Privacy-Assured Large-Scale Navigation from Encrypted Approximate Shortest Path Recommendation

Zhenkui Shi[1,2(✉)]

[1] Department of Computer Science, City University of Hong Kong, Hong Kong, S.A.R., China
zhenkshi-c@my.cityu.edu.hk
[2] City University of Hong Kong, Shenzhen Research Institute, Shenzhen 518057, China

Abstract. As the fast-paced market of smart phones, navigation application is becoming more popular especially when traveling to a new place. As a key function, shortest path recommendation enables a user routing efficiently in an unfamiliar place. However, the source and destination are always critical private information. They can be used to infer a user's personal life. Sharing such information with an app may raise severe privacy concerns.

In this paper, we propose a practical navigation system that preserves user's privacy while achieving practical shortest path recommendation. The proposed system is based on graph encryption schemes that enable privacy assured approximate shortest path queries on large-scale encrypted graphs. We first leverage a data structure called a distance oracle to create sketch information, and we further add path information to the data structure and design three structured encryption schemes. The first scheme is based on oblivious storage. The second scheme takes advantage of the latest cryptographic techniques to find the minimal distance and achieves optimal communication complexity. The third scheme adopts homomorphic encryption scheme and achieves efficient communication overhead and computation overhead on the client side. We also evaluated our construction. The results show that the computation overhead and communication overhead are reasonable and practical.

Keywords: Private navigation · Distance oracle · Oblivious storage
PIR · Homomorphic computation

1 Introduction

As the prosperity of smart phone, location based services (LBS) are becoming very common and useful. It makes our life very convenient especially when traveling to a new place. The most obvious reason is that there is always a built-in

GPS in a smart phone. At the same time, the current smart phone is very powerful in displaying, computation and communication with a powerful processor, large memory, and storage.

However, LBS application also introduces severe privacy concerns [10,14]. Location information can be used to infer users' context and analyze users' movement patterns [8].

Navigation is one of the most popular LBS applications. The client sends the origin and the destination to the LBS server. The server responds with the shortest path or the fastest route. The user follows the route and the location information provided by the GPS to the destination. The origin and the destination here are more critical. For they introduce same privacy concerns above, may be associated with the user's personal plan [19], and more and more users concern their location privacy, privacy preserving navigation services have attracted much attention. However, there may be many challenges to build up a privacy preserving navigation system. Firstly, any path query which includes the origin and destination may disclose users' privacy. The pair of origin and destination may be personally identifiable [19]. For example, when the destination is a hospital, it may be very severe private information for the user. The second challenge is about how to compute the shortest path privately and efficiently. It is very computation intensive for computing shortest path on large graphs using breadth first search or Dijkstra's algorithm directly. Especially when considering both privacy and users' experience, the efficiency is very critical. The third challenge is how to respond privately to the users. Any plaintext information about the path will disclose the users' location privacy.

Anonymity and obfuscation techniques are common strategies. And there are some previous schemes based on anonymity and obfuscation have been proposed to preserve users' privacy [14,19]. However they always need a trusted third party between the user and the server.

In this paper, we propose a new privacy preserving path recommendation system for navigation based on private approximate queries. Our work is inspired by the latest work about graph encryption scheme based on searchable symmetric encryption (SSE) [9]. Their work is based on distance oracle [4] and SSE. However, their work may be not suitable for privacy preserving navigation. Firstly, the scheme only responds the shortest distance and the corresponding path is not known. In addition, simply adding path information in their structure, their computation may be not work. Secondly, their schemes don't protect access pattern. Whereas in privacy preserving navigation, the access pattern may be used to infer users' the source, destination, and other personal information and will cause severe information leakage.

The contributions are summarized as follows:

1. We propose three privacy preserving shortest path query schemes based on approximate distance/path oracle. To the best of our knowledge, this is the first navigation system based on approximate shortest path query.
2. The proposed schemes leverage the sketch structure. One is based on oblivious storage (OS) and the other is based on computational PIR. The third is based

on PIR and homomorphic encryption which enables computing the shortest path on the server side and achieving minimum computation cost on the client side and communication overhead. We leverage these latest crypto techniques to retrieve the sketches privately.

3. Our schemes can support large scale datasets. We evaluate our system by using real road dataset. The results show that our system provides reasonable latency, computation over head and communication overhead.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 presents the problem statement, the adversary model and some preliminaries. Section 4 introduces the path oracle, the scheme based on oblivious storage, the scheme based on computational PIR, the scheme based on PIR and homomorphic computation on the server side, and security analysis. Section 5 presents our concrete experiments, evaluation, further consideration, and optimization. Section 6 concludes the whole work.

2 Related Work

Privacy Preserving Shortest Path Computation. Some general work is about privacy preserving shortest path computation which is based on graph theory [11,18,20]. For example, Wu et al. [18] proposed to compress the next-hop routing matrices in networks such as city street maps, use symmetric PIR for indexes retrieval, and leverage garbled circuits and affine encodings for inner product evaluation. However, the scheme needs to construct a database with N^2 records which will consume large space when n is too large. Xie et al. [20] introduced a scheme based on oblivious storage and KD-tree partition. Mouratidis et al. [11] leveraged the hardware-aided PIR protocol and proposed two schemes. However, both of the preprocessing overhead and auxiliary space are unacceptable.

Privacy Preserving Shortest Distance/Path Query. Besides, there are also some schemes based on privacy preserving shortest distance/path query [9,16]. Meng et al. [9] presented three schemes based on distance oracle and structured encryption which are adaptively semantically-secure with reasonable leakage functions. Although we leverage the notion of distance oracle to build our system, our scheme can return approximate shortest path. And their constructions return only distance without path information and their constructions can't be applied in our construction directly. Simply adding path information may destroy the computation structure. And the most important is that their scheme can't protect the access pattern of the sketches which may reveal the source or the destination information. And the information is very critical for privacy preserving shortest path query and navigation. Wang et al. [16] proposed SecGDB, a secure Graph DataBase encryption scheme. SecGDB supports exact shortest distance/path query. However, the amortized time complexity is based on query history which is stored on the remote server to act as a caching resource. It may take several tens of minutes for a query without history. This is due to the auxiliary computation overhead introduced by implementing a secure Dijkstra's algorithm which is based on additively homomorphic encryption and garbled circuits. In addition, the scheme introduces a third party which doesn't collude with the server.

3 Problem Statement and Preliminaries

3.1 Problem Statement

Here, we formulate the main component of the navigation as the shortest path recommendation problem. Namely, the client holds the origin/source and the destination. It sends the request to the service provider. The service provider holds the map information including the topology.

Formally, the client holds the nodes pair (s, d) as the origin/source and destination. The service provider holds the graph information corresponding to the map $G = (V, E)$. Here, V is the node set of the graph, E is the edge set of the graph, $s \in V$ and $d \in V$. The client sends a shortest path query $q = (s, d)$ to the service provider and asks for the shortest path between s and d.

3.2 Adversary Model

In this paper, we just need to consider two parties without introducing any third party, namely, the server and a client. The server holds the data. When a client has a query for shortest path, it sends the query to the server. The server responds relevant information such as sketches to the client in our case. The server honestly follows the protocols we defined. In addition, the server is curious in learning the source, the destination, the shortest distance, the corresponding path about the client. Specifically, the client wishes to keep the information and the access pattern of retrieved sketches private. Because the access pattern may also leak critical information such as the source and the destination. In addition, we don't consider the information leakage through side channel or timing channel.

3.3 Preliminaries and Notations

$G = (V, E)$ presents a graph where V is the node set and E is the edge set. We denote $q = (u, v)$ as a shortest distance query and $pq = (u, v)$ as a shortest path query. p_{vw} is the shortest path from v to w.

Distance Oracles. Typically, a distance oracle is a type of data structure. It supports approximate shortest distance queries. An obvious construction is that one can pre-compute and store all the shortest distances of different pairs of nodes in the graph. In such a solution, the query complexity is $O(1)$. However, the storage complexity is $O(n^2)$. This is not practical for large graphs. Here, we introduce the sketch-based distance oracle. Das Sarma et al. proposed the first construction of sketch-based distance oracle in 2010 [4]. And we take the

Das Sarma's construction [4] as our study case here. Formally, the sketch-based distance oracle consists of a pair of algorithms, namely, $DO = (Setup, Query)$. The $Setup$ algorithm takes three parameters, a graph G, an approximation factor α and an error bound ϵ. Finally, it outputs an oracle $\Omega_G = \{Sk_v\}_{v \in V}$ where Sk_v is the sketch which includes pairs (w, δ), where $w \in V$ and $\delta = dist(v, w)$. The $Query$ algorithm takes two parameters, the oracle Ω_G and a shortest distance query $q = (u, v)$ and $d := Query(\Omega_G, u, v)$. DO is (α, ϵ)-correct if for all graphs G and all queries q, $Pr[dist(u, v) < d < \alpha dist(u, v)]$. For the $Query(u, v)$, the query algorithm firstly finds the common nodes set I between Sk_u and Sk_v, and returns $s \in I$ such that $dist(u, s) + dist(s, v)$ is the minimum. If there is no common node, then it returns $\perp$.

4 Our Constructions

In this section, we illustrate our construction for privacy preserving navigation.

4.1 Path Oracle Setup

The sketches created by distance oracles provide our effective data structure for shortest distance queries. It greatly saves the storage. For it is pre-computed and can save computation during runtime. However, only providing the shortest distance is not enough in some cases. In many applications, providing the shortest path is essential. And navigation is one of the cases.

Definition 1 (*Path Oracle*). *A sketch-based path oracle $PO = (Setup, Query)$. The Setup algorithm takes three parameters, a graph G, an approximation factor α and an error bound ϵ. Finally, it outputs an oracle $\Omega_G = \{Sk_v\}_{v \in V}$ where Sk_v is the sketch which includes triples (w, δ, p_{vw}), where $w \in V$, $\delta = dist(v, w)$ and p_{vw} is the shortest path from v to w. The Query algorithm takes two parameters, the oracle Ω_G and a shortest path query $q = (u, v)$ and $(d, p) := Query(\Omega_G, u, v)$ where the pair (d, p) is the shortest distance and path returned by the Query. PO is (α, ϵ)-correct if for all graphs G and all queries q, $Pr[dist(u, v) < d < \alpha dist(u, v)] \geq 1 - \epsilon$.*

We can take advantage of the data structure and construct our path oracles. Similarly, we define a path oracle is a type of data structure which supports approximate shortest path queries. Here, we can construct the path oracle by adding path information to the sketches created by the distance oracle. Suppose that $\Omega_G = (Sk_{v_1}, Sk_{v_2}, ..., Sk_{v_n})$ is the collection of sketches created by a distance oracle. Each sketch Sk_{v_i} consist of λ pairs of $\{(w_z, \delta_z)\}_{0 \leq z \leq \lambda - 1}$, where w_z is a node and $\delta_z = dist(v_i, w_z)$.

Based on the structure, we add path information for each sketch. Without loss of generality, $DO = (Setup, Query)$ is a sketch-based distance oracle. It creates a collection of sketches $\Omega_G = (Sk_{v_1}, Sk_{v_2}, ..., Sk_{v_n})$. We define a path oracle as $PO = (Setup, Query)$. It inherits the sketch-based data structure created by DO. For each sketch Sk_{v_i}, it consists of triple $\{(w_z, \delta_z), p_{v_i w_z}\}_{0 \leq z \leq \lambda - 1}$, where w_z is a node, $\delta_z = dist(v_i, w_z)$, and $p_{v_i w_z}$ is the information about the shortest path from v_i to w_z. $p_{v_i w_z}$ can be pre-computed through classical shortest path algorithm such as Dijkstra's algorithm.

Given a query $q = (u, v)$, it finds the collection common nodes I between Sk_u and Sk_v. Then it find the node $s \in I$ such that $mind = dist(u, s) + dist(s, v)$ is the minimum and return $(mind, pi)$ where $p = p_{us} + p_{sv}$. From the construction, we can get the following lemma.

Lemma 1 (*Path Oracle Construction*). *The construction of path oracle above is (α, ϵ)-correct. For all graphs G and all queries q, $Pr[dist(u, v) < \dot{d} < \alpha dist(u, v)] \geq 1 - \epsilon$.*

For the path oracle construction, we setup the construction based on the Das Sarma et al. oracle [4] as our study case. There is sub routine *Sketch* used to generate a collection of sketches $(Sk^i_{v_1}, ..., Sk^i_{v_n})$ where i is the ith call to the sub routine. To create the sketch $Sk^i_{v_j}$, the sub routine samples $r + 1$ sets of nodes $S_0, S_1, ..., S_r$ where the size of S_i is 2^i where $r = \lfloor logn \rfloor$. For each node vj, for all these sets S_i, it computes $w_k, \delta i, p_{w_k, v_j}$ where w_k is the closest node to v_j, $\delta i = dis(v_j, w_k) = dis(v_j, S_i)$ and p_{w_k, v_j} is the corresponding shortest path. After calling the sub routine σ times, it collects σ collections of $(Sk^i_{v_1}, ..., Sk^i_{v_n})_{i \in [\sigma]}$ where $\sigma = \widetilde{\theta}(n^{2/(\alpha+1)})$. The final sketch $Sk_{v_j} = \bigcup_{i=1}^{\sigma} Sk^i_{v_j}$. And finally, it outputs the path oracle as $\Omega_G = (Sk_{v_1}, ..., Sk_{v_n})$.

4.2 A Scheme Based on Oblivious Storage

In this section, we illustrate one of our proposed scheme with strong privacy grantees. The scheme adopts the latest practical techniques, oblivious storage for privacy preserving sketch retrieval.

The proposed architecture is illustrated in Fig. 1. It consists of a client and a server. The server provides oblivious storage service.

Oblivious Storage. Oblivious storage (OS) is defined as a related notion to Oblivious RAM (ORAM). Bonel et al. proposed the notion as a practical implementation of ORAM in [3]. ORAM can provably hide all access patterns. In OS model, clients can outsource the data to the cloud or third parties. when running applications, OS can keep the data from disclose. In our case, we can private retrieval the sketch privately via OS. OS provides strong access pattern protection. The origin and the destination can be well protected.

Typically, there are two kinds of solutions for OS: the Square-Root and the Hierarchical solutions. We adopt the OS scheme based on Path-ORAM [15]. It has non-trivial online latencies on moderate size datasets and doesn't involve any system unavailable time which provides excellent worst-case performance guarantee on large datasets [20].

As illustrated in Fig. 1, the data is stored as a binary tree in the server. In our circumstance, there are N sketches and we pad each sketch to the same length as a block and we call it sketch block. And the hight of the binary tree is $L = \lceil logN \rceil$. Each node of the tree is called a bucket and each bucket contains Z blocks where Z is a small constant (e.g., 3–5). These Z blocks are either real

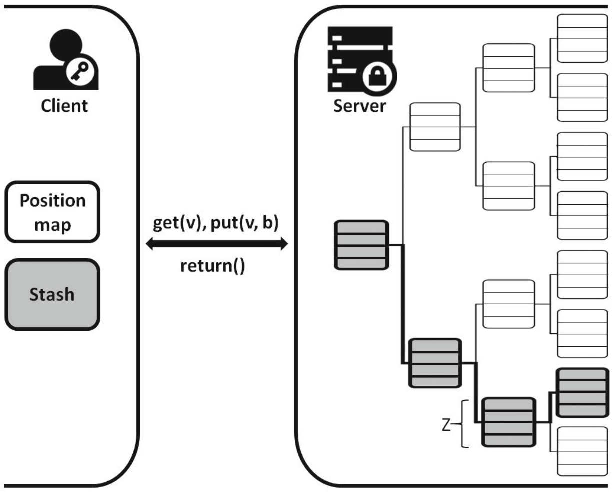


Fig. 1. The architecture based on oblivious storage.

data blocks or dummy blocks. There are two data structures stored on the client, a position map and a stash as illustrated in Fig. 2. The position map maps each of the N sketch blocks to one of the 2^L leaves independently and uniformly, e.g., $i := PosMap[x]$ means that sketch block x is associated leaf i and block x resides in some bucket in the path from the root node to the leaf x, or in the stash. The position map is refreshed as blocks are accessed and remapped.

Oblivious Storage Initialization. The client initializes each sketch as an encrypted block with the same length and sends all the encrypted blocks to the server. The blocks are stored in a binary tree on the server.

On the client side, there is a position map and a stash. The functionality of position map is like an index. Suppose there are N sketches. Each sketch is assigned with an ID. The position map can be used to map each of the N sketches to one of the 2^L leaves which are independently and uniformly at random. The stash is used to store sketch blocks during accessing and overflowed sketch blocks from the binary tree.

On the server side, all the sketch blocks are stored in the binary tree. The height of the tree is $L = \lceil logN \rceil$. Each node of the tree is considered as a bucket which contains Z blocks. Z is independent of N. There are many dummy blocks to make sure that all the buckets appear full.

Private Sketch Retrieval. When the client needs to query a shortest path $pq = (Src, Dst)$ where Src is the source node and Dst is the destination node, the client creates two retrieval requests for the source and the destination respectively. The two requests are in the same form. We take the Src sketch Sk_{Src} retrieval request as an example.

To retrieval sketch block Sk_{Src}, the client and server should take the following steps.

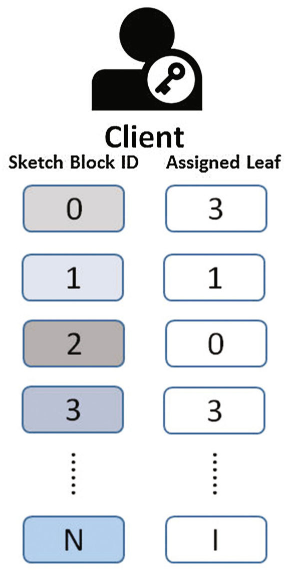


Fig. 2. The data structure on the client side.

1. The client checks the position map and finds the leaf $i := PosMap[Src]$, remaps the sketch block Sk_{Src} to a new random position. Then, it sends a read request to the server. The form of the request is like ($read$, $PosMap[Src]$, $None$).
2. The server reads all the buckets along the path from the root node to the leaf node $PosMap[Src]$ and sends all the buckets including $(L+1)*Z$ blocks to the client.
3. The client decrypts all the $(L+1)*Z$ blocks, recovers the real sketch blocks, stores in the stash, identifies the sketch block Sk_{Src} and initializes to the sketch format. The dummy blocks are discarded.
4. The client re-encrypts the sketch blocks in the stash with fresh randomness and sends a writing request to the server. The writing path is the same as the read path. The form of the writing request is like $(writing, PosMap[Src], data*)$. The sketch blocks in $data*$ are put as close to the leaves as possible while the buckets where they are must be in the same path as their assigned leaves. If buckets including the root bucket are full of sketch blocks, the remaining sketch blocks will stay in the stash.
5. The server writes the data along the path from the root node to the leaf node $PosMap[Src]$.

In the phase of private sketch retrieval, the client should send two pairs of reading/writing requests to the server and the server completes 2 pairs of reading access and writing access operations, one for the source's sketch and one for the destination's sketch.

Algorithm 1. *ComputeShortestPath*: compute shortest path

```
Require: Private key $SK$; source sketch block $ssb$; destination sketch block $dsb$.
Ensure: shortest distance and path $\{sdis, spath\}$.
 1: $ssk \leftarrow InitializeSketch(SK, ssb)$;
 2: $dsk \leftarrow InitializeSketch(SK, dsb)$;
 3: $std \leftarrow ssk + dsk$;
 4: $sdis = \perp$;
 5: for $v_j \in \{v_1, \cdots, v_m\}$ do
 6:     if $std[v_j].dis < sdis$ then
 7:         $sdis \leftarrow std[v_j].dis$;
 8:         $spath \leftarrow std[v_j].path$;
 9:     end if
10: end for
11: return $sdis, spath$;
```

Shortest Path Computation. After retrieving both the origin's sketch and destination's sketch, the client can compute the shortest path locally. The algorithm is illustrated as Algorithm 1. The client firstly initializes the source sketch and the destination sketch as step 1–2. Then the step 3 will add the distances if the nodes are both in the source sketch and the destination sketch. We say these nodes are common nodes and form a set $v_1, v_2, ..., v_m$. Then the step 4–10 of the algorithm find the minimal distance the corresponding common node where $std[v_j].dis$ is the joint distance of the distance from the source to v_j and the distance from v_j to the destination. And the shortest path $std[v_j].path$ is joint of the two paths. If there is no common node, then the algorithm return $\perp$.

4.3 A Scheme Based on PIR

In this section, we illustrate the privacy preserving scheme based on computational PIR. And PIR can provide robust privacy preserving which is equivalent to oblivious RAM storage. The scheme is inspired by the latest privacy preserving techniques XPIR [2] and its application about private media consumption [6]. Although previous work argued that traditional PIR protocols involve considerable computation and/or communication overheads on sizable datasets. Also, some practical PIR scheme may introduce an off-the-shelf hardware secured co-processor and raise a hardware-aided PIR protocol [17]. By leveraging the usable PIR, [11] proposed two schemes for shortest path queries. But they need unacceptable preprocessing and auxiliary space.

PIR for Private Sketch Retrieval. There are two typical PIR protocols, namely, computational PIR [2] and information theoretic PIR [6]. Due to the fact that the information theoretic PIR protocol needs at least k servers where $k \geq 1$. And any of the servers should not collude. This is a strict setting. Whereas the model for computation PIR is simple and there is only one computationally bound server in computational PIR protocol. And the current crypto scheme for computational PIR is efficient enough [2]. We can realize the privacy preserving

Query Construction to retrieve sketch Sk_{v_j} (the client):

```
1. For $i$ from 1 to $n$,
   -if $i \neq j$, create a random encryption of zero
   -if $i = j$, create a random encryption of one
2. Send the $q = (pk, q_1, ..., q_{n)}$ to the server;
```

Reply Construction (the server): The server performs following computations

```
1. For $i$ from 1 to $n$,
   -    $R = Sum_{i=1}^{n} q_i * Sk_{v_i}$
2. Return $R$
```

Sketch Extraction (the client):

```
1. Client decrypts the reply $R$ and recover the sketch $Sk_{v_j}$.
```

Fig. 3. The workflow for computational PIR based sketch retrieval.

sketches retrieval through the computational PIR. In our design based on computational PIR, all the sketches form a library as a database. Then the library is $L = \{Sk_{v_1}, Sk_{v_2}, ..., Sk_{v_N}\}$ where N is the number of all nodes in the road network. Suppose that l is the bit length of sketch Sk_{v_i} and l_0 is the bit length that can be used for homomorphic operations. If $l > l_0$, then the sketch should be split into chunks of l_0 bits as $Sk_{v_i} = \{Sk_{v_i,0}, Sk_{v_i,1}, ..., Sk_{v_i,l/l_0}\}$. If $l \leq l_0$, each sketch can be padded as one element entry in the database. The padding scheme just makes sure that one sketch should be contained in only one element entry in the database. This rule guarantees that the client can get the right sketch through one query. And the client should know the mapping of the sketches and the corresponding entries. There are three steps in the traditional computational PIR protocol. For simplicity, we consider each sketch as one element entry. And the workflow can be illustrated in Fig. 3.

Shortest Path Computation. In our scheme, the shortest path computation is on the local side of the client. It shares the same process with the scheme based on oblivious storage as Algorithm 1.

4.4 A Communication Efficient Scheme Based on PIR

In this sub section, we describe a communication efficient scheme. In this scheme, we leverage the PIR to retrieve the source sketch and the destination sketch. However, the server doesn't send the sketch to the client directly. The server computes the shortest path locally and return the result to the client. The scheme includes the following steps.

Setup. Compared with the previous schemes, we need a more structured data structure. We set up a data structure illustrated in Fig. 4. To make the later

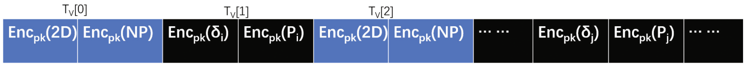


Fig. 4. The sketch hash table for a sketch.

computation effective, we adopt the same homomorphic encryption scheme $HE = (Gen, Enc, Dec, Eval)$ with the following PIR protocol and (pk, sk) is the public/secret key pair. In addition, it makes use of a family of universal hash function H. Given Ω_G, ε, let D be the maximum distance over all the sketches and S be the maximum sketch size. It samples a hash function $h \overset{\$}{\leftarrow} H$ with domain V and co-domain $[t]$, where $t = 2 \cdot S^2 \cdot \varepsilon^{-1}$.

For each node $v \in V$, it creates a sketch hash table T_v as Fig. 4. Each table has t cells. Each cell stores two parts, one for the shortest distance and the other for the shortest path. For example, $\{(w_z, \delta_z), p_{v_i w_z}\} \in SK_v$, $T_v[h(w_z)].dis \leftarrow Enc_{pk}(\delta_z)$, and $T_v[h(w_z)].path \leftarrow Enc_{pk}(p_{v_i w_z})$. For all the remaining locations, $T_v[h(w_z)].dis \leftarrow Enc_{pk}(2D)$ and $T_v[h(w_z)].path \leftarrow Enc_{pk}(NP)$ where NP is randomized path information.

Private Sketch Hash Table Identification. This phase is very similar with the scheme based on PIR. We leverage the idea of PIR to identify the source sketch hash table and the destination sketch hash table obliviously. First, the client construct two queries for the source and the destination respectively. The process is same with the PIR protocol. After receiving the queries, the server does the similar computation as PIR protocol. However, it just does the computation and doesn't return the result. After the computation, the server gets the encrypted versions of the source sketch hash table T_s and the destination sketch hash table T_d.

Private Shortest Path Computation. Now the server get the encrypted sketch hash tables T_s and T_d. Then, the server computes the homomorphic additions over each cell and creates the new hash table T_r, where $T_r[i].dis = T_s[i].dis + T_d[i].dis$ and $T_r[i].path$ is the concatenation of $T_s[i].path$ and $T_d[i].path$. Then, the server needs to find the minimum distance and the corresponding path from the new encrypted hash table T_r.

Without loss of generality, we assume that $0 \leq T_r[i].dis, T_r[j].dis < 2^l$ where $i, j \in \{0, ..., t-1\}$. Then, let $T_{i,j} = 2^l + T_r[i].dis - T_r[j].dis$. Let $\widetilde{T_{i,j}}$ be the most significant bit of $T_{i,j}$. And $\widetilde{T_{i,j}}$ can be computed through the following equation

$$\widetilde{T_{i,j}} = 2^{-l} \cdot [T_{i,j} - (T_{i,j} \ mod \ 2^l)]. \tag{1}$$

In the equation, the subtraction sets the least significant bits to zero, and the multiplication shifts the most significant bit down. Through $\widetilde{T_{i,j}}$, we can compute the minimum distance $T_{i,j}.dis$ and the corresponding path $T_{i,j}.path$ as follows:

$$T_{i,j}.dis = \widetilde{T_{i,j}} \cdot [T_r[i].dis - T_r[j].dis] + T_r[j].dis. \tag{2}$$

$$T_{i,j}.path = \widetilde{T_{i,j}} \cdot [T_r[i].path - T_r[j].path] + T_r[j].path. \tag{3}$$

Algorithm 2. *ComputeShortestPath*: compute shortest path on the server side

```
Require: Private key $SK$; the hash table $T_r$.
Ensure: shortest distance and path $\{sdis, spath\}$.
1: $sdis \leftarrow T_r[0].dis$;
2: $spath \leftarrow T_r[0].path$;
3: for $n = \{1 : t-1\}$ do
4:     $sdis_i \leftarrow sdis$;
5:     $spath_i \leftarrow spath$;
6:     $sdis_j \leftarrow T_r[n].dis$;
7:     $spath_j \leftarrow T_r[n].path$;
8:     follow the secure two-party computation protocol and compute $\widetilde{T_{i,j}}$ using equation1;
9:     compute $sdis$ using equation 2;
10:    compute $spath$ using equation 3;
11: end for
12: return $(sdis, spath)$;
```

However, there is only one challenge to compute these values over encrypted domain. There is no effective homomorphic operation to compute $(T_{i,j}\ mod\ 2^l)$ over encrypted domain. Here, we try to minimize the computation overhead of the client and introduce a third party, cryptographic service provider (CSP). This similar architecture is widely applied in many scenario [7,12]. Our adversary model follows the similar setting, namely, the server doesn't collude with the CSP. Then, the server and the CSP follow an secure interactive two-party computation protocol to compute $(T_{i,j}\ mod\ 2^l)$ [5,13].

In this architecture, the client and the server uses the homomorphic scheme provided by the CSP. The queries is created by the client through the public key of the CSP. The server runs the PIR protocol under the same homomorphic scheme. Then, the server and the CSP follow the same secure two-party computation protocol as [5].

After that, we can apply a recursive algorithm to compute the shortest distance and the corresponding path. The algorithm is illustrated as Algorithm 2.

After the server computing $sdis$ and $spath$, the server generates two random masks $smask$ and $pmask$, computes $(sdis + smask)$ and $(spath + pmask)$, encrypts $smask$ and $pmask$ using the public key of the client and sends $(sdis + smask)$ and $(spath+pmask)$ to the CSP. The CSP decrypts $(sdis+smask)$ and $(spath + pmask)$, encrypts using the public key of the client, and sends the new encrypted $(sdis + smask)$ and $(spath + pmask)$ to the server. The server sends encrypted $(sdis+smask)$, $(spath+pmask)$, $smask$ and $pmask$ to the client. The client decrypts them and gets the final results through the path information.

5 Evaluation

In this section, we present our experiment and evaluation of our schemes on several real road networks. We implemented the path oracle based on the Das

Sarma et al. distance oracle algorithm [4] and leverage existing building block such oblivious storage scheme based on path ORAM and XPIR [2]. We evaluate the schemes on Microsoft Azure. For simplicity and efficiency, we use the Microsoft Azure virtual machines to simulate the server and the client. We select the standard D4s V3 with 4 cores 16 GB RAM and 30 GB SSD as the server and select the standard D4s V2 with 4 cores 14 GB RAM and 30 GB SSD as the client.

5.1 Evaluation of Path Oracle

We leverage the boost graph library and implement the path oracle in C++11. We also take the real road datasets from [1] as our study case. To evaluate our scheme, we consider the real road network as Table 1. The sketch size is very critical in path oracle. It will affect the communication overhead in both of our schemes and computation overhead in the scheme based on computation PIR. The parameter σ defines the times to call the sub routine to create sketches. We set $\sigma = 3$ as the base [9]. For each node, the oracle program will create a sketch file named with the node ID in a local directory. In the sketch file, a line is an element in the sketch. The format of the file is illustrated as Fig. 5. Suppose that the file is the sketch file of node v_s. Fields like $(v_i, dist(v_s, v_i), Path(v_s, v_i))$ are separated by a comma.

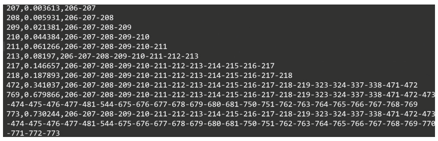
And we run our program to create all the sketch files for different road network. Besides, we also change the times to call the sub routine and check the sizes changing over the times. For the max size of sketch files will affect later processing such as the block size of oblivious storage. We present the max size of sketch files as Fig. 6. In the figure, CA(21048, 21693) means California Road Network which has 21048 nodes and 21693 edges. Through the figure, we can see that the max size is about 6 KB when calling the subroutine 3 times. Even calling the subroutine 18 times, the max size is about 24 KB. In the scheme based on oblivious storage, the size will affect the stash size on the client side and communication overhead. The result shows that the growth rate of max size is all linear or sub linear over times to call the sub routine and the scale of the graph such as the number of nodes and the number of edges.

Table 1. Real road dataset

Name of road network	Number of nodes	Number of edges
California Road Network (CA)	21048	21693
City of San Joaquin County Road Network (TG)	18263	23873
City of Oldenburg Road Network (OL)	6105	7035

5.2 Evaluation of the Scheme Based on OS

To evaluate the scheme based on OS, we focus on the metrics about the query time and communication overhead. It depends mainly on the blocks to transfer and levels to process. We should consider the level of the binary tree $L = \lceil logN \rceil$ and the parameter of OS Z. The level is associated with the number of nodes N. For security reason, we consider the cases when $Z = 4$ [15]. On the client side, we set the security parameter $\lambda = 128$ which means that the failure probability less than $2^{-\lambda}$. And the persistent local storage sizes for the max stash are 147 when $Z = 4$. In addition, the client also requires transient storage used to cache the fetched data from a path on the server. The storage size is $Z * log_2 N$ blocks.



```
207,0.003613,206-207
208,0.005931,206-207-208
209,0.021381,206-207-208-209
210,0.044384,206-207-208-209-210
211,0.061266,206-207-208-209-210-211
213,0.08197,206-207-208-209-210-211-212-213
217,0.146657,206-207-208-209-210-211-212-213-214-215-216-217
218,0.187893,206-207-208-209-210-211-212-213-214-215-216-217-218
472,0.341037,206-207-208-209-210-211-212-213-214-215-216-217-218-219-323-324-337-338-471-472
769,0.679866,206-207-208-209-210-211-212-213-214-215-216-217-218-219-323-324-337-338-471-472-473
-474-475-476-477-481-544-675-676-677-678-679-680-681-750-751-762-763-764-765-766-767-768-769
773,0.730244,206-207-208-209-210-211-212-213-214-215-216-217-218-219-323-324-337-338-471-472-473
-474-475-476-477-481-544-675-676-677-678-679-680-681-750-751-762-763-764-765-766-767-768-769-770
-771-772-773
```

Fig. 5. The file format of sketch file.

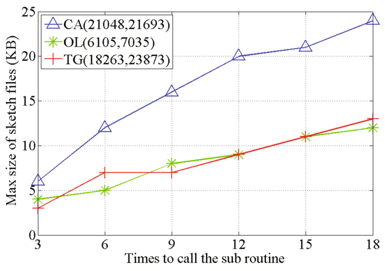


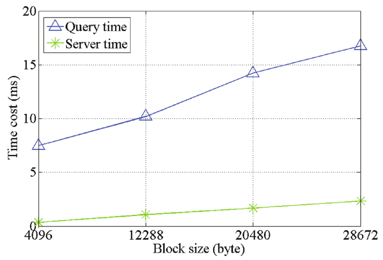
Fig. 6. The max size of sketch files changes over times to call the subroutine.

The time cost is illustrated in Fig. 7. Through the figure, we can see that there is little computation overhead on the server side. The query time is almost for the communication overhead. This is consistent with the oblivious storage protocol. Through the Fig. 7(a), the growth rate of the query time and the time of the server processing is almost linear with the growth rate of the block size. Through the Fig. 7(b), the fluctuation including the server processing time and the whole query time is sub linear over the number of nodes. It can be presented as $f(x) = \alpha \lceil logN \rceil$ where N is the number of nodes and α is a coefficient. This makes the scheme more scalable. And the total time cost is reasonable.

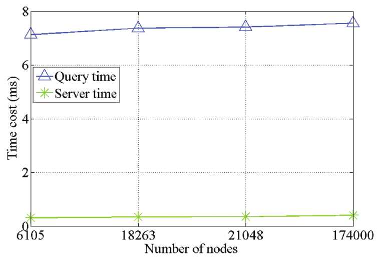
The communication overhead is about $Z * \lceil logN \rceil$. Both the computation overhead and latency are practical.

5.3 Evaluation of the Scheme Based on Computational PIR

To evaluate the scheme based on computational PIR, we focus on the metrics about the query time, computation overhead including the time cost to decrypt results, etc. We pad all the sketches and evaluate different scales of databases. We set the security parameter as 120 for the Ring-LWE encryption. We evaluate the datasets when the number of Nodes $N = 6105, 18263, 21048, 42096$, and the file size $f = 4$ KB, 8 KB, 12 KB, 24 KB respectively. The evaluation result is presented in Table 2. Our result focuses on the decrypting time and the query round-trip time (RTT). The RTT grows to 18.6123 s as the number of files increased to 42096 and the file size increased to 24 KB. The client side also should take more time to decrypt the result file.



(a) Time cost over block size



(b) Time cost over node number

Fig. 7. Time cost for server and query respectively

Table 2. Evaluation of private sketch retrieval based on computational PIR

Number of files	Max file size of sketch files	Decrypting time (S)	Query RTT (S)
6105	4 KB	0.337939	0.400975
18263	8 KB	1.47355	1.60825
21048	12 KB	2.70837	2.813172
42096	24 KB	18.4685	18.6123

To further investigate the relation ship between the max file size, the query RTT, and computation cost, we fix the number of files, increase the max file sizes and evaluate the metrics. This is due to that the times to call the sub routine also affect the max file size and query correctness. We fix the file number as $N = 21048$. The evaluation result is shown as Fig. 8. Through the results, we can see that both the decrypting time and query RTT grow over the max file size increasing. The query RTT is from 1.8 s to 8.8 s when the max file is from 8 KB to 28 KB respectively.

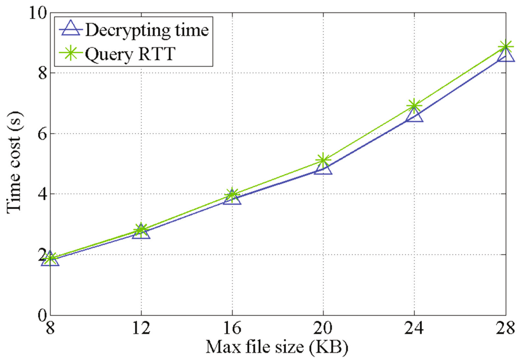


Fig. 8. The time cost of PIR changes over the max file sizes when the number of files is 21048.

6 Conclusion

In this paper, we propose three schemes for approximate shortest path query which aims to protect users' privacy and achieve shortest path query. Our designs provide robust privacy guarantees by leveraging the latest cryptographic progress on oblivious storage and PIR respectively. The first two schemes don't need any third party. In the third scheme, most of the computation is on the server side and can achieve efficient communication overhead. Security and evaluation are conducted to show our design can achieve strong security guarantees with practical performance.

Acknowledgement. This work was supported by the Natural Science Foundation of China (Project No. 61572412) and the Microsoft Azure Research Grant.

References

1. Real datasets for spatial databases: Road networks and points of interest. https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm
2. Aguilar-Melchor, C., Barrier, J., Fousse, L., Killijian, M.-O.: XPIR: private information retrieval for everyone. In: Proceedings of PETS (2015)
3. Boneh, D., Mazieres, D., Popa, R.A.: Remote oblivious storage: making oblivious ram practical (2011)
4. Das Sarma, A., Gollapudi, S., Najork, M., Panigrahy, R.: A sketch-based distance oracle for web-scale graphs. In: Proceedings of ACM WSDM, pp. 401–410 (2010)
5. Erkin, Z., Franz, M., Guajardo, J., Katzenbeisser, S., Lagendijk, I., Toft, T.: Privacy-preserving face recognition. In: Goldberg, I., Atallah, M.J. (eds.) PETS 2009. LNCS, vol. 5672, pp. 235–253. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03168-7_14
6. Gupta, T., Crooks, N., Mulhern, W., Setty, S.T., Alvisi, L., Walfish, M.: Scalable and private media consumption with popcorn. In: Proceedings of NSDI (2016)

7. Kim, S., Kim, J., Koo, D., Kim, Y., Yoon, H., Shin, J.: Efficient privacy-preserving matrix factorization via fully homomorphic encryption. In Proceedings of ACM ASIACCS (2016)
8. Krumm, J.: A survey of computational location privacy. Pers. Ubiquit. Comput. **13**(6), 391–399 (2009)
9. Meng, X., Kamara, S., Nissim, K., Kollios, G.: Grecs: graph encryption for approximate shortest distance queries. In: Proceedings of ACM CCS (2015)
10. Microsoft Trustworthy Computing. Location based services and privacy (2011). http://www.microsoft.com/en-us/download/confirmation.aspx?id=3250
11. Mouratidis, K., Yiu, M.L.: Shortest path computation with no information leakage (2012)
12. Nikolaenko, V., Ioannidis, S., Weinsberg, U., Joye, M., Taft, N., Boneh, D.: Privacy-preserving matrix factorization. In: Proceedings of ACM CCS (2013)
13. Rahulamathavan, Y., Phan, R.C.-W., Chambers, J.A., Parish, D.J.: Facial expression recognition in the encrypted domain based on local fisher discriminant analysis. IEEE Trans. Affect. Comput. **4**(1), 83–92 (2013)
14. Shin, K.G., Ju, X., Chen, Z., Hu, X.: Privacy protection for users of location-based services. IEEE Wirel. Commun. **19**(1), 1536–1284 (2012)
15. Stefanov, E., Van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., Devadas, S.: Path ORAM: an extremely simple oblivious RAM protocol. In: Proceedings of ACM CCS, pp. 299–310 (2013)
16. Wang, Q., Ren, K., Du, M., Li, Q., Mohaisen, A.: SecGDB: Graph Encryption for Exact Shortest Distance Queries with Efficient Updates. In: Kiayias, A. (ed.) FC 2017. LNCS, vol. 10322, pp. 79–97. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70972-7_5
17. Williams, P., Sion, R.: Usable pir. In: Proceedings of NDSS (2008)
18. Wu, D.J., Zimmerman, J., Planul, J., Mitchell, J.C.: Privacy-preserving shortest path computation. arXiv preprint arXiv:1601.02281 (2016)
19. Xi, Y., Schwiebert, L., Shi, W.: Privacy preserving shortest path routing with an application to navigation. Pervasive Mob. Comput. **13**, 142–149 (2014)
20. Xie, D., Li, G., Yao, B., Wei, X., Xiao, X., Gao, Y., Guo, M.: Practical private shortest path computation based on oblivious storage. In: Proceedings of IEEE ICDE, pp. 361–372 (2016)