\mathbf{x}_2, \ldots, \mathbf{x}_n$ in X_c and the smoothing parameter h are required to describe the kernel density estimator of each cluster.

Fusion with Departure Probability. For each cluster, the kernel density estimator describes the probability for taxis to depart from somewhere in it and have a given location (lon, lat) as their destination. However, it didn't consider the departure probabilities of different roads in the cluster. We distinguish each road with the departure probability, i.e., the probability to have a taxi departing from certain road, for more accurate prediction. Given $\mathbf{x}_{new} = (lon, lat, t)^T$, the probability $P(r_j \rightarrow \mathbf{x}_{new})$ to have a taxi departing at time t from road r_j in cluster c_i and towards destination (lon, lat) is:

$$P(r_j \rightarrow \mathbf{x}_{new}) = p(A_{r_j}) * p(\mathbf{x}_{new}) \tag{2}$$

where $p(A_{r_j})$ is the probability to have a taxi pick up a passenger at road r_j, i.e.,

$$p(A_{r_j}) = \frac{N_{r_j}}{\sum_{r_x \in c_i} N_{r_x}} \tag{3}$$

where N_{r_j} denotes the number of pick-ups on road r_j. Noted that $p(A_{r_j})$ does not incur any additional computation since it is obtained when we category records into groups according to which clusters their sources belong to. $p(\mathbf{x}_{new})$ is computed by Eq. 1.

3.2 Recommendation with Patterns

After learning patterns, and before talking about how to perform ridesharing recommendation in a privacy-preserving way, we describe how the client application can compute the ridesharing potential based on patterns to make recommendation. Specifically, given a ridesharing query $Q = (ID, \mathit{timestamp}, l_s, l_d)$, the client application performs network expansion technique [12] to get all the roads $R = \{r_1, \ldots, r_d\}$ that are within distance d_s from l_s. With the road clustering information in the client application, i.e., which road belongs to which cluster, we can easily group roads in R to candidate clusters. Denote the candidate set as $L = \{C_1, C_2, \ldots, C_x\}$ and $C_i = \{r_y | r_y \in R \wedge r_y \in c_i\}$. Cluster information, i.e., the kernel density estimator and departure probabilities of roads, are required to compute the ridesharing potential for each $C_i \in L$.

For roads that are reachable within d_e from l_d, which are obtained with the same technique, we generate $\mathbf{x}_{new_k} = (r_k^{lon}, r_k^{lat}, t)^T$ where (r_k^{lon}, r_k^{lat}) is the midpoint of road r_j and t is the time by transforming *timestamp* in the query Q in the same way as records. Denote the sample set as D' and the probability to have taxis departing from roads in C_i toward roads in D as $P(C_i \rightarrow D)$, we have $P(C_i \rightarrow D) = \sum_{\mathbf{x}_{new_k} \in D'} P(C_i \rightarrow \mathbf{x}_{new_k})$, where $P(C_i \rightarrow \mathbf{x}_{new_k})$ is the probability for taxis to depart from any road $r_j \in C_i$, i.e., within distance d_s from l_s at time t towards somewhere on road r_k. By combining with Eq. 2, we have $P(C_i \rightarrow \mathbf{x}_{new_k}) = \sum_{r_j \in C_i} P(r_j \rightarrow \mathbf{x}_{new_k}) = p(\mathbf{x}_{new_k}) * \sum_{r_j \in C_i} p(A_{r_j})$[2].

[2] All roads in C_i share the same kernel density estimator and thus the same $P(\mathbf{x}_{new_k})$.

Since $\sum_{r_j \in C_i} P(A_{r_j})$ is not relative to $\mathbf{x}_{new_k}$, we further have

$$P(C_i \to D) = \sum_{\mathbf{x}_{new_k} \in D'} p(\mathbf{x}_{new_k}) * \sum_{r_j \in C_i} p(A_{r_j}) = \sum_{r_j \in C_i} p(A_{r_j}) * \sum_{\mathbf{x}_{new_k} \in D'} p(\mathbf{x}_{new_k}).$$

Noted that if no pick-ups exist on any the road in C_i, i.e., $\sum_{r_j \in C_i} P(A_{r_j}) = 0$, there is no need to further compute $p(\mathbf{x}_{new_k})$ by plugging in different $\mathbf{x}_{new}$ in Eq. 1. We display the computation of $P(C_i \to D)$ in Algorithm 1.

Algorithm 1. Ridesharing potential of cluster C_i

```
1 foreach $r_j \in C_i$ do
2     $P_1$ += p($A_{r_j}$);//Compute $p(A_{r_j})$ with Eq. 3;
3 if $P_1$ is not 0 then
4     $P_2 = 0$;
5     foreach $r_k \in D$ do
6         $\mathbf{x}_{new_k} = (r_j^{lon}, r_j^{lat}, t)$;
7         $P_2$ += $p(\mathbf{x}_{new_k})$;//Compute $p(\mathbf{x}_{new_k})$ with Eq. 1;
8     Return $P_1$ * $P_2$;
```

Only candidates with ridesharing potential $P(C_i \to D)$ greater than a threshold are considered valid for recommendation. If no valid cluster exists for Q, the user will be suggested not to wait for ridesharing and take taxi directly. In case many road clusters satisfy the condition, we return top-k clusters according to the probabilities. The client application will highlight the corresponding roads $r_j \in R$ for each recommended cluster on the map, i.e., roads that are within distance d_s from l_s.

3.3 Privacy-Preserving Ridesharing Recommendation

Given the discussion on learning patterns and making recommendation with patterns, we describe the overall privacy-preserving ridesharing recommendation. We organize the patterns as follows. For $Pattern_i$ of each cluster on the cloud server, it records samples and the smoothing parameter of the kernel density estimator, and the departure probabilities[3]. *SSE* is utilized on the server side to keep sensitive patterns confidential, while resuming the ability to selectively retrieve encrypted patterns. A *SSE* scheme is a collection of four polynomial-time algorithms (Kengen, BuildIndex, Trapdoor, Search) such that: (i) Keygen(1^k): outputs symmetric key K. (ii) BuildIndex(K, $\mathcal{D}$): outputs a secure index I built on encrypted patterns $\mathcal{D}$ that helps the server to search without decryption. (iii) Trapdoor(K, w): outputs a trapdoor T_w. Cluster id w is associated with a trapdoor which enables server to search while keeping w hidden. (iv) Search(I, T_w):

[3] Departure probabilities are ordered according to road ids in ascending order. Suppose in $Pattern_1$ of cluster c_1 there are three probabilities 0.3, 0, 0.7 and $c_1 = \{r_1, r_3, r_4\}$. We get p(A_{r_1}) = 0.3, p(A_{r_3}) = 0 and p(A_{r_4}) = 0.7.

outputs the identifier of the pattern of cluster w. Noted that Kengen, BuildIndex, Trapdoor are run by the user, while Search is run by the server.

Let $Enc_s(\cdot)$, $Dec_s(\cdot)$ be semantic secure encryption and decryption functions based on symmetric key s. In addition, we make use of one pseudo-random function (PRF) $f : \{0,1\}^* \times key \rightarrow \{0,1\}^l$ and two pseudo-random permutations (PRP) $\pi : \{0,1\}^* \times key \rightarrow \{0,1\}^*$ and $\psi : \{0,1\}^* \times key \rightarrow \{0,1\}^*$. We are now ready for the details of the privacy-preserving ridesharing recommendation.

Generating Key. Generate random keys x, y and z where $x, y, z \overset{R}{\leftarrow} \{0,1\}^{k'}$ and output $K = (x, y, z, s)$.

Building a Secure Index. The secure index I is a look-up table, which contains information that enables one to locate the pattern of certain cluster c_i. Each entry corresponds to a cluster c_i and consists of a pair $\langle address, addr(Pattern_i) \oplus f_y(id_i)\rangle$. id_i is the id of cluster c_i. $Pattern_i$ is the pattern of cluster c_i. The address of $Pattern_i$, i.e., $addr(Pattern_i)$, is set to $\psi_x(id_i)$, which means that the location of a pattern is permutated and protected. $addr(Pattern_i) \oplus f_y(id_i)$ indicates that the address of $Pattern_i$ is encrypted using the output of a PRF $f_y(.)$. The other field, $address$[4], is used to locate an entry in the look-up table. We set $I[\pi_z(id_i)] = \langle addr(Pattern_i) \oplus f_y(id_i)\rangle$.

After building the index I, $Enc_s(Pattern_i)$ is performed for each pattern. Both the secure index and encrypted patterns are outsourced to the cloud server. Noted that we pad $Enc_s(Pattern_i)$ to the same length to prevent leaking the length information. Table 1 indicates the storage on the cloud server.

Table 1. Encrypted storage on the cloud server

Address	Key		Samples	h	Departure Prob.
$\pi_z(id_k)$	$addr(Pattern_k) \oplus f_y(id_k)$	$\rightarrow$	$Enc_s(\mathbf{x}_a)$, $\forall x_a$	$Enc_s(h_k)$	$Enc_s(p(A_{r_d}))$, $\forall r_d$
...	...	$\rightarrow$	...	...	...
$\pi_z(id_n)$	$addr(Pattern_n) \oplus f_y(id_n)$	$\rightarrow$	$Enc_s(\mathbf{x}_b)$, $\forall x_b$	$Enc_s(h_n)$	$Enc_s(p(A_{r_e}))$, $\forall r_e$
...	...	$\rightarrow$	...	...	...
$\pi_z(id_1)$	$addr(Pattern_1) \oplus f_y(id_1)$	$\rightarrow$	$Enc_s(\mathbf{x}_c)$, $\forall x_c$	$Enc_s(h_1)$	$Enc_s(p(A_{r_f}))$, $\forall r_f$

Trapdoor Construction. Output $T_w = (address, key)$, where $address = \pi_z(id_i)$, $key = f_y(id_i)$ and id_i is the requested cluster id.

Searching. With the trapdoor $T_w = (address, key)$, the server retrieves $\theta = I[address]$ and uses key to decrypt θ. Let $\langle addr(Pattern_i)\rangle = \theta \oplus key$. With the address of $Pattern_i$, the server sends the encrypted pattern of cluster c_i to the user. For each received pattern, the user performs $Dec_s(Pattern_i)$.

Multi-user *SSE*. To allow an arbitrary group of users submit queries to search the data on the cloud server, we combine single-user *SSE* with broadcast encryption to achieve multi-user *SSE* [5]. We assume the pre-sharing of the trapdoor

[4] We manage *address* with indirect addressing [5] to provide efficient storage and access of sparse tables.

generation key between the service provider and users. Adding/revoking users can be properly done via broadcast encryption. An authorised user applies a PRP ϕ (keyed with a secret key r) on a regular single-user trapdoor T_w. Upon receiving $\phi_r(T_w)$, the server recovers the trapdoor by computing $\phi_r^{-1}(\phi_r(T_w))$. Unauthorised users cannot get the valid r to yield a valid trapdoor for searching.

On behalf of an authorised user, the client application generates a search request T_w for each required cluster via a certain one-way function with the trapdoor generation key. We have $T_w = (\phi_r(T_{w_1}), \phi_r(T_{w_2}), \ldots, \phi_r(T_{w_x}))$, where $T_{w_i} = (\pi_z(id_i), f_y(id_i))$ and id_i is the id of cluster c_i. After construction, T_w is submitted to the cloud server. Given T_w, the server recovers $T_{w_i} = (\pi_z(id_i), f_y(id_i))$ with key r and preforms searching. In this way, the server searches over the stored data without decryption, and sends back required cluster patterns, i.e., $Pattern_1, Pattern_2, \ldots, Pattern_x$. Noted that the server is not aware of which cluster is requested. After receiving the required patterns, the client application decrypts them on behalf of an authorised user and computes the ridesharing potential of each cluster C_i as shown in Algorithm 1. Recommendations about whether and where the user should wait for ridesharing are made as shown in Sect. 3.2.

4 Security Analysis

We follow the security definition in searchable symmetric encryption [5] that nothing beyond the encrypted outcome and the repeated search queries should be leaked from the remote storage. We adapt the simulation-based security model of [5] and prove non-adaptive semantic security is guaranteed. We first introduce notions used in [5]: 1. *History:* an interaction between the user and the cloud server, determined by a collection $\mathcal{C}$ of cluster patterns and a set of cluster ids searched by the user, denoted as $H = (\mathcal{C},\ id_1,\ id_2,\ \ldots,\ id_x)$. 2. *View:* what the cloud server can see given a history H, denoted as $V(H)$, including the index I of $\mathcal{C}$, the trapdoors of the queried cluster ids $\{T_{id_1}, T_{id_2}, \ldots, T_{id_x}\}$, and the encrypted collection of $\mathcal{C}$. 3. *Trace:* what the cloud server can capture, denoted as $Tr(H)$, including the size of the encrypted patterns, the outcome of each search (i.e., the patterns $Pattern_i$) and whether two searches were performed for the same cluster id or not.

Given two histories with the identical trace, the cloud server cannot distinguish the views of the two histories. Our mechanism is secure since the cloud server cannot extract additional knowledge beyond the trace, which we are willing to leak. We can describe a simulator S such that, given trace $Tr(H)$, it can simulate a view V^* (composed of encrypted patterns, index and trapdoors) indistinguishable from the cloud server's view $V(H)$ [5]. In particular, the simulated encrypted pattern is indistinguishable due to the semantic security of the symmetric encryption. The indistinguishability of index and trapdoors is based on the indistinguishability of pseudo-random function output and a random string.

5 Experiments

Dataset. We make use of the Uber trip data of NYC[5]. Each record is in the format (t, $source$, $destination$), where t is the pick-up time, $source$ and $destination$ are respectively pick-up location and drop-off location, described as (lon, lat). We transform the pick-up time from the original format $hh{:}mm{:}ss$ to $(hh*3600 + mm*60 + ss)/(24*3600)$ in preprocessing. We randomly select 1,000 records as ridesharing queries. The time, source and destination of the trip are treated as $timestamp$, l_s and l_d in the queries.

Effectiveness Evaluation. We compare our ridesharing recommendation (RR) with the naive strategy to "wait at where your are" (WW). In WW users wait for ridesharing at where they are, i.e., the cluster that l_s is in. To evaluate the effectiveness of recommendation, we category ridesharing recommendation into two types, namely *to-rideshare* and *not-to-rideshare*. An accurate *to-rideshare* means that users can rideshare with others at the recommended locations. An accurate *not-to-rideshare* indicates that users are recommended not to wait for ridesharing and there are indeed no others to satisfy the ridesharing requirements. We consider the following measurements. (i) Ridesharing successful ratio ($RSRatio$). We measure the ratio of successful ridesharing of both RR and WW by $RSRatio$, defined as $RSRatio = \frac{\#\text{ accurate } to\text{-}rideshare}{\#\ to\text{-}rideshare}$. (ii) Prediction accuracy ($Accuracy$). We measure the accuracy of predicting whether the user should wait for ridesharing or not by $accuracy$, defined as $Accuracy = \frac{\#\text{ accurate } not\text{-}to\text{-}rideshare + \#\text{ accurate } to\text{-}rideshare}{\#\text{ queries}}$. (iii) Recommendation quality[6]. To find out how many clusters that actually have others to rideshare a query are discovered by our framework, we employ standard metrics, i.e., *precision* and *recall*: $precision = \frac{\#\text{ discovered clusters}}{\text{k}}$, $recall = \frac{\#\text{ discovered clusters}}{\#\text{ positive clusters}}$. Positive clusters are clusters with others to rideshare a query. Discovered clusters are the positive clusters in the recommended clusters. *Precision* and *recall* are averaged over all queries.

Table 2. Effect of waiting time t_w (seconds)

Metrics	RR					WW				
	150	300	450	600	750	150	300	450	600	750
RSRatio	0.218	0.255	0.309	0.327	0.364	0.061	0.078	0.082	0.091	0.124
Accuracy	0.535	0.543	0.567	0.574	0.588	0.061	0.078	0.082	0.091	0.124
Precision	0.475	0.478	0.544	0.556	0.586	-	-	-	-	-
Recall	0.618	0.658	0.686	0.694	0.712	-	-	-	-	-

[5] https://github.com/fivethirtyeight/uber-tlc-foil-response. Destinations are generated based on a check-in dataset of Foursquare from http://download.csdn.net.

[6] We didn't study precision and recall of WW since users wait at where they are.

Table 3. Execution time

Learning patterns	enc.pattern	SSE.index	SSE.token	SSE.search	Recommendation
263.37 s	14.12 s	40.34 ms	117.95 µs	24.60 µs	0.269 s

Table 2 shows the influence of waiting time t_w on the performances of RR and WW. The default values of maximum walking distances d_s and d_e are set to 500 m. The number of recommended clusters k is set to 5. As t_w increases, users can wait for ridesharing for a longer time. *RSRatio* and *Accuracy* increase for RR and WW. As more passengers appear during a longer t_w, the potential to rideshare increases. Both the number of discovered clusters and the number of positive clusters increase, leading to the increase in *precision* and *recall*.

Performance Evaluation. Experiments were performed on a Windows machine with Intel i7-3770 CPU and 16 GB RAM. We measure the time of learning patterns (Learning patterns), the time to make recommendation after obtaining patterns from the server (Recommendation). To study the efficiency of *SSE*, we measure the average execution time to generate indexes (SSE.index), generate trapdoors per query (SSE.token), and execute search operations per query (SSE.search), as well as the time to encrypt pattern (enc.pattern) before outsourcing to a server. We have 1,000 patterns with the size of 106MB. As shown in Table 3, the major cost is introduced in the learning pattern stage and the encryption stage, which is acceptable because it is a one-time setup. The introduced security overhead (enc.pattern + SSE.index) is around 5 % to the cost of learning patterns which is also required in plaintext applications. Meanwhile, the time cost for processing each secure query (SSE.token + SSE.search) is small, and obtaining the potential for ridesharing from encrypted patterns is also efficient.

6 Related Works

Private Searching in Cloud. Public-key searchable encryption is usually adapted to deal with third-party data [2], where anyone with the public key can write to the data stored on the server but only users with the private key can search. For user-owned data, symmetrically encryption is widely adapted and client uploads additional encrypted data structures to help search, including oblivious RAMs [7] and searchable symmetric encryption (*SSE*) [5,8]. *SSE* is applicable for any forms of private retrieval based on keywords [4,15,17], varying from exact matching [5,8] to similarity search on textual files [15] or images [4]. Built on locality-sensitive hashing, similarity search is transformed to keyword search to handle millions of encrypted records [17]. Yet, the above studies do not focus on achieving privacy-preserving recommendation in geosocial networks.

Privacy-Preserving Recommender Systems. Privacy-preserving tests for proximity is proposed to test whether a friend is close to her without revealing location information of any of them [11]. To perform online behavioral advertising without compromising user privacy, an efficient cryptographic billing system is proposed so that the correct advertiser is billed without knowing which advertisement is displayed to the user [14]. Secure image-centric social discovery [18] is modeled as secure similarity retrieval of encrypted high-dimensional image profiles. Social media sites can recommend friends or social groups for users from public cloud without disclosing the encrypted image content.

Ridesharing. Current studies about ridesharing can be categorized into two scenarios, either drivers change their routes to pick up and drop off passengers [10] or users walk certain distance to get on a taxi and to their own destinations after getting off [9]. We consider ridesharing in geosocial networks as the second scenario to recommend users with the places that are most likely to have other users with similar trips. Unlike previous work [6] that only considers destinations of trips to model the pattern of each cluster, we fusion with departure probabilities of each road to distinguish each road with their pick-up probabilities for more accurate prediction. Noted that none of existing works [6,9,10] consider privacy issues in ridesharing, we preserve user privacy as well as protect the proprietary data of service providers from any unauthorised access.

7 Conclusion

In this paper, we proposed a privacy-preserving framework to recommend whether and where the users should wait to rideshare in geosocial networks. The privacy of both users and recommendation service providers are well protected. As future work, we plan to study how to enable the server to directly compute the results in the encrypted domain.

Acknowledgement. This work was supported in part by the Research Grants Council of Hong Kong (Project No. CityU 138513), and the Natural Science Foundation of China (Project No. 61572412).

References

1. Bao, J., Zheng, Y., Mokbel, M.F.: Location-based and preference-aware recommendation using sparse geo-social networking data. In: SIGSPATIAL, pp. 199–208 (2012)
2. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
3. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. In: NDSS (2015)
4. Cui, H., Yuan, X., Wang, C.: Harnessing encrypted data in cloud for secure and efficient image sharing from mobile devices. In: INFOCOM, pp. 2659–2667 (2015)
5. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: CCS, pp. 79–88 (2006)

6. Dai, C.: Ridesharing recommendation: whether and where should i wait? In: Cui, B., Zhang, N., Xu, J., Lian, X., Liu, D. (eds.) WAIM 2016. LNCS, vol. 9658, pp. 151–163. Springer, Heidelberg (2016). doi:10.1007/978-3-319-39937-9_12
7. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM **43**(3), 431–473 (1996)
8. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: CCS, pp. 965–976 (2012)
9. Ma, S., Wolfson, O.: Analysis and evaluation of the slugging form of ridesharing. In: SIGSPATIAL, pp. 64–73 (2013)
10. Ma, S., Zheng, Y., Wolfson, O.: Real-time city-scale taxi ridesharing. IEEE Trans. Knowl. Data Eng. **27**(7), 1782–1795 (2015)
11. Narayanan, A., Thiagarajan, N., Lakhani, M., Hamburg, M., Boneh, D.: Location privacy via private proximity testing. In: NDSS (2011)
12. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: VLDB, pp. 802–813 (2003)
13. Silverman, B.W.: Density Estimation for Statistics and Data Analysis, vol. 26. CRC Press, Boca Raton (1986)
14. Toubiana, V., Narayanan, A., Boneh, D., Nissenbaum, H., Barocas, S.: Adnostic: privacy preserving targeted advertising. In: NDSS (2010)
15. Wang, C., Ren, K., Yu, S., Urs, K.M.R.: Achieving usable and privacy-assured similarity search over outsourced cloud data. In: INFOCOM, pp. 451–459 (2012)
16. Wang, R., Chow, C., Lyu, Y., Lee, V.C.S., Kwong, S., Li, Y., Zeng, J.: TaxiRec: recommending road clusters to taxi drivers using ranking-based extreme learning machines. In: SIGSPATIAL, pp. 53:1–53:4 (2015)
17. Yuan, X., Cui, H., Wang, X., Wang, C.: Enabling privacy-assured similarity retrieval over millions of encrypted records. In: Pernul, G., Y A Ryan, P., Weippl, E. (eds.) ESORICS. LNCS, vol. 9327, pp. 40–60. Springer, Heidelberg (2015). doi: 10.1007/978-3-319-24177-7_3
18. Yuan, X., Wang, X., Wang, C., Squicciarini, A.C., Ren, K.: Enabling privacy-preserving image-centric social discovery. In: ICDCS, pp. 198–207 (2014)