Private Buddy Search: Enabling Private Spatial Queries in Social Networks

Ali Khoshgozaran
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
Email: jafkhosh@usc.edu

Cyrus Shahabi
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
Email: shahabi@usc.edu

***Abstract*—With the abundance of location-aware portable devices such as cellphones and PDAs, a new emerging application is to use this pervasive computing platform to learn about the whereabouts of one's friends and relatives. However, issues of trust, security and privacy have hindered the popularity and safety of the systems developed for this purpose. We identify and address the key challenges of enabling private spatial queries in social networks using an untrusted server model without compromising users' privacy. We propose Private Buddy Search (PBS), a framework to enable private evaluation of spatial queries predominantly used in social networks, without compromising sensitive information about its users. Utilizing server side encrypted index structures and client side query processing, PBS enjoys both scalability and privacy. Our extensive experimental evaluation shows that PBS supports very efficient user operations such as location updates, as well as spatial queries such as range and k-nearest neighbor search.**

I. Introduction

We are witnessing the emergence of a new killer-application at the crossroad of two popular paradigms: Location-Based Services (LBS) and Social Networking (SN). People make friends and create buddy lists in virtual worlds using social-networking sites such as MySpace (www.myspace.com) and then use their mobile devices to locate their virtual buddies in the real world. Enabling this emerging application, however, has serious privacy ramifications. The users of mobile devices may be willing to reveal their locations and/or profiles to their buddies but not to the LBS+SN server and other users.

Hence, the challenge is how to support buddy searches without revealing information to the server and other users in the system. To illustrate the difficulty of this challenge, let us consider two extreme solutions. One solution is to first encrypt and then store all the users' location and profile information at a centralized server. The advantage of this approach is that all the frequent updates to users' locations (and profile) will only be communicated to a single server. The disadvantage is that the server cannot support any querying/searching on the data efficiently because it is all encrypted. That is, for every client query, the entire database needs to be sent to the client for searching, hence, a high query overhead. The other extreme solution is to eliminate the central server altogether and push server's asks to the clients. The problem with this approach is that either every update to a user's location needs to be sent to all the members of the user's buddy list (push) or the user needs to communicate with all its buddies at the query time (pull). Some studies try to make this approach more practical by searching only the area around the user by wireless broadcasting [3]. While this approach can find user's buddies around him, it cannot answer whether the user has buddies beyond his cellphone's short range BlueTooth signals. Furthermore, the proposed techniques to protect location privacy in LBS [11], [12], [6], [15] do not apply to this problem because they focus on searching for the location of static objects (e.g., hospitals), while here the buddies are very dynamic and continuously move in the environment. Meanwhile, the approaches to protect privacy in SN [2], [3], [5] do not work either because they do not consider spatial query processing using an untrusted server and mainly focus on protecting a user from those not in his buddy list. However, here no entity beyond a user's buddy list is trusted while our goal is to enable users to query their buddy lists privately.

In this paper, we propose a framework, called Private Buddy Search (PBS), which would protect the users' profiles and locations from both the server and other non-buddy users. Our approach strikes a compromise between the two above mentioned extremes by storing users' aggregate information in various encrypted index structures at a centralized server and then pushing the querying to the clients. Hence, the clients utilize the encrypted index structures to only retrieve a small portion of the database related to the query area. Consequently, our approach benefits from the advantages of both worlds. First, all the location (and profile) updates are only sent to and stored (encrypted) in a single centralized server. Second, at the query time, by securely communicating with the (untrusted) server, the client receives enough information to answer most typical LBS+SN queries. Another major contribution of this paper is that we discuss our approach as part of a complete end-to-end framework for PBS, which includes proposed cryptographic protocols to certify users, enable secure communication within groups, support various group operations (e.g., join, leave) and enable private spatial queries such as range and k-nearest neighbor (kNN) search.

To evaluate our PBS framework, we performed extensive sets of experiments measuring client and server overhead for various PBS operations and queries. The results confirm that by distributing cryptographic and querying workload between

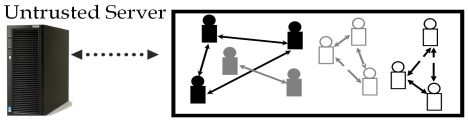


Fig. 1. The PBS Trust Model

the clients and the server, PBS supports very efficient interactions for a large number of mobile users.

The remainder of the paper is organized as follows. Section II sets out our trust and adversary models. Section III reviews the key design decisions made in PBS and Section IV details PBS constructs that collectively provide strong user privacy. In Section V, we present PBS's spatial query processing techniques whereas Section VI covers the client/server computation protocols which enable secure user interactions with the server and their peers. Our experimental evaluations are detailed in Section VII and Section VIII reviews the security issues of PBS. Finally, Section IX surveys the related work and we conclude the paper in Section X.

II. Trust and Threat Model

We model a social networking framework as a central *location server* LS (or server for short) and a set of users $U = \{u_1, u_2, \ldots, u_n\}$. Each user carries a client device (e.g., cellphone, laptop or PDA) equipped with a positioning technology such as GPS, Wi-Fi or GSM. All users communicate with LS which acts as a central repository for users' data and querying needs. For the rest of the paper, by referring to a user, we imply his client device by which he communicates with LS or other users. Each user belongs to a *group* from a set of groups $G = \{g_1, g_2, \ldots, g_m\}$. We assume users are partitioned into m disjoint groups and defer the discussion of multiple group affiliations to Section VIII. We also define a user's *buddy list* or peers as all users belonging to his group.

Users trust members of their buddy list with their sensitive information. This trust is established when a user invites another user to become part of his buddy list. We assume the trust relationship between two users is symmetric (see Figure 1). Users are willing to share their current location and other non-spatial information with their peers and query their buddies' information if they so choose. Users trust neither anyone outside their buddy list nor the LS. We use the term *adversary* to refer to any such entity.

Although users trust their peers, recent studies have highlighted that users might not be willing to share their location information even with their peers at certain times and prefer to maintain their "social boundaries" [2]. Therefore, it is important to allow users to temporarily stop sharing their location information even with their trusted buddy lists.

We assume users do not trust the location server. However, we take an *honest* but *curious* behavioral model for the location server. That is, LS does not deviate from PBS protocols. However, it is *curious* to take advantage of any sensitive user data. This is a practical assumption in many disciplines such as database outsourcing [8] and secure file sharing [10].

To interact with the location server and other peers, each user u_j creates a random pseudonym as his confidential identity. User u_j also creates the asymmetric public/private key pair $u_j.pub$, $u_j.pri$ and securely stores $u_j.pri$ on his client device. The mapping between u_j's real identity and his pseudonym is only revealed to u_j's buddies during the group invitation process (Section VI-A). Therefore, while $u_j.pub$ is used to verify u_j during his communication to the server and other peers, his pseudonym cannot be resolved to his real identity by an adversary. We use public key digital signatures to allow peers to efficiently authenticate each other and henceforth denote u_j's *Verifiable Pseudonym* by vp.

Each user u_j owns a *profile* denoted by $u_j.t$ which contains non-spatial attributes such as gender, age and self-description. We refer to a u_j's current position by the pair $< u_j.x, u_j.y >$.

To enable querying users' profile and location information within a group, any sharable user information has to be stored and maintained on the untrusted location server. In this paper, we do not consider a peer-to-peer architecture with no central repository and assume the existence of a centralized server to resemble real-world social network and location-based service architectures. In addition, a P2P solution provides a limited set of features such as location discovery of peers within a user's proximity [3] and cannot efficiently support spatial queries over a user's buddy list. Finally, while P2P approaches enable two-party computation schemes such as "where is Bob now?", they cannot respond to queries such as "which one of my friends are now in New York?" which is a fundamentally different type of query also supported by PBS.

While querying other peers, user's communication with LS should not reveal sensitive information about both identity and location of the querying user or the user being queried to any adversary [12]. Obviously, PBS or other privacy-enabling frameworks cannot protect users' information from being leaked to adversaries through other means such as physical observation. Moreover, while we always encrypt client/server communications to protect the *content* of information, anonymous communication is orthogonal to our problem and constructs such as Onion Router (Tor) [4] can be used to protect PBS against traffic analysis or eavesdropping attacks.

III. Querying an Untrusted Server

In this section, we review the major design decisions we made in PBS to support two conflicting objectives namely query efficiency and user privacy. We show how indexing *aggregate data* at the server side and shifting the query evaluation to the client side enable efficient yet private query processing. While we mainly focus on querying spatial data, non-spatial data can be queried in a similar manner.

A. Client Side Query Processing with Space-Driven Indexing

Storing encrypted user information in a central location server enables users to share and query location information of their buddies. However, as we discussed in Section I, the immediate effect of encrypting such information is crippling LS from being able to efficiently process queries. To address this drawback, we push query processing to the client side and demote the server's responsibility to a central storage and retrieval module for our encrypted indexes. However,

Fig. 2. (a) Object Space (b) Object List (c) Naive Encryptions (c) Aggregate (top) and Isolated (bottom) Indexing

client side query processing imposes certain restrictions on the choice of the partitioning used to index spatial data.

One of the most conventional ways of efficiently querying spatial data is to partition objects according to their distribution into *sets* of nearby objects and maintaining these sets in a tree structure. The most well-known example of such *data-driven index structures* [14] is the R-tree index [7] and its variants where each set is the MBR of a group of nearby objects. However, data-driven indexes are not viable solutions for our framework. First, these indexes maintain a hierarchical tree representation of objects at the server. Since the server is not trusted in our model, the encrypted tree which indexes the objects (MBR's) should be maintained by the clients and be communicated to them for query processing. This approach requires users to go through several rounds of downloading, decrypting, modifying, re-encrypting and communicating tree nodes with the server for each query or location update request. More importantly, while efficient with static data, maintaining data-driven indexes is costly in the presences of highly dynamic data [16], [9]. Therefore, using encrypted data-driven indexing is not an attractive approach in our setting.

We avoid these drawbacks by using fixed grids which is an example of *space-driven index structures* [14]. With this class of indexes, objects are mapped to a certain cell independent of other objects and solely based on some geometric criterion. Knowing the grid granularity, henceforth denoted by δ, users can directly query a certain region without having a global knowledge of objects distribution. For instance, the client can quickly identify a set of cells that (partially or fully) overlap with his range query. This is a clear advantage over data-driven indexing in terms of the complexities of maintaining and querying a centralized encrypted index. While we use grids as our primary index structure to achieve efficiency, we stress that PBS can also utilize data-driven index structures such as quad-trees at the cost of more complex client side query processing and higher communication cost. We leave as part of our future work, the design of an encrypted *data-driven* index with reasonable cost for decentralized maintenance.

B. Plain Indexing: Aggregation and Isolation

We showed how client side query processing and space-driven indexing are necessary to ensure privacy and efficiency. However, simply encrypting space-driven indexes in general, and grids in particular, do not guarantee privacy and efficiency. Consider the object distribution of Figure 2a where the server stores for each grid cell C_{x_c,y_c}, its enclosing objects as shown in Figure 2b (for simplicity, we have only shown object identifiers). To ensure privacy, we can encrypt each record as $e(C_{x_c,y_c})$**:** $< e(u_1)$**,**$e(u_2)$**,**$\ldots$**,**$e(u_r) >$ where $e()$ represents an encryption. However, looking at Figure 2c top, it is obvious that the server can roughly obtain user distribution and mobility patterns from the size of each tuple.

Alternatively, one can treat all objects in a cell as a whole during encryption. As illustrated in the bottom of Figure 2c, this process results in the encrypted index $e(C_{x_c,y_c}) :< e(u_1, u_2, \ldots, u_r) >$. However, while such encryption does not resolve the information leak, it further exacerbates the communication cost for each location update as well as buddy tracking request as an entire row has to be queried, downloaded and decrypted by the client for accessing each object.

We address the security threats and inefficiencies associated with storing raw encrypted data by breaking the non-uniform encrypted index discussed above into an encrypted *Aggregate Cell Index* (ACI) and an encrypted *Isolated Object Index* (IOI). Both of these indexes are *plain*, meaning all encrypted tuples in either index have the same size regardless of object distribution. The plain structure of ACI is achieved by storing *aggregate* data for each cell while the plain structure of IOI is achieved by indexing each individual object independent and *in isolation* from other objects. Figure 2d illustrates a simplified example to show how breaking the object information into two plain indexes prevents an adversary from learning the object distribution. We defer more details regarding the ACI and IOI structures to Section IV-B.

IV. The PBS Framework

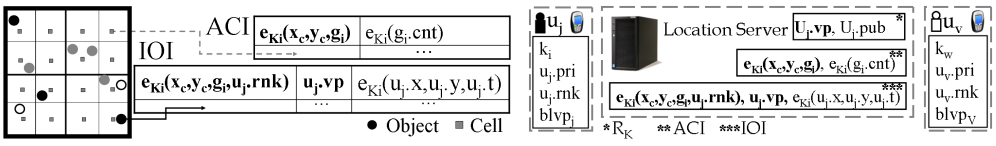
We now proceed to provide more details about PBS. We introduce the concept of group keys and detail how our proposed secure server side indexes enable users to efficiently and privately query their peers in a social networking environment.

A. Group Keys

To ensure the privacy of users, PBS should support two privacy features: *(i)* enabling peers to execute location queries on their buddy list. *(ii)* preventing an adversary from obtaining sensitive information about users during query processing. To achieve these goals, members of each group share a symmetric secret *group key* which enables users to query the current location or other information of users in their buddy list. For a group g_i, all communications between the members, as well as any sharable user information stored at LS are encrypted by g_i's group key denoted by k_i. We use $e_{k_i}()$ and $d_{k_i}()$ to denote encryption and decryption of a value with k_i, respectively.

B. Server Side Indexes

In Section III-B, we briefly discussed how to distribute users' raw encrypted location data into an encrypted Aggregate Cell Index (ACI) and an encrypted Isolated Object Index (IOI) and presented simplified versions of these indexes. We now



(a) (b)

Fig. 3. (a) ACI and IOI Structures (b) PBS Architecture

provide more details about these two encrypted indexes. We recognize two fundamentally different query types supported in PBS. *Data-driven queries* allow a user to query another user in his buddy list for his current location or other profile information. *Space-driven queries* such as range and kNN queries on the other hand, allow users to query a region (as opposed to an object) for the presence of other peers. In PBS, space-driven queries are supported by ACI while data-driven queries are supported by IOI. Figure 3a illustrates the ACI and IOI indexes stored at LS. Each tuple in ACI stores aggregate user information for each cell per each group and is represented by $ACI = \{\mathbf{e_{k_i}(x_c, y_c, g_i)}, < e_{k_i}(g_i.cnt) >\}$, where cnt or the object count denotes the number of u_j's peers co-located in u_j's cell. Each object in PBS owns a record in IOI with the schema $IOI = \{\mathbf{e_{k_i}(x_c, y_c, g_i, u_j.rnk)}, \mathbf{u_j.vp}, < e_{k_i}(u_j.x, u_j.y, u_j.t) >\}$. A user's rank $u_j.rnk$ is a sequence number assigned to each object denoting an ordering between peers of a group in each cell based on their arrival time. Therefore, $u_j.rnk \in \{1 \dots cnt\}$. Note that the bold-faced columns are indexed and searchable. Aside from the above indexes, the server maintains each user's public key certificate in a *Key Relation* $R_K = \{\mathbf{u_j.vp}, < u_j.pkc >\}$ for authentication purposes. Figure 3b illustrates a global view of how private information is distributed between different PBS entities.

The benefits of breaking the object information into the ACI and IOI indexes are threefold. First, our two proposed indexes prevent the adversaries from learning any information about the object distributions from the size of the encrypted indexed data. This property is achieved via the plain nature of ACI and IOI indexes. Second, ACI and IOI indexes efficiently support various types of queries. With space-driven queries, knowing the grid granularity, users first identify the right cells which are likely to contain information about their buddies. Next, by learning each cell's aggregate information from ACI, they form a *request packet* which is a set of requests for IOI tuples each containing information about one of user's buddies. For data-driven queries, users directly query their peers in IOI using their pseudonyms to *locate* or *track* them without knowing their current location. We detail query processing in Section V. Third, as we discuss in Section VI, PBS supports a wide range of features that allow users to interact with their peers and other users in a social networking environment.

V. Private Spatial Queries with PBS

We now discuss how the ACI and IOI indexes enable private evaluation of space and data-driven queries in a social network.

A. Range Queries

A range query $Range(R, g_i)$ allows a user u_j of group g_i to find the location of his peers in the rectangular region R (circular ranges are approximated by their surrounding rectangles by later filtering excessive results at the client side). Algorithm 1 illustrates how range queries are supported with PBS. Users first query ACI for aggregate information of the cells overlapping with R. Next, the server's response is decrypted and a second request for IOI tuples each corresponding to one of the user's buddies in R is formed. Processing each range query involves two rounds of client/server communication. These steps are underlined in Algorithm 1 which illustrates range query processing in PBS.

Algorithm 1 Range Queries

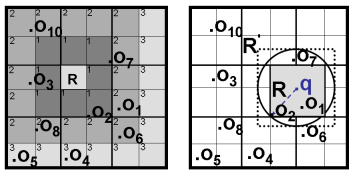
Require: R, g_i, δ; {range, group and grid granularity info }
  for all $C =< x_c, y_c > overlapping\ with\ R$ **do**
    $req_1 \leftarrow req_1 \cup e_{k_i}(x_c, y_c, g_i)$;
  $res_1 \leftarrow LS.ACI[req_1]$; {server processing req_1}
  for all $T = e_{k_i}(x_c, y_c, g_i), e_{k_i}(cnt) \in res_1$ **do**
    $C < x_c, y_c > \leftarrow d_{k_i}(e_{k_i}(x_c, y_c, g_i)); cnt \leftarrow d_{k_i}(e_{k_i}(cnt))$;
    if $(cnt \neq 0)$ **then**
      $U \leftarrow U \cup C < x_c, y_c >$; {find non-empty cells}
  for all $C \in U$ **do**
    for $(rnk \leftarrow 1; rnk \leq C.cnt; rnk++)$ **do**
      $req_2 \leftarrow req_2 \cup e_{k_i}(x_c, y_c, g_i, rnk)$; {add objects $\in R$}
  $res_2 \leftarrow LS.IOI[req_2]$;{server processing req_2}
  for all $T' =< e_{k_i}(u_j.x, u_j.y, u_j.t) > \in res_2$ **do**
    $< u_j.x, u_j.y, u_j.t > \leftarrow d_{k_i}(T')$;
    $res \leftarrow res \cup < u_j.x, u_j.y, u_j.t >$
  return (res);

B. k-Nearest Neighbor Queries

Resolving kNN queries is similar to range queries except that here, the region containing users' k-nearest peers is not known in advance. Therefore, users progressively form concentric rectangular regions and query ACI until enough cells are found that include at least k objects. Figure 4a illustrates this progressive expansion strategy. The cells are shaded and numbered according to the step they are visited. Next, a second request queries IOI for objects located in the expanded region. The server's response will hence include location information of k nearby objects. However, approximating a circular region with rectangular regions might result in some false negatives (points such as O_7 in Figure 4b located inside the circle but outside the rectangle) that are part of the result set. Therefore, once the k^{th} object is found, users expand the queried region R to a *safe region* R' which represents the region including false negatives to guarantee query accuracy. It is easy to verify that R' is a square with sides $2 \times \lceil ||c_q - far_q(k)|| \rceil$ where c_q is the cell containing q and $far_q(k)$ is the cell containing q's k^{th} nearest object in R and $||.||$ is the Euclidean norm [16]. This process is performed by the $addSafeRegion()$ function in Algorithm 2 which details kNN query processing (underlined sections represent client/server communication).

C. Buddy Tracking

In addition to the *space-driven* queries discussed above, an important functionality in a mobile social networking



(a) Expansion (b) Safe Region

Fig. 4. kNN Algorithm

environment is to enable users to query a specific peer's location or profile information. To enable these *data-driven* queries, users keep the list of their buddy lists vp's in their client devices (denoted by *blvp* in Figure 3b). The user u_j trying to track u_v's location (or to view $u_v.t$), queries IOI with $u_v.vp$. As part of their invitation, users have received the inviter's vp, as well as his/her real identity and hence they keep this mapping in *blvp*. Note that the server cannot verify whether u_j and u_v belong to the same group and hence cannot prevent u_v's information from being queried by adversaries. However, this is not an issue as an adversary cannot decrypt the sever's response if he is not part of u_v's buddy list.

Algorithm 2 kNN Queries

Require: q, k, g_i, δ; {kNN center, k, group and granularity}
  $ct \leftarrow 0$; {object count}
  $x_c \leftarrow \lfloor \frac{q.x}{\delta} \rfloor$; $y_c \leftarrow \lfloor \frac{q.y}{\delta} \rfloor$;
  $req_1 \leftarrow e_{k_i}(x_c, y_c, g_i)$; {adding the querying cell}
  Let $region \leftarrow C =< x_c, y_c >$;
  while $(ct < k)$ **do**
    $region = expand(region)$; {stripe surrounding $region$}
    for all $C =< x_c, y_c > \in region$ **do**
      $req_1 \leftarrow req_1 \cup e_{k_i}(x_c, y_c, g_i)$;
    $res_1 \leftarrow LS.ACI[req_1]$; {server processing req_1}
    for all $T =< e_{k_i}(cnt) > \in res_1$ **do**
      $C.cnt \leftarrow d_{k_i}(T)$;
      if $(C.cnt \neq 0)$ **then**
        $ct+ \leftarrow C.cnt$;
        for $(rnk = 1; rnk \leq C.cnt; rnk++)$ **do**
          $req_2 \leftarrow req_2 \cup e_{k_i}(x_c, y_c, g_i, rnk)$; {tag cell's objects}
    $res_2 \leftarrow LS.IOI[req_2]$;{server processing req_2}
    for all $T' =< e_{k_i}(u_j.x, u_j.y, u_j.t) > \in res_2$ **do**
      $u =< u_j.x, u_j.y, u_j.t > \leftarrow d_{k_i}(T')$;
      $res_2 \leftarrow res_2 \cup u$;
    $req_1 \leftarrow \emptyset; req_2 \leftarrow \emptyset$;
  return $(res_2 \cup addSafeRegion(u_j.x, u_j.y, res_2))$;

VI. PBS Operations

PBS supports a range of functionalities that enable various user interactions with other peers. In this section, we provide the two-party computation protocols between the users and the server that enable such interactions.

A. Group Related Operations

Initiating Groups: To initiate a group g_i, a user u_j creates a secret symmetric group key k_i and computes the respective ACI and IOI tuples by setting $g_i.cnt = 1, u_j.rnk = 1$. He then signs this *group init request* and sends it to the server. The server verifies the signature and processes the request. Note that for all client/server communications, users bind a *nonce* to their request to thwart replay attacks.

Joining Groups: A user can be invited to a group by any of the group members. In order for a user $u_j \in g_i$ to form an *invitation request* to u_v, he first needs to learn $u_v.vp$ (this step is analogous to asking u_v for his email address or phone number except that here, vp is anonymous). Next, u_v receives the invitation $e_{u_v.pub}\left(u_j.vp, u_j, e_{u_j.pri}(invitation, k_i)\right)$. The invitee (i.e., u_v) first decrypts the invitation with $u_v.pri$. This step ensures no one else can take advantage of the invitation or learn anything by snooping the communication. Next, u_v tries to decrypt the invitation. If successful, this step guarantees the invitation is sent by u_j and transfers the group key to u_v. Having g_i and k_i, u_v queries ACI and learns the object count of his cell (i.e., cnt). He then sets $u_v.rnk = cnt + 1$ and constructs a signed *join request* to the sever which updates ACI (incrementing u_v's cnt) and adds one tuple to IOI storing u_v's information. Note that the server does not learn any information about the group or the identity of either user.

User Revocation: Revoking users from a group is challenging as the revoked user can share the secret group key with adversaries to snoop future group communications. One solution to revocation is for the remaining users of each group to negotiate a new group key and to re-encrypt all relevant ACI and IOI tuples. Unfortunately, this is a costly approach.

We use the *lazy revocation* with *key rotation* scheme from [10] for revoking users' access to shared files. Following the eviction of a user from g_i, the remaining users negotiate a new group key which will be used to encrypt all *future* ACI and IOI tuples during write operations such as location updates. This scheme is called lazy revocation as the revoked users still have access to the content they had access to, prior to their eviction. However, this is not a security flaw as such members could have cached the data and hence blocking their access to unmodified data does not have any advantages. As new users join PBS or during updates to the two indexes, one always uses the most recent group key. However, the *key rotation* scheme proposed by [10] guarantees that after each eviction: *(i)* given the most recent key, it is easy for all existing group members to *rotate a key backward* and obtain previous keys that are still in use for certain tuples while *(ii)* it is computationally infeasible for any expelled user from the group to compute future keys given their current version of the key. This scheme allows group members to only keep the most recent version of the key and only if needed, compute the previous key versions.

B. User Related Operations

Location and Profile Updates: Depending on their types, location and profile updates can be divided into two groups. For intra-cell movements of user u_j from (x, y) to (x', y') in the cell C_{x_c,y_c} and updates on $u_j.t$, only a single change in IOI is required. The user u_j sends a signed *update request* to LS for his record indexed by $e_{k_i}(x_c, y_c, g_i, u_j.rnk)$ to be updated to $e_{k_i}(u_j.x', u_j.y', u_j.t')$. LS verifies the signature and updates the IOI tuple. For inter-cell movements, three rounds of more complex communications between the client and server are needed. The basic steps taken are querying the old and new cell information for u_j, replacing u_j's position with the last user who has joined u_j's old cell and finally updating the ACI and IOI records affected by u_j's move in old and new cells.

Although the cell count cnt remains encrypted in ACI and IOI during the above process, the server knows the initial value of cnt for each cell C. Therefore, each new (repeated)

encrypted value of cnt would imply an increase (decrease) in the number of objects in C. To avoid this vulnerability, we always attach a timestamp to each cnt before encryption. This makes an increase or decrease equiprobable (during updates) from the server's point of view. Therefore, after i updates, the server can guess $C.cnt$ only with probability $\frac{1}{\lceil \frac{i}{2} \rceil}$ which quickly declines after a short time. Note that even a right cnt guess reveal neither the actual location of C nor the identity of its enclosed objects at any time.

Privacy Mode Request: As we discussed in Section I, users sometimes prefer to stop sharing their location information even with their peers due to a variety of reasons [2]. PBS allows users to efficiently go to a *Privacy Mode* by sending an *IOIremove* request to temporarily remove their location information from IOI, as well as an *ACIupdate* request to accordingly adjust the count value of their current cell after querying the server for the cell's current information. To switch back to the original *Privacy Mode* and start sharing location information, the above process is simply reversed.

VII. Experimental Evaluation

In this section, we empirically examine the overall efficiency of PBS. We conducted extensive experiments to determine the effectiveness of our framework in terms of *(i)* PBS operations overhead *(ii)* the effect of different datasets and grid granularity on spatial queries *(iii)* the client/server computation and communication overhead for Algorithms 1 and 2 and *(iv)* comparing PBS with similar approaches.

(a) Oldenburg (b) Hennepin

Fig. 5. Datasets

A. Datasets and Experimental Setup

We used the widely accepted network-based generator for moving objects [1] to generate our datasets. The generator takes the road map of a region (e.g., a city) and outputs for each object, a set of locations along the road network of the given region. We used as input the city of Oldenburg in Germany and the Hennepin County in Minnesota. Figure 5 illustrates simulated user locations for these two datasets. The second dataset is also used to compare PBS with Capser [13] which also enables querying moving users through a trusted anonymizer. For Oldenburg, we generated three user datasets O_1, O_2 and O_3 containing 500, $5K$ and $50K$ users, respectively. We fixed the group number to 5 in these three datasets to generate groups with average size of 100, $1K$ and $10K$ users, respectively. Similarly, we used $50K$ users in 5 groups for the Hennepin County dataset, denoted by HC.

Experiments were run on two different Intel P4 2.66 GHz machines with 4 GB of RAM acting as clients and the server. We used sockets for the client server communications over the TCP/IP protocol to measure the actual network latency, DES for symmetric key encryption/decryption, 1024 bit DSA for public key cryptography and authentication and SHA1 for one way functions and pseudorandom number generation.

From the algorithms and operations introduced in previous sections, it is obvious that most of computation complexity is transferred to the client side in order to achieve both security and scalability. Therefore, throughout the following experiments, we focus on average end-to-end response time from client side, denoted by T_C, in milliseconds as a key metric for PBS efficiency. Later in Section VII-D, we provide a comprehensive breakdown of the response time in terms of client and server computation and communication time.

B. PBS Operations

As our first set of experiments, we measure the overall response time for joining groups (t_{join}), location update (t_{update}) and buddy tracking (t_{track}) operations. Results for t_{join} and t_{track} were averaged over $1K$ requests. As for t_{update}, we averaged the overhead for $1K, 100K, 1M$ and $1.5M$ location updates for the O_1, O_2, O_3 and HC datasets, respectively. The results are shown in Figure 6.

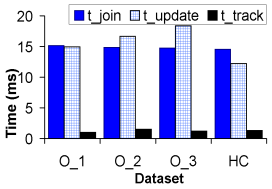


Fig. 6. PBS Operations

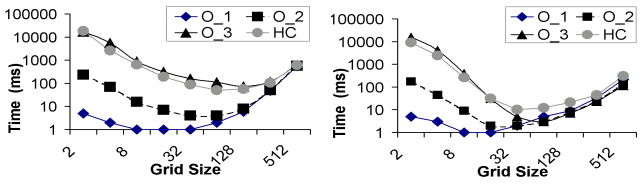
We observe that the overhead of all three operations is almost invariant to the choice of the dataset and stays around 15 to $20ms$ for t_{join} and t_{update} and less than $1ms$ for location tracking (due to its simplicity). Also, due to the plain structure of ACI and IOI and non-spatial nature of these operations, they are not significantly affected by grid granularity.

C. Spatial Queries

In this section, we evaluate the effect of the grid granularity (δ) on the performance of Algorithms 1 and 2. We first study the effect of δ on the response time for 100 randomly selected range queries. Figure 7a illustrates the overall response time t_{range} for different datasets with 1% selectivity (i.e., relative range size). While having very similar trends, shrinking selectivity to 0.5% and 0.1% resulted in smaller values of t_{range}.

There is a trade-off in choosing the right value of δ for space-driven queries. For coarse grids, users have to falsely query numerous objects from IOI simply because they are co-located in a large cell with other objects relevant to the query. This increases the communication and processing overhead. Alternatively, fine-grained grids result in numerous cells overlapping with range queries (or expanded with kNN queries) whose information has to be queried from ACI which in turn increases the client/server communication overhead.

One important observation from Figure 7a is that while optimum grid granularity is a function of the number of users [16], t_{range} is not affected by the number of objects for highly fine-grained grids and converges to very similar values across all datasets. This is because $t_{range} = n_1 \times t_{ACI} + n_2 \times t_{IOI}$



(a) (b)

Fig. 7. (a) Range Queries (b) NN Queries

and for very small values of δ, the number of cell queries from ACI significantly dominates the number of object queries from IOI (i.e., $n_1 >> n_2$). Given that both indexes are plain, $t_{ACI} \approx t_{IOI}$ and hence t_{range} will be dominated by the common value of $n_1 \times t_{ACI}$ across different datasets. Finally, it is obvious from Algorithm 1 and Figure 7a that t_{range} increases for more dense datasets due to an increase in average IOI tuple requests from the server per each query.

Next, we examined the response time of Algorithm 2 for evaluating kNN queries (t_{kNN}). Figure 7b illustrates t_{kNN} for 100 randomly generated nearest neighbor queries for all datasets (similar trends were observed for higher values of k). Similar to range queries, there is a trade-off for choosing the right value of δ for optimum query performance. There is however, a distinct trend observed for processing kNN queries which is caused by the fundamental difference between these two queries. While processing range queries in densely populated areas requires more accesses to IOI tuples, kNN queries are evaluated more efficiently in dense areas simply because the region containing the result set is relatively small. This explains the trend change in Figure 7b for fine-grained grids. While the number of IOI requests remains the same, sparse datasets examine significantly more ACI tuples to find the result set. Finally, the slight variation between the response times of HC and O_3 are caused by different mobility patterns of simulated users in these two cities.

D. End to End Query Processing

As our next set of experiments, we measure the overall efficiency of PBS based on *(i)* t_s, the time it takes to process a query at the server *(ii)* t_c, the client side processing time *(iii)* t_{cs}, client to server communication time and *(iv)* t_{sc}, server to client communication time. For this experiment, we generated 100 randomly chosen range queries with 0.1%, 0.5% and 1% selectivity and measured the above four values.

Several observations can be made from our findings summarized in Figure 8. The first noticeable trend is an increase in client and server's overhead for larger datasets, as well as for higher selectivity (notice the different scales of Y axes). To explain these trends we first note that $t_{range} = n_1 \times t_{ACI} + n_2 \times t_{IOI}$. For a fixed selectivity, increasing the dataset size will only increase n_2 in the above equation and higher selectivity for a fixed dataset causes both n_1 and n_2 to increase. The graphs also show that both client and server's overhead are reasonable for all 6 different cases always staying below 60 milliseconds. We also see that the HC dataset consistently yields better results than O_3 despite having the same number of objects. This is caused by more uniform distribution of objects in the HC dataset. Finally, t_{cs} in Figure 8 increases more rapidly than t_c as dataset size grows due to higher server overhead for larger datasets causesing the client to spend more time communicating with the server.

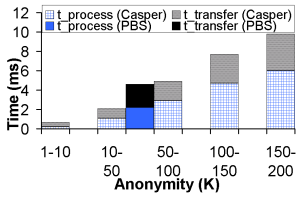


Fig. 9. Comparison with Casper

E. Comparison with Other Approaches

The closest work to PBS in terms of enabling private queries over dynamic user locations is the Casper system [13] which allows users to perform range and kNN queries to request other users' locations. However, privacy in Casper is achieved by relying on a trusted anonymizer to cloak users, per query, in an *anonymity set* which contains at least $K-1$ other users.

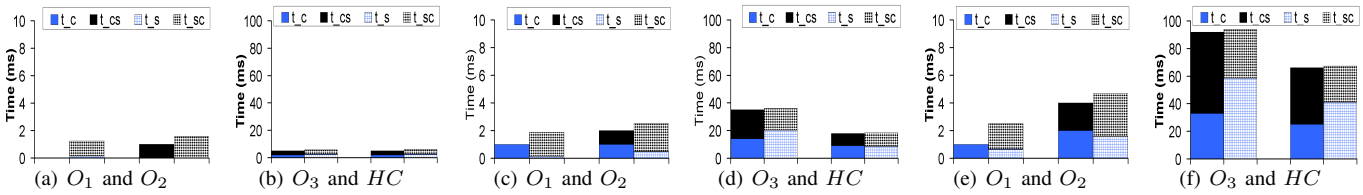
We used the HC dataset with $50K$ moving users to compare PBS with Casper for evaluating NN queries. Aside from the drawbacks of relying on an anonymizer (detailed in Section IX and [12], [6], [15], [11]), as illustrated in Figure 9, Casper suffers from a costly privacy/efficiency trade-off. To achieve comparable performance with PBS, Casper provides significantly lower privacy guarantees by making a user indistinguishable among a small anonymity set of ($K < 50$) users.

VIII. Security Analysis

In this Section, we briefly review some key security strengths and weaknesses of PBS.

Multiple Group Affiliation and Variable Privacy: So far we have assumed a binary notion of trust between two users (i.e., *buddies* vs. *adversaries*). However, users might have a more flexible approach towards privacy. For instance, while a user is willing to *continuously* share her *exact* location with her family or close friends, she might prefer to share much coarser information during certain times with her co-workers. This notion of *variable privacy* can be supported in PBS by allowing users to join multiple groups with different levels of privacy where members of each group negotiate a *group-specific* δ (i.e., spatial resolution) based on their common privacy preferences. Users with multiple group affiliations use a different vp for each of their group memberships. This technique, however, exposes PBS to a powerful attack where a user $u_j \in g_i, g_i'$ shares g_i's secret key k_i with someone in his buddy list from $g_i' \neq g_i$. Addressing this attack is challenging and existing approaches do not provide a solution for it. One solution is to store each group key in a client's tamper-resistent device to prevent users from accessing and hence being able to share keys with their peers from other groups.

Server Collusion with Adversaries and Trusted Users: While group keys prevent the sever (or multiple adversaries) to collude against a user, PBS cannot protect user privacy against an adversary colluding with a seemingly trusted user in one's buddy list who might share the group key with an outsider. This powerful attack remains an open problem in our system



(a) O_1 and O_2 (b) O_3 and HC (c) O_1 and O_2 (d) O_3 and HC (e) O_1 and O_2 (f) O_3 and HC

Fig. 8. Response Time for (a,b) 0.1%, (c,d) 0.5%, and (e,f) 1% Selectivity

as well as other privacy management systems such as [3]. However, it can only affect users of the compromised group.

Statistical Cryptanalysis: The server can compile query frequencies for different IOI tuples to find the most frequently queried user, or the user with most number of location updates. While such *known plaintext* attacks are powerful with querying *static* data (e.g., restaurants), the server cannot infer the original and encrypted objects mapping to identify or locate users due to the highly dynamic nature of users. Similarly, the server might infer relative cell positions by the sequence ACI tuples are requested by clients in Algorithms 1 and 2. To thwart this attack, users can randomly break their requests packets into two or more sub-requests to protect the expansion sequences of cells.

IX. Related Work

Privacy issues in mobile social networking systems have been the focus of several social and technical studies [2], [3], [5]. Perhaps the most relevant study to our work is the SmokeScreen framework [3] proposed for private location sharing. SmokeScreen supports presence sharing with trusted users, as well as with strangers which is a feature PBS does not support. However, this work bears strong differences with our approach. First, SmokeScreen operates under a model with users "periodically broadcasting their identity via short-range wireless technology such as BlueTooth or WiFi". Second, it does not study query processing. Third, it employs a complex trusted broker which maintains a user interest graph and other sensitive information about user relationships.

Numerous research studies have also examined user privacy in location based services [11], [12], [6], [15]. However, they mostly rely on cloaking or trusted anonymizers to blur a user's location and do not focus on querying dynamic user locations. The only study in this group which addresses querying dynamic user data is the Casper framework proposed in [13]. Although Casper supports range and kNN queries, it suffers from several privacy issues shared among cloaking-based approaches. For instance, under certain distributions, cloaking might reveal exact user locations to malicious entities [11]. Furthermore, the quality of service degrades significantly for users with strict privacy preferences. Finally, Casper does not address the issues of trust among users and assumes that all users trust each other and a central anonymizer.

To the best of our knowledge, PBS is the first work to address privacy issues of enabling mobile users to execute a set of spatial queries predominantly used in social networks. While supporting these queries, PBS does not suffer from privacy implications of cloaking techniques or their costly query overhead by utilizing decentralized and self-maintaining encrypted index structures stored at a central untrusted server.

X. Conclusion and Future Work

In this paper we presented the *Private Buddy Search* (PBS) framework which enables users to privately perform a variety of queries and interactions with other users in a highly dynamic social network. Our experimental evaluation verified that PBS is highly scalable due to its distributed query workload. PBS provides various user interactions currently supported in social networks while protecting the privacy of its users. As part of our future work, we are performing an in-depth study of PBS components' security. We are also extending PBS to employ more complex indexing schemes to achieve more scalability for various user distributions. We also plan to focus on the social aspects of PBS such as relaxing the single group affiliation assumption.

Acknowledgment

This research has been funded in part by NSF grants IIS-0238560 (PECASE), IIS-0534761, CNS-0831505 (CyberTrust), and the NSF Center for Embedded Networked Sensing (CCR- 0120778). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.

[2] S. Consolvo, I. E. Smith, T. Matthews, A. LaMarca, J. Tabert, and P. Powledge. Location disclosure to social relations: why, when, & what people want to share. In *CHI'05*, pages 81–90.

[3] L. P. Cox, A. Dalton, and V. Marupadi. Smokescreen: flexible privacy controls for presence-sharing. In *MobiSys'07*, pages 233–245.

[4] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *USENIX'04*, pages 303–320.

[5] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt. Micro-blog: sharing and querying content through mobile phones and social participation. In *MobiSys'08*, pages 174–186.

[6] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private queries in location based services: anonymizers are not necessary. In *SIGMOD'08*, pages 121–132.

[7] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD'84*, pages 47–57.

[8] S. Hohenberger and A. Lysyanskaya. How to securely outsource cryptographic computations. In *TCC'05*, pages 264–282.

[9] D. V. Kalashnikov, S. Prabhakar, and S. E. Hambrusch. Main memory evaluation of monitoring queries over moving objects. *Distrib. Parallel Databases*, 15(2):117–135, 2004.

[10] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable secure file sharing on untrusted storage. In *FAST'03*.

[11] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preserving anonymity in location based services. *A Technical Report*, 2006.

[12] A. Khoshgozaran and C. Shahabi. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In *SSTD'07*, pages 239–257.

[13] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The new casper: Query processing for location services without compromising privacy. In *VLDB'06*, pages 763–774.

[14] P. Rigaux, M. Scholl, and A. Voisard. *Introduction to Spatial Databases: Applications to GIS*. Morgan Kaufmann, 2000.

[15] M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu. Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In *ICDE'08*, pages 366–375.

[16] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *ICDE'05*, pages 631–642.