SPIRAL:A Scalable Private Information Retrieval Approach to Location Privacy

Ali Khoshgozaran, Houtan Shirani-Mehr and Cyrus Shahabi
University of Southern California Los Angeles, CA 90089

Abstract

Protecting users' location information in location-based services, also termed location privacy*, has recently garnered significant attention due to its importance in satisfying users' privacy concerns when using location-aware services. Several approaches proposed in the literature blur the user's location in a region by increasing its spatial extent or* anonymizing *the user among several other users. Such approaches in nature require users to communicate through a trusted anonymizer for all of their queries which can impose unrealistic overall communication/computation overhead between the server and the anonymizer for users with more stringent privacy requirements. We revisit the location privacy problem with the objective of providing significantly more stringent privacy guarantees and propose SPIRAL,* a Scalable Private Information Retrieval Approach to Location privacy*, which is to the best of our knowledge, the first approach to utilize practical Private Information Retrieval (PIR) as a more fundamental approach to enable blind evaluation of range queries. We perform several experiments on real-world data to evaluate the effectiveness and the feasibility of our approach.*

1 Introduction

With many applications, locating dynamic objects is of particular interest. Almost all location-based services are somehow *aware* of their users locations in order to provide customized services. However, the implicit assumption that clients are willing to share their private location information with a potentially untrusted location server is being challenged. The explosive growth of affordable GPS-enabled cellphones has resulted in a variety of innovative applications based on user's location data. Meanwhile, utilizing such ubiquitous devices as means of locating people and accessing their private locations in return of a location-aware service has become the source of many concerns [17] and in some cases distressing privacy violations [1].

Existing approaches to achieve location privacy are mostly based on hiding a user's location in a larger (and thus harder to track) region (*cloaking*) or among a set of other users (*k-anonymity*). These approaches by design require an anonymizer as sophisticated as the location server itself, to act as a proxy between users and the server per query. Aside from creating a single point of failure/attack, this approach has two important drawbacks. (i) In many scenarios *cloaking* and anonymization techniques cannot protect user's location information. This is due to the fact that based on user distributions in the space and the value of k (or similarly the cloaking region size), precise user location can be revealed using several techniques such as monitoring a sequence of queries over time or reasoning about the possible location of the query point [11]. (ii) With these approaches, users should trade-off their privacy with the accuracy of the query result or the efficiency of the query processing because a larger cloaking region or a bigger k may result in a significantly large query result which includes many unnecessary data points. Alternatively, decreasing k or the size of the cloaking region will directly increase the probability of narrowing down the user's location. Therefore, preserving users' location information might not always be possible regardless of the size of the cloaking space or k.

In this paper, our goal is to provide significantly more stringent privacy guarantees that are invariant to the total number of users, the size of the region enclosing users or their querying patterns. We propose SPIRAL, *a Scalable Private Information Retrieval Approach to Location privacy*, which to the best of our knowledge is the first realization of utilizing *Private Information Retrieval* (PIR) as a more fundamental approach of protecting location information and show how SPIRAL enables blind evaluation of range queries. As part of our future work, we are extending SPIRAL to support the K-nearest neighbor queries as well.

The main intuition behind using PIR is to provide a more generalized and robust way of blinding the untrusted location server by converting spatial query processing into several private database retrievals. Although in SPIRAL we utilize a specific set of PIR schemes, the PIR module can be treated as a black box throughout the query resolution process and thus any practical PIR scheme can replace our current PIR scheme. However, utilizing PIR for location privacy poses a fundamental dilemma for processing spatial queries. Suppose a user located at point P is interested in a subset of objects located in a subregion S of the space where an object O's information ($O \in S$) is stored at the

ith record in the database DB. Evaluating the user's spatial query clearly requires information about the objects in the *vicinity* of S, P or both. This information cannot be provided to DB because knowing the spatial relationships between the items in its database clearly reveals user's private location information to the untrusted server. Alternatively, moving this knowledge to the users will require the query processing to happen at the client or a linear scan of the whole database at the server, both of which are not desirable. In order to avoid this dilemma, we propose to perform the query evaluation on a trusted computing environment while the required data to perform the query is being privately queried from the local untrusted location server. This way, the queries can be evaluated blindly as the server does not learn which items were retrieved and the communication overhead is minimized as the trusted computing module only transmits the result set to the users. In summary:

- We propose SPIRAL, a novel PIR-based approach to blindly evaluate range queries (Section 4).
- We show how SPIRAL satisfies the most stringent privacy concerns in location-based services (Section 5).
- We perform several experiments on the implementation of our proposed system and study its efficiency and feasibility with real-world data (Section 6).

2 Location Privacy Preliminaries

The most important property of any location privacy scheme is to protect users' location information. We now define the privacy metrics, the adversary and the information leak model and use them throughout the paper to evaluate how our proposed approaches enable location privacy.

2.1 Privacy Metrics

Given a set of objects $DB = (o_1, o_2, \ldots, o_n)$ in 2-D space and a set of users $U = (u_1, u_2, \ldots, u_M)$, we try to satisfy the (slightly modified variants of) privacy requirements proposed by [12] to ensure that evaluating a spatial query does not reveal any sensitive location information to the potentially untrusted server. Throughout the paper, o_is are assumed to be objects represented by the triplet $<$ *longitude,latitude,id* $>$.

Definition 1. u-anonymity: While resolving a query, the user issuing the query should be indistinguishable among the entire set of users. Therefore, for each query q, $P(q) = \frac{1}{M}$ where $P(q)$ is the probability that query q is issued by a user u_j where $j \in \{1 \ldots M\}$ and M is the number of users. Definition 1 ensures the server does not know which user issued the query q; however, we also need to ensure the server does not know from which point the query q is issued. This requirement is captured by Definition 2.

Definition 2. a-anonymity: While resolving a query, the location of the query point should not be revealed. In other words, for each query q, $P'(q) = \frac{1}{area(A)}$, where A is the entire region covering all the objects in DB and $P'(q)$ is the probability that the query q was issued by a user located at a point inside A.

It is clear that the commonly used anonymity and cloaking approaches (e.g., [11] and [13]), in which a user is indistinguishable among k other users or his location is blurred in a cloaked region R are special cases of Definitions 1 and 2 with $M = k$ and $A = R$, respectively.

Definition 3. Blind evaluation of spatial queries: A spatial query is said to be blindly evaluated if the *u-anonymity* and *a-anonymity* constraints are both satisfied.

It is clear now that location privacy is achieved if all spatial queries issued by users are evaluated blindly. A location server is considered *privacy aware* if it is capable of blindly evaluating a spatial query while providing accurate results. Throughout the paper, we adopt these stronger metrics and ensure our query processing technique is privacy aware.

It is important to be able to measure how much private information is revealed by performing the necessary steps in responding to a spatial query. Similar to [3], we use entropy to define how much information is *leaked* by following a privacy preserving protocol. A set of queries $Q = \{q_1, q_2, \ldots, q_k\}$ is privately evaluated if and only if the joint entropy of the variables $q_1, q_2, \ldots, q_k$ is maximal. This definition of information leak captures the "absence of information about a set of queries".

As discussed above, our motivation is to use a more generalized way of blinding the untrusted location server through *Private Information Retrieval (PIR)* and converting spatial query processing into several private database retrievals. Therefore, one of the key challenges behind such a framework is devising spatial algorithms that enable query evaluation using a privacy aware server that can only retrieve items privately and does not possess any location information. In the next section we provide an overview of several PIR schemes and the one we utilize in our framework. Later in Section 5, we show how our proposed PIR scheme integrates with other SPIRAL components.

2.2 Adversary Model

The adversary's goal is to find user's location information. In order to obtain this information the computationally bounded adversary can take different sets of approaches. We assume the strongest adversary, who subscribes to the system as a normal user and colludes with the untrusted server to find the relationship between users' locations and what is stored at the untrusted server. In addition, we assume that for static datasets, the original data is publicly available and thus the adversary can perform what is known as a *known plain text attack*. The above characteristics model the strongest adversary.

3 Private Information Retrieval

There is a wide spectrum of scenarios in which a user needs to gain access to a specific record of a database but does not want to reveal the record in which he is interested. More formally, a Private Information Retrieval (PIR) protocol allows a user to retrieve the ith record from a database of size n stored at an untrusted server, without revealing i to the server. The class of PIR approaches can be roughly divided into cryptographic and hardware-based approaches. While cryptographic approaches make use of homomorphic encryption, quadratic residues and other cryptographic properties to achieve PIR, hardware-based techniques utilize a secure coprocessor which acts as a securely protected computing space residing at the untrusted *host* machine that enables private querying of the data [3]. We utilize the latter scheme as a building block for our privacy-aware location server to achieve acceptable communication and computation complexity. However, it is important to note that we treat the PIR module as a black box throughout the query resolution process and thus any other practical PIR scheme (that is either proposed or will emerge) can be incorporated into our current SPIRAL framework. We now elaborate on how secure coprocessors enable a practical PIR scheme.

3.1 Hardware-Based PIR

A Secure Coprocessor (SC) is a general purpose computer designed to meet rigorous security requirements that assure unobservable and unmolested running of the code residing on it even in the physical presence of an adversary [15]. These devices are equipped with hardware cryptographic accelerators that enable efficient and fast implementation of cryptographic algorithms such as DES and RSA [14]. Recent advances in hardware technology have enabled successful implementation of several real-world applications such as data mining [5] and trusted co-servers for Apache web-server security [9] on trusted computing environments. Note that trusting a secure processor is substantially different from trusting a location server in several respects. First, aside from being built as a tamper resistent device, the secure coprocessor is a hardware device specifically programmed to perform a given task while a location server consist of a variety of applications using a shared memory. Secondly, unlike the secure coprocessor in which the users only have to trust the designer, using a location server requires users to trust the server admin and all applications running on it as well as its designer. Last but not least, in our setting, the secure coprocessor is mainly a *computing* device that receives its necessary information, per session from the server, as opposed to a server which both stores location information and processes spatial queries.

The idea behind using a secure coprocessor is to place a trusted entity as close as possible to the untrusted host to disguise the selection of desired records within a black box. In order to avoid the linear cost of going through each record in the host or sending the entire dataset to the user (i.e., O(n) computation and communication cost, respectively) we use the technique proposed by Asonov et al. [4] to achieve optimal (i.e., constant) query computation and communication complexity at the cost of performing as much offline precomputation as possible. Since the protocol uses shuffling techniques, we first offer a brief overview of how shuffling can be efficiently performed.

Definition 4. Random Permutation: For a database DB of n items the random permutation π transforms DB into DB_π such that $DB[i] = DB_\pi[\pi[i]]$. For example for $DB = \{o_1, o_2, o_3\}$ and $DB_\pi = \{o_3, o_1, o_2\}$ the permutation π represents the mapping $\pi = \{2, 3, 1\}$. Therefore $DB[1] = DB_\pi[\pi[1]] = DB_\pi[2] = o_1$, $DB[3] = DB_\pi[\pi[3]] = DB_\pi[1] = o_3$ etc. It is easy to verify that the minimum space required to store a permutation π of n records is $n \log n$ bits.

The basic idea behind utilizing a secure coprocessor is to use π to privately shuffle and encrypt the items of the entire dataset DB. While this encrypted shuffled dataset DB_π is written back to the server, SC keeps π for itself. Later, a user interested in the ith element of DB, encrypts his query using SC's public key and sends it to SC through a secure channel. SC can then retrieve and decrypt $DB_\pi[\pi[i]]$, re-encrypt it with users' public key and send it back (hereinafter we distinguish between a *queried item* which is the item requested by the user and *retrieved/read record* which is the item SC reads from DB_π). Although the server is *blindly* retrieving an encrypted record and returning it to SC, the scheme is not yet private. Suppose two consecutive queries for the ith and jth elements of DB are received by SC. Retrieving $DB_\pi[\pi[i]]$ and $DB_\pi[\pi[j]]$ *leaks* some information to the untrusted server or an adversary monitoring the reads (retrieved records) as he can verify if $i = j$ (i.e., whether similar or different records were read from DB_π). Using an even more effective attack, the adversary can subscribe to the system as a user and actively query for an item k and verify if i (or j)$= k$. This attack is similar to *known plaintext and chosen plaintext cryptanalysis* in cryptography. Using the above approach, it is easy to show that a set of consecutive queries's joint entropy is not maximal.

The above problem can be avoided using the following scheme. SC maintains a list L which contains the indices of all items retrieved so far. SC also caches the records retrieved from the beginning of each session. In order to answer the kth query, SC first searches its cache. If the item does not exist in its cache, SC retrieves $DB_\pi[\pi[k]]$ and stores it in its cache. However, if the element is already cached, it randomly reads a record not present in its cache and caches it. With this approach, each record of the database might be read at most once regardless of what items are queried by users. This way, an adversary moni-

toring the database reads can obtain no information about the record being retrieved. The problem with this approach is that after $T_{threshold}$ retrievals, SC's cache becomes full. At this time a *reshuffling* is performed on DB_π to clear the cache and L. Note that since $T_{threshold}$ is a constant number independent of n, query computation and communication cost remain constant if several instances of reshuffled datasets are created offline [4], alternatively shuffling can be performed regularly on the fly which makes the query processing complexity equal to the complexity of the shuffling performed regularly. The reshuffling proposed in [3] requires $O(n\sqrt{n})$ operations for a database of size n. Iliev and Smith [8] use Benes networks to decrease this overhead to $O(n \log n)$ operations. Recently, Wang et al. [16] further reduce this overhead to $O(n)$ operations. With regards to space complexity, Asonov [3] uses an $O(n \log n)$ space for reshuffling while Iliev et al. [8] reduce the required space to $O(\log n)$ by utilizing pseudorandom permutations. Depending on system resources, each of the above variants can be used to reshuffle the database.

Algorithm 1 $read(DB_\pi, i)$

Require: DB_π, T {Threshold}, L {Retrieved Items}
1: **if** $(|L| \geq T)$ **then**
2: $DB_\pi \leftarrow$ Reshuffle DB_π using a new random permutation π;
3: $L \leftarrow \emptyset$;
4: Clear SC's cache
5: **if** $i \notin L$ **then**
6: $record \leftarrow DB_\pi[\pi[i]]$;
7: Add $record$ to SC's cache
8: $L = L \cup \{i\}$;
9: **else**
10: $r \leftarrow$ random index from $DB_\pi \backslash L$;
11: $temp \leftarrow DB_\pi[\pi[r]]$;
12: Add $temp$ to SC's cache
13: $L = L \cup \{r\}$;
14: return $record$;

We have now developed the necessary operations to enable private information retrieval. Every external entity querying the ith element of a shuffled encrypted database DB_π should query it by a *read* operation formed as $read(DB_\pi, i)$. All details regarding the shuffling, reshuffling, permutation etc. are hidden from the entity interacting with DB_π. Algorithm 1 details how the $read$ operation is performed privately and Figure 1 illustrates a snapshot of the proposed architecture. Note that algorithm 1 reads a different record per query and thus ensures each record is accessed at most once. The only communication between the server and SC for reading a record happens in lines 6 and 11. Therefore, in order to prevent side channel attacks (in all algorithms dealing with the server's database), both paths of each condition statement should take the same amount of time to execute so that the adversary cannot distinguish between the lines $6-8$ and $10-13$ of the algorithm by monitoring reads from DB_π.

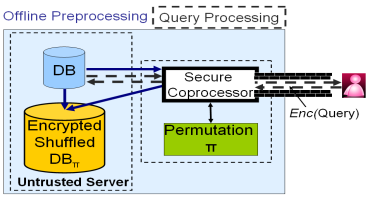

Figure 1. The Overall PIR Architecture

Theorem 1. Algorithm 1 does not leak information.

Proof: Proved by Wang et al. [16] and Asonov [3] showing the joint entropy of a sequence of queries is maximal.

4 Private Range Queries

So far we have enabled private retrieval from an untrusted server. However, we have not focused on how range queries can be evaluated privately. Section 3 enables replacing a normal database in a conventional query processing with its privacy-aware variant. However, the query processing needs to be able to utilize this new privacy-aware database as well. Building on the PIR scheme proposed in Section 3, we now discuss how range queries can be evaluated privately. The key idea is to perform the query processing on a trusted computing environment while the required data to perform the query is being privately queried from an untrusted data owner. This way, the queries can be evaluated blindly as the server does not learn which items were retrieved. Note that what distinguishes our work from the use of encrypted databases is the impossibility of blindly evaluating a sophisticated spatial query on an encrypted database without a linear scan of all encrypted items.

Due to the nature of our chosen PIR scheme, some processing is needed during the offline phase (Section 3) to generate the required data structures that should be integrated into our PIR scheme detailed above. The main objective behind designing such data structures is to enable range query processing using our encrypted shuffled database. We now focus on these data structures and how they enable range query evaluation of dynamic queries over static set of objects (private queries over public data [13]). Later in Section 5, we provide a holistic view of the entire framework and the end-to-end query resolution process.

4.1 Data Structures

During the preprocessing phase, the server needs to index the objects and perform as much preprocessing as possible in order to achieve minimum query response and communication time. Furthermore, as these data structures are privately kept at the server, they should allow efficient pruning of the space based on how they have stored object information. The challenge here is to minimize the required number of retrievals from DB for two reasons: first avoiding the I/O cost between the server and SC and secondly

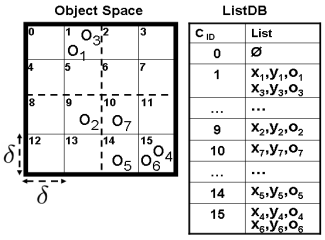

Figure 2. The Underlying Grid Structure

maximizing the time during which reshuffling is not needed (this is because the reshuffling frequency increases with query frequency). We propose to use the grid structure for indexing objects in space. The advantages of such a structure is threefold. First, the simplicity of grids as spatial index structures enables us to store the query evaluation *intelligence* on a secure coprocessor with limited memory and computation power. Secondly, our grid-based index structure allows us to prune large portions of the space without requiring object information remotely stored in DB. Finally, several studies have shown the significant efficiency of using the grid structure for evaluating range, KNN and other types of spatial queries [10, 18].

Figure 2 illustrates the underlying data structure created during the offline phase. Without loss of generality, we assume the entire area A enclosing all objects is represented by a unit square. The grid index uniformly partitions the unit square into cells with side length δ ($0 < \delta < 1$). These cells are then used to construct ***listDB*** storing the exact objects locations which are needed for generating the final result set for range queries. The $listDB$ schema, represents a flat grid and looks like $< c_{id}, list >$ where $list$ is a sequence of triplets representing objects falling in each grid.

4.2 Processing Range Queries

A range query R is defined as a rectangle of size $l \times w$ ($0 < l, w < 1$). To answer each range query using $listDB$, we must first find the set of cells R' that encloses R. R' forms a $L \times W$ rectangular grid (Figure 3) where $L = \lceil \frac{l}{\delta} \rceil$ and $W = \lceil \frac{w}{\delta} \rceil$. For now we assume the function $list(DB_\pi, c_{id})$ privately queries DB_π and performs the necessary processing to return a list of all objects enclosed in the cell c_{id}. The query $Range(R)$ can then be evaluated using a sweeping algorithm to query the cells in R' privately as shown in Algorithm 2. Since $R \neq R'$, for the range R of size $l \times w$, Algorithm 2 queries $L \times W$ cells which is linear w.r.t. R. For a uniform distribution of n objects, each cell on average contains $n \times \delta^2$ items. Therefore, the total number of items queried is $O(\alpha \times n)$ for $\alpha = L \times W \times \delta^2$ which is also linear with respect to n. Therefore, Algorithm 2 queries the smallest number of cells for any R.

Algorithm 2 Range(R)

Require: P_{ll}=R's lower left point, P_{ur}=R's upper right point;
$S \leftarrow \emptyset$;
2: **for** $(col = \lceil \frac{P_{ll}.x}{\delta} \rceil, col \leq \lceil \frac{P_{ur}.x}{\delta} \rceil, col++)$ **do**
  for $(row = \lceil \frac{P_{ll}.y}{\delta} \rceil, row \leq \lceil \frac{P_{ur}.y}{\delta} \rceil, row++)$ **do**
4: $c_{id} = (\frac{row-1}{\delta}) + row$;
$L = list(listDB, c_{id})$;
6: **for all** $o_i \in L$ **do**
if $((P_{ll}.x \leq o_i.x \leq P_{ur}.x)$ and $(P_{ll}.y \leq o_i.y \leq P_{ur}.y))$ **then**
8: $S = S \cup \{o_i\}$;
return S;

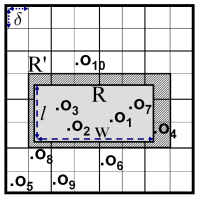

Figure 3. The Range Query R

5 SPIRAL

The integration of our PIR scheme (Section 3) and private range queries (Section 4) into SPIRAL is now straightforward. In this section, we show how they are integrated to facilitate an end-to-end query processing. The entire query processing can be divided into the following two phases.

5.1 Preprocessing

During this phase, the server creates $listDB$ using the underlying grid structure defined in Section 4. Next the secure coprocessor SC generates a random permutation π and privately shuffles $listDB$ and encrypts it. The encrypted shuffled relation $listDB_\pi$ is then written back to the server. Depending on the storage availability of SC, several instances of the encrypted and permuted datasets can be created offline and stored at the server as long as their corresponding (relatively small) permutation indices fit in SC.

5.2 Online Query Processing

During the query processing phase, users first establish a secure channel with SC through an SSL tunnel and submit their queries to the secure coprocessor. A range query q initiated by a user u_i is received by SC as $Enc_{spk}(Range(R), s_{id})$. Using Enc, each user encrypts his query with SC's public key spk and sends it along with his s_{id} (session ID). After decrypting the received range query, SC invokes algorithm 2 and privately retrieves a sequence of encrypted records from $listDB_\pi$ that contains the result set for q. Each member of the result set is first decrypted by SC and then re-encrypted with the users' public key and is transferred. Note that Algorithm 2 returns R' which might include some extra objects. Although SC can remove them before sending the final results to the user, we assume the filtering step is performed by the user mainly to

reduce SC's computation overhead. Also, as shown in Section 6, the extra results for range queries are small enough that are negligible as compared to the size of the result set.

The main success factor behind this technique is the *anonymization* of users' queries by converting them into a sequence of PIR reads from the server. Also note that the function $list()$ from Section 4 directly maps to the $read()$ function developed in Section 3. Therefore, the only interaction between SC and the server during the query processing is facilitated through the use of the private $read$ operations. Furthermore, one of the key advantages of SPIRAL is its *scalability*. The first consideration in order to make SPIRAL scalable is the simplicity inherent in Algorithm 2. However, while being simple range queries are evaluated by examining only a necessary subset of the entire space. In Section 6, we will examine the effectiveness of our range algorithm through experimental evaluation. Note that a conventional PIR scheme would have to query/transfer the entire database several times for each grid being queried. Even our efficient PIR scheme would yield completely unacceptable results if the proposed algorithm for range query evaluation had required a full scan of the dataset. This is our main motivation behind utilizing the grid index structure to enable range query processing through privately retrieving only records related to the query. The following theorem formally proves the blind evaluation of our range queries.

Theorem 2. SPIRAL blindly evaluates range queries.

Proof: We need to prove SPIRAL can blindly evaluate q (i.e., it provides u-anonymity and a-anonymity). Upon receiving each range query q , the Secure Coprocessor decomposes it into a set of retrievals $r_1, r_2, r_3, \ldots, r_s$ from the $listDB$ relation. This conversion is taking place inside SC and is not observable to anyone. As illustrated in Figure 4, sensitive query information goes through two phases. During the first phase, query information is received from the users. Based on our assumption of a secure channel between users and SC, no information is revealed to an adversary at this phase. During phase two, each q is converted to a sequence of retrievals that might be observed by an adversary. Although q is strongly correlated to r_is, we proved in Section 3 that regardless of the access sequence, the joint entropy of r_is are maximal and thus no information is revealed to the adversary. Therefore, SPIRAL is u-anonymous as SC is the only entity querying the $listDB$ relation and due to the fact that SC *anonymizes* the identity of users querying the server. It is a-anonymous as the only way of revealing the location of the query point is for the adversary to know the query window location. However, we showed this is not possible in phase one. ❑

6 Performance Evaluation

In this section we empirically examine the overall efficiency of SPIRAL. We conducted several experiments to determine the effectiveness of SPIRAL in terms of 1) the effect of system parameters such as the grid size 2) the overall response time of Algorithm 2 for a real-world dataset 3) the effect of PIR overhead on the response time.

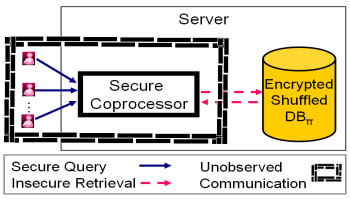

Figure 4. Secure Zones of SPIRAL

Figure 5. The 40K Restaurants Datasets

6.1 Experimental Setup

Our experiments are performed on a real-world dataset obtained from NAVTEQ (http://navteq.com) covering 40000 restaurants in an 800 by 800 mile area in central United States (Figure 5). Experiments were run on an Intel P4 3.20 GHz with 2 GB of RAM emulating a secure coprocessor. One of the key challenges in hardware-based PIR is dealing with relatively slow computation power and available storage space of secure coprocessors. For example, the IBM's PCIXCC secure coprocessor, one of the latest models in SC technology, runs at 266 MHz, and supports 80MB of main memory, Ethernet connection and a Linux OS [2]. Although such processors are significantly slower than their non-secure counterparts, they are equipped with cryptographic accelerators that significantly outperform a typical high-speed processor in performing cryptographic operations. For example PCIXCC generates roughly 900 RSA signatures per second [14], compared to our $3.20GHz$ processor in which generating the same number of signatures takes more than 5 seconds. Due to very frequent encryption (decryption) operations required in our algorithms, such accelerators can significantly improve the overall response time. Furthermore, as we show in this section, for optimal system parameters (i.e., grid and cache size), our response times are mostly in the orders of milliseconds and even a secure coprocessor that runs almost 10 times slower, would still achieve satisfactory response times. Furthermore, with regards to space, running the same experiments on a secure coprocessor would not strain its relatively limited memory due to the fact that the large index structures are being stored at the untrusted server and Algorithms 1 and 2 are designed to consume the limited amount of space available in SC. Furthermore, as illustrated in Figure 6, the

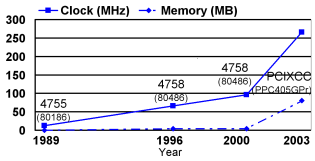

Figure 6. IBM SC Trends [5]

Table 1. Storage Requirements of SPIRAL

	Storage (Byte)	Value (KByte)
L	$\frac{log(M)}{4}$	2
Cache	$(\frac{2\times s+log(n)}{8}) \times \frac{n}{M^2} + \frac{\log(M)}{4}$	≈ 28
π (π')	$M^2 log(M)$	256

computation and storage capabilities of secure coprocessors are growing rapidly.

6.2 Space Complexity of SPIRAL

As our first set of experiments, we briefly analyze the space requirements of the whole SPIRAL framework. Table 1 summarizes the required space for storing each item in L (the list storing the index of previously retrieved items in PIR), SC's cache and the permutations π and π', respectively. The second column shows parameter values closely representing the bulk of our experiments: $n = 10^5$ (i.e., the total number of objects), $M = 2^8$ (i.e, a grid of size 2^{16}), $T_{threshold} = 1000$ (i.e, the shuffling threshold) and $s = sizeOf(double)$. As expected, the most significant space requirement is imposed by π and π'. However, their sizes are determined by the granularity of the underlying grid structure and are completely invariant of n. Furthermore, since our records are relatively small, the storage requirements of SC's cache are fairly nominal. If enough space is not available in SC's cache to store significantly larger records, we can use a variant of PIR scheme proposed by [3] which minimizes cache use by reading all $k-1$ previously accessed records for evaluating the kth query instead of using its cache to read only a single element.

6.3 The Effect of δ

Choosing the right value of δ can significantly affect the total query response time. Therefore, we now examine the effect of grid's granularity on the performance of Algorithm 2. Unless otherwise stated, we generate a set of 500 window queries of size 10×10 square kilometers randomly distributed in the entire space and measure (1) the average number of cells (i.e, $listDB$ records) being queried, hereafter being represented by C (2) the average number of excessive objects being queried, hereafter represented by E and (3) the overall query response time T in milliseconds (which in order to save the space, is drawn together with the other two parameters). Figures 7 (left) illustrates the effect of δ on these three parameters.

Obviously, it is desirable to minimize all three factors

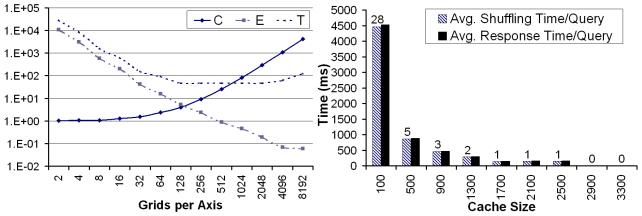

Figure 7. Effects of δ (left) and PIR (right) on Range Query

C, E and T simultaneously, to achieve optimal performance. However, very small values of C only occur while dealing with very coarse grids which clearly increases E. Alternatively, a small value of E can only be guaranteed when dealing with very fine-grained cells (i.e., large values of δ) which comes at the cost of a sharp increase in C. Therefore, we are mostly interested in values of δ corresponding to the region close to the intersection of C and E. It is easy to show that this sub-optimal point occurs at $\delta = \frac{1}{\sqrt{n}}$, where n is the total number of objects being indexed [18]. An important observation derived here is the fact that the value of δ at T's local minimum is close to its value at the sub-optimal point. More importantly, our optimal δ value is very close to the optimal δ value of $\frac{1}{128} < \delta = \frac{1}{\sqrt{n}} < \frac{1}{256}$, derived independent from the PIR module. Hence, these experiments verify that the δ parameter can be chosen in advance using the above equation. We set $\delta = 256$ for our remaining experiments. Similar results were observed for other range query sizes.

6.4 The PIR Overhead

We have so far evaluated the performance of Algorithm 2 in isolation from underlying PIR modules. We now measure the PIR overhead for private evaluation of range queries. Figure 7 (right) illustrates the average shuffling and response time per query in terms of the number of cached objects. The numbers above each pair of bars represent the total number of reshuffles occurred in evaluating queries for different cache sizes. As cache size decreases, it fills more rapidly resulting in more frequent shuffling. This is the reason behind the significant overhead imposed by PIR for very small cache sizes. For example, responding to 500 large range queries with the cache size of 100 required 28 reshuffles while the number of reshuffles drops to 5 for the cache size of 500. However, even for small available cache, the query response time is in the order of milliseconds.

7 Related Work

A large body of work in location privacy is based on k-anonymity or location cloaking (e.g., [11] and [13]). With this approach, a trusted *anonymizer* is usually in charge of receiving user's precise location information and trying to disguise it by blurring user's exact location by (for example)

extending it from a *point* location to an *area* (spatial extent) and sending a region containing several other users to the server. As we discussed in Section 1, this approach suffers from several drawbacks. For instance, the quality of service or overall system performance degrades significantly as users choose to have more strict privacy preferences

Recently, efficient utilization of secure coprocessors has broken the theoretical limitations enabling constant communication and computation time at the cost of preprocessing the data [4]. The most recent approaches have reduced the storage cost to logarithmic [8] and offline computation cost to linear [16], respectively. Our PIR scheme is built on the improvements suggested by [4, 8, 16] to achieve optimal communication cost and query response time while minimizing the storage requirements of the secure coprocessors. The challenge however, is to devise spatial query processing algorithms that can utilize such a privacy-aware server (only communicate with the server through private read operations) to process users' spatial queries.

We are aware of two studies proposing the use of PIR for location privacy. Ghinita et al. [6] utilize theoretical work on PIR instead of hardware-based approaches to enable private evaluation of nearest neighbor queries. Also, Hengartner [7] presents an architecture that uses PIR and trusted computing to protect users' location information from an untrusted server. However, the proposed architecture is not yet implemented. We have implemented SPIRAL and theoretically proved that our framework is privacy aware. We also analytically and experimentally studied range query processing in this framework and proposed to use the grid data structure as the underlying data structure for range query processing.

Most recently, Sion et al. [14] argue that using single-server computational PIR, it takes more time to process one bit of information privately than to transfer it over the network and therefore, conclude that it is more efficient to transfer the entire database to the user instead of privately retrieving an item from it and hence dismissing theoretical PIR as impractical. However, Ghinita et al. [6] have successfully implemented a location privacy scheme based on theoretical PIR while SPIRAL has also enabled an end-to-end framework to enable location privacy using practical PIR. Furthermore, evaluating sophisticated queries such as KNN queries are not always possible in the absence of the entire dataset (which cannot exist at the client side with limited available memory space and computation power).

8 Conclusion and Future Work

In this paper, we proposed SPIRAL which is to the best of our knowledge, the first study to utilize practical private information retrieval to privately evaluate range queries. We formally proved that SPIRAL guarantees perfect privacy even in the presence of the strongest adversaries and showed how it privately evaluates range queries. As part of our future work, we are currently extending SPIRAL to support KNN queries as well as private queries over moving objects utilizing the private index structure proposed in this paper.

References

[1] New york taxi strike causes longer waits. http://nytimes.com/2007/09/06/nyregion/05cnd-taxi.html.

[2] T. W. Arnold and L. van Doorn. The IBM PCIXCC: A new cryptographic coprocessor for the IBM eServer. *IBM Journal of Research and Development*, 48(3-4):475–488, 2004.

[3] D. Asonov. *Querying Databases Privately: A New Approach to Private Information Retrieval*, volume 3128 of *Lecture Notes in Computer Science*. Springer, 2004.

[4] D. Asonov and J. C. Freytag. Almost optimal private information retrieval. In *PET '02*, San Francisco, CA.

[5] B. Bhattacharjee, N. Abe, K. Goldman, B. Zadrozny, V. R. Chillakuru, M. del Carpio, and C. Apte. Using secure coprocessors for privacy preserving collaborative data mining and analysis. In *DaMoN '06*, Chicago, IL.

[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*, Vancouver, Canada.

[7] U. Hengartner. Hiding location information from location-based services. In *MDM'07*, Mannheim, Germany, 2007.

[8] A. Iliev and S. W. Smith. Private information storage with logarithm-space secure hardware. In *International Information Security Workshops*, Toulouse, France, 2004.

[9] S. Jiang, S. Smith, and K. Minami. Securing web servers against insider attack. In *ACSAC '01*, Washington, DC, USA.

[10] 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.

[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*, Boston, MA.

[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*, Seoul, Korea.

[14] R. Sion and B. Carbunar. On the computational practicality of PIR. In *NDSS '07*, San Diego, CA.

[15] S. W. Smith and D. Safford. Practical private information retrieval with secure coprocessors. Technical report, IBM, August 2000.

[16] S. Wang, X. Ding, R. H. Deng, and F. Bao. Private information retrieval using trusted hardware. In *ESORICS '06*, Hamburg, Germany.

[17] J. Warrior, E. McHenry, and K. McGee. They know where you are. In *IEEE Spectrum*, pages 20–25, 2003.

[18] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *ICDE '05*, Tokyo, Japan.