Processing Private Queries over Untrusted Data Cloud through Privacy Homomorphism

Haibo Hu, Jianliang Xu, Chushi Ren, Byron Choi

Department of Computer Science, Hong Kong Baptist University
Kowloon Tong, Hong Kong SAR, China
{haibo,xujl,csren,bchoi}@comp.hkbu.edu.hk

***Abstract*—Query processing that preserves both the data privacy of the owner and the query privacy of the client is a new research problem. It shows increasing importance as cloud computing drives more businesses to outsource their data and querying services. However, most existing studies, including those on data outsourcing, address the data privacy and query privacy separately and cannot be applied to this problem. In this paper, we propose a holistic and efficient solution that comprises a secure traversal framework and an encryption scheme based on privacy homomorphism. The framework is scalable to large datasets by leveraging an index-based approach. Based on this framework, we devise secure protocols for processing typical queries such as k-nearest-neighbor queries (kNN) on R-tree index. Moreover, several optimization techniques are presented to improve the efficiency of the query processing protocols. Our solution is verified by both theoretical analysis and performance study.**

I. Introduction

Cloud computing has recently emerged as a new platform for deploying, managing, and provisioning large-scale services through an Internet-based infrastructure. Successful examples include Amazon EC2, Google App Engine, and Microsoft Azure. As a result, hosting databases in the cloud has become a promising solution for Database-as-a-Service (DaaS) and Web 2.0 applications.

In the cloud computing model, the data owner outsources both the data and querying services to the cloud. The data are private assets of the data owner and should be protected against the cloud and querying client; on the other hand, the query might disclose sensitive information of the client and should be protected against the cloud and data owner. Therefore, a vital concern in cloud computing is to protect both data privacy and query privacy among the data owner, the client, and the cloud. The social networking service is one of the sectors that witness such rising concerns. For example, in Fig. 1 user Cindy wants to search an online dating site for friends who share with her similar backgrounds (e.g., age, education, home address). While the site or the data cloud should not disclose to Cindy personal details of any user, especially those sensitive ones (e.g. home address), Cindy should not disclose the query that involves her own details to the site or the cloud, either.

More critical examples exist in business sectors, where queries may reveal confidential business intelligence. For example, a retail business plans to open a branch in a district. To calculate the target customer base, it needs to query the demographic data of that district, which the data owner has outsourced to a data cloud. While personal details in the demographic data should not be disclosed to the outsourcing cloud or the business, the district name in that query should not be disclosed to the cloud or data owner, either.

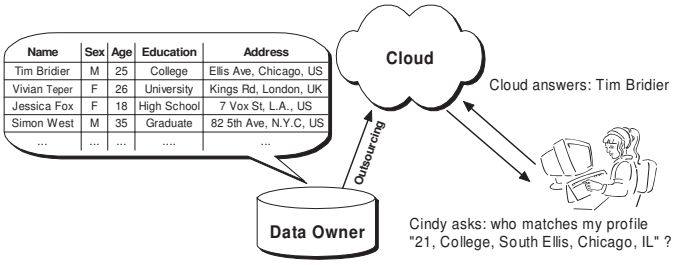


Fig. 1. Mutual Privacy Protection in Online Friend Matching

It is also noted that the cloud computing model worsens the consequence of privacy breaches in the above scenarios as a single cloud may host querying services for many data owners. For example, two queries from the same user, one on local clinic directory and another on anti-diabetic drugs, together give a higher confidence that the user is probably suffering from diabetes. All the above concerns call for a query processing model that preserves both data privacy and query privacy among the data owner, the client, and the cloud. The data owner should protect its data privacy, and does not reveal any information beyond what the query result can imply. On the other hand, the client should protect its query privacy so that the data owner and the cloud know nothing about the query, and is therefore unable to infer any information about the client.

Unfortunately, existing privacy-preserving query processing solutions are not sufficient to solve this new problem arising in the cloud model. Most research work in the literature addresses data privacy or query privacy separately. For example, generalization techniques have been proposed to protect data privacy by hiding quasi-identifier attributes and avoiding the disclosure of sensitive information [3], [29]. Similar techniques are proposed for query privacy on both relational data and spatial data [16], [12], [13], [18]. Only very few, such as the $Casper^*$ framework [5], consider data and query privacy as a whole. Furthermore, generalization-based solutions like the $Casper^*$ still disclose the data or query in a coarser and imprecise form. Not much research work addresses the *unconditional* privacy required for this problem. Although some encryption schemes

are proposed to protect the data hosted on the outsourcing server [1], [27], [20], [31], [28], they cannot be adopted in this problem for several reasons. First, accurate query processing on encrypted data is difficult, if not impossible at all. Most existing encryption schemes only support some specific queries. For example, space transformation (e.g., a space filling curve) used in [20] only supports approximate kNN queries as it cannot preserve the accurate distances in the original space. Second, even though suitable encryptions are found for these queries, they become flawed when applied to our problem, as these encryptions are not designed for mutual privacy protection in the first place. In particular, to evaluate the query on the encrypted data, the client must encrypt the query by the same scheme and send it to the outsourcing server, who may then forward it to the data owner, where the query can be decrypted by her encryption parameters. Third, some encryptions or transformations are shown to be vulnerable to certain security attacks. For example, distance-preserving space transformations are vulnerable to principal component analysis [22], [25], [28].

In spite of the insufficiency of these prior studies for our problem, they show us that a secure framework and an alternate encryption scheme are both indispensable. In this paper, we propose a holistic and efficient solution that is based on Privacy Homomorphism (PH). PHs are encryption transformations which map a set of operations on cleartext to another set of operations on ciphertext. In essence, PH enables complex computations (such as distances) based solely on ciphertext, without decryption. We integrate a provably secure PH seamlessly with a generic index structure to develop a novel query processing framework. It is efficient and can be applied to any multi-level tree index. We address several challenges in this framework. First, an index consists of multiple nodes, and query processing on the index involves traversing these nodes. The cloud or data owner should not be able to trace the access pattern and hence get any clue of the query. We propose a client-lead processing paradigm that eliminates the disclosure of the query to any other party. Second, to evaluate various types of complex queries, such as kNN and other distance-based queries, a comprehensive set of client-cloud protocols must be devised to work together with a PH that supports most arithmetic operations. Third, we prove the security and analyze the complexity of the proposed algorithms and protocols. In particular, we present several optimization techniques to improve the protocol efficiency and also show their privacy implications. To summarize, our contributions in this paper are as follows:

- To the best of our knowledge, this is the first work that is dedicated to mutual privacy protection for complex query processing over large-scale, indexed data in a cloud environment.
- We present a general index traversal framework that accommodates any multi-level index. The framework can resist the index trace attempt of the cloud during query processing. Based on this framework, we present a set of protocols to process typical distance-based queries.
- We thoroughly analyze the security and complexity of the proposed framework and protocols. In particular, we present several optimization techniques to improve the protocol efficiency.
- An extensive set of experiments are conducted to evaluate the actual performance of our basic and optimized techniques.

The rest of the paper is organized as follows. Section II reviews existing work on privacy-preserving query processing on outsourced data. Section III formulates the problem and Section IV introduces ASM-PH, the privacy homomorphism used in this paper. Section V overviews the secure processing framework, followed by detailed discussions on the protocols in Sections VI and VII, with a focus on distance-based queries. Section VIII presents three optimization techniques to improve the protocol efficiency. Section IX analyzes the security and possible threats of our approach, followed by the performance evaluation in Section X. Section XI concludes this paper with some future research directions.

II. Related Work

In this section, we review existing privacy-preserving data outsourcing techniques for query processing purposes. The common model is that an *untrusted* outsourcing server stores and manages the data on behalf of the data owner, who then invites *trusted* users to query the data. The first category of techniques is based on the generalization principle to minimize the disclosure of precise information. For relational data, generalization can protect quasi-identifier attributes and avoid the disclosure of sensitive information [3], [29]. For spatial data and query, a similar technique called location cloaking has been proposed to generalize (i.e., blur) the user or object locations [16], [12], [13], [18]. However, these techniques still disclose the data or query in a coarser and imprecise form.

The second category encrypts or transforms both the query and the data into another space for evaluation, using hashing or space filling curves. Agrawal *et al.* proposed an order-preserving encryption scheme (OPES) for 1D numeric values [1]. SQL statements such as MAX, MIN, COUNT, GROUP BY, and ORDER BY can then be rewritten and processed over the encrypted data. However, OPES does not support SUM or AVG statements, in which the original data must be decrypted first. Yiu *et al.* extended this transformation technique to 2D spatial data points and proposed hierarchical space-division (HSD) [31]. To protect mutual privacy as in this paper, Khoshgozaran and Shahabi used space filling curves as the transformations for nearest neighbor search [20]. However, since distance is not completely preserved in the transformed space, the results are only approximate kNNs. Another disadvantage of transformation techniques is the potential disclosure risks. As they are almost distance-preserving, adversaries may utilize this knowledge and recover the original data by linear algebra or principal component analysis [22], [25]. To resolve this issue, Wong *et al.* recently proposed a new encryption scheme that only preserves the scalar product value of two points, which is sufficient to answer accurate

kNN queries [28].

Our work falls into this category but distinguishes itself from the others as being the first work that is dedicated to mutual privacy protection. We propose a secure, encryption-integrated framework that is suitable for processing complex queries over large-scale, *indexed* data. It is noteworthy that privacy-preserving search on tree-structured data has been studied in some existing studies [6], [27], [2]; however, these works either consider one-way privacy or cannot provide unconditional privacy guarantee.

The third category considers a distributed environment where the data are partitioned and outsourced to a set of independent and non-colluding outsourcing servers. The privacy-preserving query processing requires a distributed and secure protocol to evaluate the result without disclosing the data in each outsourcing server. The security foundation of such protocols originates from secure multiparty computation (SMC), a cryptography problem that computes a secure function from multiple participants in a distributed network. Privacy-preserving nearest neighbor queries have been studied in this context for data mining. Shaneck *et al.* presented a solution [24] for point data on two parties. Qi and Atallah improved this solution by applying a blind-and-permute protocol, together with a secure selection and a multi-step kNN protocol [23]. For approximate aggregation queries, Li *et al.* proposed randomized protocols based on probabilistic computations to minimize the data disclosure [30]. For vertically-partitioned data, privacy-preserving top-k, kNN and join queries are studied in [26], [19], [21]. More recently, Ghinita *et al.* proposed a private-information-retrieval (PIR) framework to evaluate kNN queries in location-based services [14]. Thanks to oblivious transfer, a common primitive in SMC, the user can retrieve the results without being pinpointed. However, solutions in the third category typically suffer from heavy CPU, communication and storage overhead, as most SMC-based protocols do. As such, they cannot scale well to large-scale databases.

III. Problem Formulation

We consider a cloud computing model of three parties: the data owner, the querying client, and the cloud service provider (or simply the cloud). The data owner owns a voluminous dataset $\mathcal{D}$, and outsources its query processing service to the cloud. The dataset contains some proprietary and sensitive attributes θ (e.g., salary, date of birth, social security number) that need to be protected from the cloud and the querying client. On the other hand, the client queries on the same sensitive attributes θ to retrieve the identifiers of qualified objects in $\mathcal{D}$. After the query processing, these identifiers can be used to retrieve non-sensitive contents (e.g., name, sexuality) of these objects. The query q needs to be protected against both the data owner and the cloud. To summarize, the problem in this paper is to process queries on sensitive attributes θ while protecting both data and query values on θ.

There are several remarks about the problem definition. First, without loss of generality, in this paper we assume θ are a set of attributes whose values are indexed by a multi-dimensional hierarchy. For ease of presentation, we assume it is an R-tree as illustrated in Fig. 2. The index has a hierarchy of nodes, each of which is composed of many *entries*. An entry corresponds to a child node, and it consists of a pointer p pointing to the child node together with a minimum bounding box (MBB) e of all data objects that belong to the child node. An entry in a leaf node has its p store the identifier of a data object and its e store the values of sensitive attributes θ. Second, this paper focuses on queries that involve a complex value function between the query q and the attribute values in θ. Typical example of function is the Euclidean distance. In the rest of this paper, we use two typical distance-based queries as illustration, namely, the kNN query and distance range query. The latter returns result objects whose distances from the query are within a specified threshold. Third, while the cloud and data owner should know absolutely nothing about the query, the client can always know about the data values on θ that are implied in the query. As such, our problem is to prevent the client from knowing beyond what the query tells. A malicious client may attempt to narrow down or pinpoint the θ values of returned objects by exhaustively sending distance-range queries with extremely small thresholds. Depending on the business model, this threat can be prevented by access control, query parameters screening, or imposing penalties or charges on heavy users; however, this topic is beyond the scope of this paper.

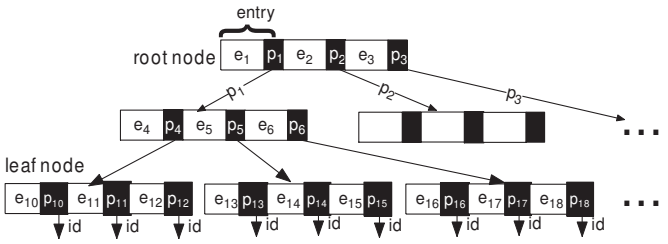


Fig. 2. Illustration of Multi-Level Index

A. Adversary Model

In this paper, we assume the three parties follow a semi-honest model. That is, they follow the protocol properly except that they may record all intermediate results and try everything they can to deduce about the other parties [8], [15]. The cloud and data owner may collude to share their information about the client, as they normally do in practice. However, the client does not collude with the cloud or data owner, as they have no common interests.

The adversary may represent the client, data owner, or the cloud. The client adversary attempts to obtain the values of proprietary attributes θ from the owner during query processing. The data owner adversary attempts to obtain the query value on θ from the client during query processing. The cloud adversary attempts to obtain both the values of proprietary attributes θ from the owner during data outsourcing and the client's query value on θ during query processing. We assume that an adversary knows all the protocols and algorithms except for the secret keys of other parties. However, he/she

does not have a-priori knowledge of the data or query, in the form of sample values or distributions. Furthermore, the capability of an adversary is bounded by both computational power and storage space.

IV. Preliminary

Before we present the secure processing framework, we introduce privacy homomorphisms (PH), the internal encryption scheme. PH are encryption transformations which map a set of operations on cleartext to another set of operations on ciphertext. Formally, they are encryption functions $E_k : T' \longrightarrow T$ that allow a set of operations F on encrypted data without knowledge of the decryption function D_k. The following is a simple illustration.

Let p and q be two large and secret primes. $m = pq$ is public. The cleartext set $T' = Z_m = \{0, 1, ..., m-1\}$ and the set of cleartext operations $F' = \{+_m, -_m, \times_m\}$, which are addition, subtraction and multiplication modulo m. The ciphertext set $T = Z_p \times Z_q$. The set of ciphertext operations F are the same as in F' except that they are componentwise. Define the encryption key $k = (p, q)$ and the encryption $E_k(a) = [a \mod p, a \mod q]$. The decryption is by the Chinese remainder theorem, which says for k positive integers $n_1, ..., n_k$ that are pairwise coprime, there exists a unique solution x in Z_N (N = $n_1 n_2 ... n_k$) that satisfies: $x \equiv a_i \mod n_i, \forall 1 \leq i \leq k$. Furthermore, x can be found in polynomial time by applying the extended Euclidean algorithm.

Obviously, this encryption is privacy homomorphism under the operations defined by F' and F, because $m = pq$. However, this encryption suffers from known-plaintext attacks [4], which means p and q could be found if a pair of cleartext and ciphertext is known to an adversary.

A. A Provably Secure Privacy Homomorphism

In [7], Domingo-Ferrer enhanced the above simple PH and proposed a provably secure privacy homomorphism under the same set of operations, i.e., modular addition, subtraction and multiplication. We name it ASM-PH after its supported operations. It works as follows. The public parameters are a positive integer $t > 2$ and a large integer m. t controls how many components a cleartext is split into ($t = 2$ in the above PH). m should have many small divisors (compared to t). Further, many integers smaller than m should be invertible modulo m. That is, $\exists r \in Z_m$ such that there is r^{-1} and $r \times r^{-1} \equiv 1 \mod m$. The private parameters are such an r and a divisor $m' > 1$ of m. Thus, the secret key is $k = (r, m')$.

The set of cleartext is $T' = Z_{m'}$. The set of ciphertext is a t-tuple, i.e., $T = (Z_m)^t$. The set F' of cleartext operations is formed by addition, subtraction and multiplication in T'. Similar to the simple PH, the set F of ciphertext operations are the corresponding componentwise operations in T. Finally, the encryption and decryption of this PH can be described as follows.

- **Encryption.** Randomly split a cleartext $a \in Z_{m'}$ into secret $a_1, ..., a_t$ such that $a = \sum_{j=1}^{t} a_j \mod m'$, where $a_j \in Z_m$. Then the ciphertext of a is:

$$E_k(a) = (a_1 r \bmod m, a_2 r^2 \bmod m, ..., a_t r^t \bmod m)$$

- **Decryption.** Obtain r^{-1}. Compute the scalar product of the j-th component by $r^{-j} \bmod m$ to get $a_j \bmod m$. Then compute $\sum_{j=1}^{t} a_j \mod m'$ to get cleartext a.

The set F' of ciphertext operations consists of:

- **Addition and Subtraction.** They are done componentwise, i.e., between terms with the same r degree.
- **Multiplication.** All terms are cross-multiplied in Z_m, which means t_1-th degree term by a t_2-th degree term yields a $t_1 + t_2$-th degree term. Terms that have the same degree are added up.

While ASM-PH can perform addition, subtraction and multiplication directly on the ciphertexts, these operations still cost considerable computations. Let η_+ denote the cost of a modular sum and $\eta_\times$ denote the cost of a modular multiplication.[1] Then the costs of these three operations are $t\eta_+$, $t\eta_+$, and $t^2\eta_\times + (t^2 - t)\eta_+$, respectively. It is also noteworthy that the multiplication will double the size of the ciphertext from t components to $2t$ components, with the first component being zero.

As for the computation cost of encryption, it is $t\eta_\times$ as each component requires a modular multiplication with r^j.[2] The cost of decryption is similar, except that all components are summed up in the end. As such, the cost is $t\eta_\times + (t-1)\eta_+$. It is noteworthy that the encryption will increases the size of the cleartext from 1 component to t components. Since each component is a positive integer in Z_m, the size of the ciphertext is thus $t \cdot l(m)$, where $l(m)$ denotes the number of bits in m.

In practice, the cost of modular addition is dominated by that of modular multiplication [9] and thus can be omitted when the latter presents. Modular multiplication, especially for large modulus, also becomes extremely efficient (in the magnitude of 10^{-5} second) since the introduction of Montgomery Reduction [11].

ASM-PH is shown to be secure against known-plaintext attacks. Analytically, the size of the subset of keys that are consistent with n known cleartext-ciphertext pairs grows exponentially with $s - n$, where $s = \log_{m'} m$. This means the genuine key could be from an arbitrarily large key set. More security aspect of ASM-PH is analyzed in Section IX.

V. Overview of Secure Query Processing on R-Tree Index

In essence, processing distance-based queries over a multi-dimensional index can be regarded as a traversal on the tree nodes. More specifically, it can be separated into two alternate procedures: node traversal and distance access. The distance access determines the next node to traverse based on the distances computed from the current node and query point. To preserve query and data privacy, both procedures must remain secure in the outsoucing model of three parties. That is, during

[1] In a more accurate sense, the computational cost of any modular operation depends on the modulus. However, we omit this discrepancy as a unique modulus m or m' is chosen for encryption or decryption throughout the ASM-PH.

[2] Since r is part of the encryption key and thus known in advance, all $r^j \bmod m$ can be precomputed.

query processing neither the data owner nor the cloud can identify the traversed nodes or obtain any information that can pinpoint the query point (such as the exact distances to the query point). Meanwhile, the client should have no access to the actual node contents during distance access and node traversal.

In this section, we present the overview of a secure query processing framework that can achieve these requirements. Figure 3 shows the framework overview. The key idea is to let the client lead in the distance access and keep track of the traversal path, i.e., the query processing state, so that neither the data owner nor the cloud knows the exact node the client is accessing, let alone the query point. On the other hand, to protect data privacy, the client has only access to an encrypted version of the index, and must proceed the query processing together with the cloud who can decrypt the distances it computes locally. As such, the distance access is a joint process of the client and data cloud, in which neither party has access to the actual distances.

The detailed process flow of this framework is as follows. During initialization (step 0), the data owner sends to the client a *shadow index* — an encrypted version of the index I (denoted by $E(I)$). Specifically, in each index node i, the key range of each entry (e.g., e_1, e_2, e_3) is encrypted by encryption scheme $E(\cdot)$, while the pointers (e.g., p_1, p_2, p_3) are not encrypted. In other words, the shadow index has the same topology as the original index, but all the key values are encrypted. The shadow index is then stored at the client side for future access. Meanwhile, the data owner sends the decryption scheme $E^{-1}(\cdot)$ to the data cloud for future distance decryption. It is worth noting that the data owner only involves in this initialization step and she can further reduce its involvement by delegating the shipping task of shadow index to the cloud. In this case, the shadow index kept at the cloud must be further encrypted by the owner's privacy key through any public-key cryptography. Then during initialization, she only needs to send her public key to the client, who then retrieves and decrypts the shadow index from the cloud.

Next, during the traversal, each time the client is to access an index node i, it retrieves the shadow node $E(i)$ locally and computes the local distances between the entries in $E(i)$ and $E(q)$ (step ①). These distances are then sent to the data cloud, who decrypts and recodes them for the client (step ②) for the client. The recoding ensures the client only receives an encrypted version of the actual distances that are only sufficient for the query processing. On the other hand, to prevent the cloud from accessing the actual distances after decryption, the client needs to scramble the local distances before sending them to the cloud. After the distance access, if the node is a leaf node, the client will update the query result $\mathcal{C}$. The client then finds the next node to traverse (step ③), which will go through the same steps ①②③. It is noteworthy that step ③ is the only step that depends on the query type, based on which the decision will be made to find the next node to traverse. Alg. 1 shows the complete pseudo-code of this secure query processing on a multi-level index. The traversal starts from the root node, and the node access process repeats until the query is completed.

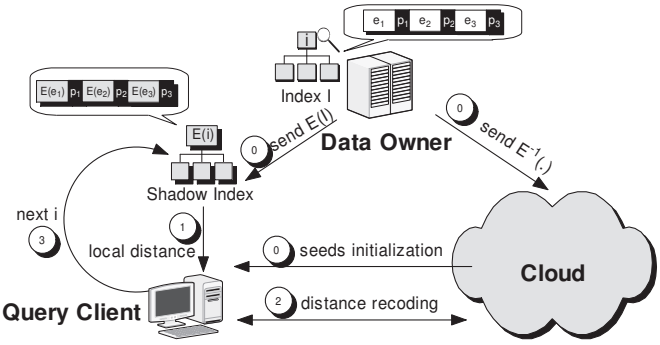


Fig. 3. Privacy-Preserving Query Processing Framework

Algorithm 1 Privacy-Preserving Processing Framework for Distance-based Queries

Input: q: the query at the client
I: the index at the data owner
$E(\cdot)$: the encryption known to the data owner and cloud
Output: $\mathcal{C}$: the query result
Procedure:
1: data owner sends shadow index $E(I)$ to the client;
2: data owner sends the decryption $E^{-1}(\cdot)$ to the cloud;
3: client initializes a set of seeds $\mathcal{S}$ with cloud;
4: client initializes the root of $E(I)$ as i, the next node to access;
5: **while** q is not completed **do**
6: client retrieves shadow index node $E(i)$, computes and scrambles the local distances from $E(q)$; // step ①
7: server receives the scrambled local distances, decrypts and recodes them; // step ②
8: client updates i and $\mathcal{C}$ according to the recoded distances; // step ③

A. Encryption Schemes

The framework above requires the actual distances between the original node i and any query point q can be restored from local distances — the distances between the encrypted version of the node $E(i)$ and $E(q)$. This calls for the encryption scheme $E(\cdot)$ a homomorphic one so that the distances in the encrypted space equal to the encrypted distances in the original space. That is, $dist(E(i), E(q)) = E(dist(i, q))$. Without loss of generality, we assume in this paper that the distance metric is Euclidean distance: given two d-dimensional points $\overrightarrow{x}$ and $\overrightarrow{y}$, the distance is the sum of square distance in each dimension, i.e., $dist^2(\overrightarrow{x}, \overrightarrow{y}) = \sum_{i=1}^{d} (x_i - y_i)^2$. Since the Euclidean distance only involves addition, subtraction and multiplication, an arithmetic homomorphic encryption on these three operations, like the ASM-PH introduced in Section IV, is sufficient for $E(\cdot)$.

However, several issues must be solved before ASM-PH can be applied in this framework. The first is the domain of the cleartext. In ASM-PH, the set of cleartext is $Z_{m'} = \{0, 1, ..., m' - 1\}$. As such, the original space must be discretized into this integer domain and this process could lose precision. Second, the operations defined in ASM-PH are modular operations while the distance operation is not. To ensure the arithmetic addition, subtraction and multiplication do not overflow in ASM-PH, we need to restrict its cleartext domain and revise it to allow negative values. The new domain

is denoted as $\mathcal{Z}_{m'}$ and is formally defined as below:

1) $\mathcal{Z}_{m'} = \{-\lceil m'/2 \rceil + 1, ..., 0, ..., \lfloor m'/2 \rfloor\}$.
2) m' must be sufficiently large, in particular, $m' > \sum_{i=1}^{d} (\max x_i - \min x_i)^2$, where $\max x_i$ and $\min x_i$ are the maximum and minimum values in the i-th dimension.

The last issue is the encryption of the query point q. It appears to be a hard problem if $E(\cdot)$ is a general encryption scheme: the client owns the query point but has no access to the encryption key; on the other hand, the data cloud has the encryption key but has no access to the query point. Fortunately, ASM-PH enables us a convenient solution to this issue. By integer decomposition algorithms (a classic one based on the extended Euclidean algorithm was proposed by Gallant *et al.* [10]), the query q can be decomposed into an integer form $q = q_1 + q_2\lambda \bmod m$. By choosing a set of cleartexts including q_1, q_2 and λ as "seeds" and sending them to the cloud for encryption, the client can compute the encrypted value of q by $E(q) = E(q_1) + E(q_2)E(\lambda) \bmod m$. In addition to obtaining $E(q)$, as will be discussed in Section VI-C, these seeds will also be used for the scrambling process.

B. Overhead and Challenges

The framework imposes computational and communication overhead on top of the conventional query processing. As for the computational overhead, in each node traversal, there is a local distance computation on the client side followed by a decryption and recoding on the server side. As for the communication overhead, in each node traversal, both sides send and receive a set of distances for the node entries.

There are several challenges regarding security and efficiency in this framework that will be addressed in the following sections:

1) The core of this framework is distance access. It comprises local distance computation, decryption and recoding, and client scrambling, all of which will be presented in Section VI.
2) Since each node traversal and distance access incur both computational and communication overhead, optimization techniques will be designed in Section VIII to prune unnecessary distance computation and node traversal.
3) Regarding the security, we will prove in Section IX that this framework preserves both data privacy and query privacy, based on the security of ASM-PH. Nonetheless, we admit certain amount of privacy loss in this framework, such as the disclosure of index topology to the client.

VI. Distance Access Over R-tree Index

In this section, we study the problem of distance access on R-tree indexed data. Specifically, given the query point q and an encrypted index node $(E(e_1), E(e_2), ..., E(e_n))$ at the client, and the decryption $E^{-1}(\cdot)$ at the data cloud, we study how to find for the client the index entry (or entries) p to traverse next for query q without the cloud knowing either q or p. The challenge arises from several aspects. First, the client only holds an encrypted version of the node, whose decryption is only known to the cloud. This creates barriers for both parties to compute the distances between q and node entries. Second, to decide the next node to traverse, the client should have access to the distances computed. However, the disclosure of such distances to either party may compromise the data or query privacy. As such, a minimum disclose of the distance information should be designed that is only sufficient for the client to proceed with the index traversal. This process is through distance recoding at the cloud. At the client side, the local distances are scrambled to prevent the cloud from accessing the actual distances after decryption.

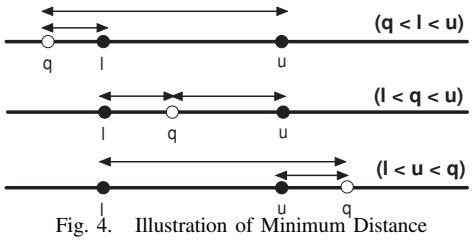


Fig. 4. Illustration of Minimum Distance

In the rest of this section, we start with local distance computation, and then present the distance recoding scheme and scrambling process, and finally summarize the overall distance access protocol.

A. Local Distance Computation of Minimum Square Distance

According to Euclidean distance, given two d-dimensional points $\overrightarrow{x}$ and $\overrightarrow{y}$, the square distance $dist^2(\overrightarrow{x}, \overrightarrow{y})$ is $\sum_{i=1}^{d} (x_i - y_i)^2$. Distance-based queries on indexes usually involve computing the minimum distance between the query point $\overrightarrow{q}$ and an index entry $[\overrightarrow{l}, \overrightarrow{u}]$, where $\overrightarrow{l}$ are the lower bounds and $\overrightarrow{u}$ are the upper bounds in all dimensions. In each dimension, the minimum distance between value q and range $[l, u]$ is defined in Eqn. 1 (see Fig. 4). More specifically, if $q > u$, the distance is $u - q$; if $q < l$, the distance is $l - q$; otherwise the distance is 0.

$$2 \cdot min\_dist(q, [l, u]) = |u - q| + |l - q| - (u - l) = sign(u - q) * (u - q) + sign(l - q) * (l - q) - (u - l) \quad (1)$$

Here $sign$ denotes the sign function. By applying the encryption scheme $E(\cdot)$ on both sides of the equation and using the property of ASM-PH, we have

$$E(2 \cdot min\_dist(q, [l, u])) = sign(u - q) * (E(u) - E(q)) + sign(l - q) * (E(l) - E(q)) - (E(u) - E(l)) \quad (2)$$

It is noteworthy that the equation above treats $sign(u - q)$ and $sign(l - q)$ as constants of -1 or 1. While all other terms $E(q), E(u), E(l)$ are available at the client, these two constants need to be determined by the data cloud. As such, we introduce a two-phase distance computation method in which the first phase determines the signs of $u - x$ and $l - x$ and the second phase computes the minimum distance according to Eqn. 1. Specifically, in the first phase, the client computes $E(u) - E(q) = E(u - q)$ and $E(l) - E(q) = E(l - q)$ and sends both to the cloud for decryption. But in order to hide $u - q$ and $l - q$ from both sides, the client need to scramble these values before sending them to the cloud. The general scrambling process will be discussed in Section VI-C. The cloud, on the other hand, only sends back the sign results. In the second phase, after restoring $sign(u - q)$ and $sign(l - q)$,

the client computes the $E(2{\cdot}min\_dist)$ as in Eqn. 2. The local distance between $E(\overrightarrow{q})$ and an entry $E([\overrightarrow{l}, \overrightarrow{u}])$, denoted by $local\_dist(E(q), E([\overrightarrow{l}, \overrightarrow{u}]))$ is simply the sum of the square of the minimum distance over all dimensions (see Eqn. 3). Note that to align with Eqn. 2, we leave the constant factor 4 inside the definition.

$$local\_dist(E(q), E([\overrightarrow{l}, \overrightarrow{u}])) = E(4{\cdot}min\_dist^2(\overrightarrow{q}, [\overrightarrow{l}, \overrightarrow{u}])) = \sum_{i=1}^{d} E^2(2 \cdot min\_dist(q_i, [l_i, u_i])) \quad (3)$$

It is noteworthy that rather than iterating the two phases for each dimension, all dimensions can proceed simultaneously in each phase, as dimensions are independent of each other. For each dimension, the client computational costs in each phase are $2dt\eta_+$ and $5dt\eta_+$, respectively. These costs are dominated by the self-multiplication in Eqn. 3. So the total client computation is approximately $dt^2\eta_\times$ plus the scrambling cost. On the other hand, the cloud computational cost in each phase is $4dt\eta_\times$ and $2t\eta_\times$, respectively.[3] So the cloud computation is approximately$(4d + 2)t\eta_\times$ plus the recoding cost. As for the bandwidth, the client sends $4dt$ and $2t$ components in both phases, so the client bandwidth cost is $(4d + 2)tl(m)$.

B. Distance Recoding Scheme

The local distances computed above are encrypted by $E(\cdot)$. They must be sent to the cloud for decryption. This process is similar to the first phase of local distance computation: the client scrambles the encrypted distances and the cloud decrypts them. However, rather than sending the $sign$ results directly, the cloud must encrypt the distances to prevent the client from accessing the actual distances. This process is distance recoding and it sends back a recoded version of the distances that are only sufficient for distance comparison. To this end, the recoding scheme should at least satisfy the following two properties.

1) **Strictly monotonic**: This property guarantees the recoded distances can still be compared with each other.
2) **Immune to chosen ciphertext attack**: This property is required because the client can issue any chosen plaintext distance and obtain its recoded value.

In the following, we propose a recoding scheme that satisfies both properties. The key idea is to record at the cloud side all existing recoded value pairs to guarantee strict monotonicity. At any time, two recoded values are always farther away than their original values so that it is always possible to find for a new value its recoded value while still maintaining monotonicity. On the other hand, this recoded value is random to guard against chosen-ciphertext attacks. Alg. 2 shows the pseudo-code of the procedure. It first locates the input value x in the set of recoded pairs as before value s and after t. Then it finds a proper and random recoded value x' based on s' and t'. It is noteworthy that to guarantee the future recoding between s and x still affords to be random, $x' - s'$ is set to be at least $\lfloor (t' - s')/3 \rfloor$ larger than $x - s$, and so is $t' - x'$. As such, the random values generated by $rand()$ must be large enough to ensure $\lfloor (t' - s')/3 \rfloor > 0$ throughout the lifetime of a query, i.e., the duration for the monotonicity.

[3]After the scrambling process, the scrambled distance received by the server has $2t$ components.

Algorithm 2 $recode$: Distance Recoding Scheme

Input: x: the decrypted distance at the cloud
$\mathcal{X}$: the recoded value pairs
$rand()$: a random function
Output: x': the recoded value of x
Procedure:
1: locate x in $\mathcal{X}$ such that s, t are adjacent in $\mathcal{X}$ and $s < x < t$;
2: **if** t does not exist (i.e., x is the largest) **then**
3: t is the largest value in $\mathcal{X}$;
4: $x' = t' + (x - t) + rand()$;
5: **else if** s does not exist (i.e., x is the smallest) **then**
6: s is the smallest value in $\mathcal{X}$;
7: $x' = s' - (s - x) - rand()$;
8: **else**
9: choose x' from range $[s' + 1, t' - 1]$ such that $x' - s' > x - s + \lfloor (t' - s')/3 \rfloor$ and $t' - x' > t - x + \lfloor (t' - s')/3 \rfloor$;
10: insert (x, x') into $\mathcal{X}$;

C. Scrambling

In general, the scrambling process is invoked by the client on the set of encrypted values $E(\xi)$ (e.g., the local distances) before they are sent to the cloud for operation $\mathcal{F}$ (e.g., decryption and recoding). It prevents the cloud from obtaining the actual values of ξ by modifying values in $E(\xi)$ into $E'(\xi)$. As such, the actual operation result $\mathcal{F}(E(\xi))$ needs to be restored (or called "descrambled") based on the result $\mathcal{F}(E'(\xi))$ that is returned from the cloud. There are two levels of scrambling: permutation $\Pi(\cdot)$ and deviation $dev(\cdot)$. The former changes the order of a set of encrypted values, which are otherwise in the same order of the entries. The latter changes every value $E(\xi)$ by arithmetic operations with the seeds. To enable the restoration of the operation result of $\mathcal{F}$, however, the deviation should depend on $\mathcal{F}$. In this framework, $\mathcal{F}$ can be either of the following two:

1) $\mathcal{F} = sign(E^{-1}(\cdot))$: The deviation multiplies $E(\xi)$ with a seed $E(s)$ and sends $E(s) \cdot E(\xi) = E(s \cdot \xi)$ to the cloud. Upon receiving $sign(s \cdot \xi)$, the client can restore $sign(\xi)$ by checking the sign of s.

2) $\mathcal{F} = recode(E^{-1}(\cdot))$: The deviation generates two values s_1 and s_2 from the set of seeds by any combination of multiplication, addition and subtraction. Then it sends $E(s_1){\cdot}E(\xi){+}E(s_2) = E(s_1{\cdot}\xi{+}s_2)$ to the cloud. To ensure the scrambled distance is consistent over the dimensions, entries and nodes, s_1 and s_2 must be kept the same throughout the lifetime of a query. Upon receiving $recode(s_1 \cdot \xi + s_2)$, the client can compare it with other recoded distances based on the sign of s_1. This deviation works as both itself and the recoding scheme are monotonic.

Note that to strengthen the anonymity of seeds against the cloud, composite seeds are used in the deviation process. They are generated by arithmetic operations (addition, subtraction, and even multiplication) on a set of initial seeds. In addition, while deviation operations effectively protect the query, they

incur a multiplication, which costs the client $t^2\eta_\times$ computation for a t-component ciphertext.

D. Distance Access Protocol

Now we describe the node access protocol for distance-based queries. This access involves the client with query $\overrightarrow{q} = (q_1, q_2, ..., q_d)$ and the cloud with entries $\overrightarrow{e} = (e_1, e_2, ..., e_n) = ([\overrightarrow{l_1}, \overrightarrow{u_1}], ..., [\overrightarrow{l_n}, \overrightarrow{u_n}])$. Alg. 3 summarizes the procedures in pseudo-code. As for the computational cost, the client involves in n local distance computations (which costs $dnt^2\eta_\times$) and scrambles of $2dn$ t-component and n $2t$-component ciphertexts (which costs $(2d+4)nt^2\eta_\times$). So the total computation cost is $(3d+4)nt^2\eta_\times$. The cloud involves in n local distance computations, so the cost is $(4d+2)nt\eta_\times$. The communication cost of this protocol is $(4d+2)ntl(m)$.

Algorithm 3 $dist\_access$: Node Access Protocol for Distance

Input: $\overrightarrow{q}$: the query value at client
$\Pi(\cdot)$: the permutation by the client
$dev(\cdot)$: the deviation by the client
$\overrightarrow{e}$: the entries at cloud
$recode$: the distance recoding scheme by cloud
Output: $\mathcal{D}(\overrightarrow{q}, e_i))$: the recoded minimum square distance between $\overrightarrow{q}$ and e_i;

Procedure:
1: for each entry e_i, client sends $E(\overrightarrow{u_i}) - E(\overrightarrow{q})$ and $E(\overrightarrow{l_i}) - E(\overrightarrow{q})$ to cloud, scrambled by $\Pi(\cdot)$ and $dev(\cdot)$; // Phase 1
2: cloud decrypts them and sends back their sign values;
3: client restores them and computes $local\_dist(E(\overrightarrow{q}), E([\overrightarrow{l_i}, \overrightarrow{u_i}]))$ according to Eqns. 2 and 3;
4: client scrambles them by $\Pi(\cdot)$ and $dev(\cdot)$, sends to cloud; // Phase 2
5: cloud decrypts and recodes them;
6: client restores the distances as $\mathcal{D}(\overrightarrow{q}, e_i))$;

VII. Processing Distance-based Spatial Queries

In this section, we present the complete query processing protocols for distance range and k-nearest neighbor queries, using the above node access protocol for distance.

A. Processing Distance Range Queries

In a distance range query, the client searches for records whose distances are within r away from the query point $\overrightarrow{q}$. Alg. 4 shows the pseudo-code of the distance range search protocol. The search at the client starts from the root entry of the shadow index and traverses every node whose minimum square distance to $\overrightarrow{q}$ is less than or equal to $4r^2$ ("4" aligns with Eqn.3). To enable the comparison, $4r^2$ is sent to the server for recoding when query starts.

B. Processing k-Nearest Neighbor Queries

We base the kNN search protocol (shown in Alg. 4) on the *best-first* search (BFS) [17], a state-of-the-art kNN search algorithm on R-trees. BFS uses a priority queue Q to store entries to be explored during the search. The entries are sorted by their minimum square distances. BFS pops up the top entry in the queue, pushes its child entries into the queue and then repeats the process. When a leaf entry is popped, the corresponding record is retrieved as the nearest neighbor. This protocol proceeds to search k nearest neighbors and terminates when all kNNs are retrieved.

Algorithm 4 Complete Query Processing Protocol

Input: $\overrightarrow{q}$: the query point at client
r: the threshold for distance range query at client
k: for kNN query at client
$root$: the root entry of the shadow index at client
Output: $\mathcal{C}$: the set of result objects
Procedure:
1: client initializes queue Q and $\mathcal{C} = \emptyset$;
2: client enqueues $root$ into Q;
3: **while** Q is not empty **do**
4: client dequeues entry p from Q;
5: **if** p is a leaf entry **then**
6: $\mathcal{C} = \mathcal{C} \cup p$;
7: **kNN query** : **if** $|\mathcal{C}| == k$, **return** $\mathcal{C}$;
8: **else**
9: client retrieves shadow index node $\overrightarrow{e} = (e_1, e_2, ...)$ pointed by p;
10: client gets $\mathcal{D}(\overrightarrow{q}, \overrightarrow{e}) = dist\_access(\overrightarrow{q}, \overrightarrow{e})$;
11: client enqueues $(e_i, \mathcal{D}(\overrightarrow{q}, e_i))$ of qualified e_i into Q as below;
12: **Range query** : those e_i whose $\mathcal{D}(\overrightarrow{q}, e_i)) \leq recode(4r^2)$;
13: **kNN query** : all e_i;

VIII. Performance Optimization

In previous sections, we introduce the basic framework and protocols for privacy-preserving distance-based queries. In this section, we further propose several optimization techniques on the basic protocols to further improve the performance. These optimizations are orthogonal to each other and are introduced in the ascending order of their optimization scope.

A. One-Off Local Distance Computation

The basic local distance computation requires a two-phase protocol, in which the first phase is to identify the position of the query point with respect to the range to facilitate the minimum square distance calculation in the second phase. However, this protocol has two performance issues. First, it requires two rounds of client-cloud interaction, which may increase the query response time. Second, all costly multiplications (including the scrambling) are performed at the client side, while the cloud only involves in decryptions. To remedy these drawbacks, we propose a one-off approach by sending for each dimension only the scrambled $E(u) - E(q)$ and $E(l) - E(q)$ to the cloud. As shown in Fig. 4 and Eqn. 2, the minimum distance takes three different values according to the comparison result of $u-q$ and $l-q$, and furthermore, $u-q$ and $l-q$ are all the cloud needs to compute the minimum distance. Since the cloud eventually knows the comparison result, the scrambling $\mathcal{F}$ of $E(u-q)$ and $E(l-q)$ can be simplified as follows. The deviation multiplies $E(\xi)$ with a seed $E(s)$ and sends $E(s) \cdot E(\xi) = E(s \cdot \xi)$ to the cloud. The same s will be used for both $E(u-q)$ and $E(l-q)$ in all dimensions, entries, and nodes throughout a query's lifetime.

Once the cloud receives and decrypts $E(s(u-q))$ and $E(s(l-q))$, if they have different signs, the local distance is 0; otherwise, the cloud uses the one with a smaller absolute value to compute the minimum distance. Note that this minimum distance is computed in the cleartext domain, and thus no modular operations are involved. As for the computational cost, for each d-dimensional entry, the client's cost is dominated by the scrambling cost of $2d$ t-component ciphertexts, which is $2dt^2\eta_\times$. Compared to $(3d+4)dt^2\eta_\times$ for the two-phase approach, this saves $\frac{d+4}{3d+4}$ computation. For 2D data, this

tops up to a 60% saving. On the other hand, the cloud's cost is dominated by the decryption cost of the $2d$ $2t$-component ciphertexts, which is $4dt\eta_\times$. This saves $2t\eta_\times$ decryption cost in the second phase. The communication cost of the second phase ($2tl(m)$) is also saved, so the one-off approach only needs $4dntl(m)$.

It is noteworthy that this one-off method discloses more information to the cloud and restricts the scrambling process. We will study its privacy implication in Section IX.

B. Distance Folding

Both this and the next optimizations aim to reduce unnecessary distance computations during the distance access for a single node. The key observation for the distance folding optimization is from Eqn. 3, where the local distance is added up from the encrypted minimum square distance in each dimension. Since distance is always positive, a partial local distance from a subset of all dimensions becomes a natural lower bound of the actual local distance. This lower bound is particularly useful because an R-tree node usually has tens or hundreds of entries and some entries could be faraway from the query point, the complete local distances of these entries are not necessary and can be replaced by a lower bound which serves the same query processing purpose. We call this process "distance folding". It is noteworthy that a folded distance can always be unfolded into a larger lower bound or even into the actual local distance if necessary later on.

The main challenge lies in when to stop adding up for the partial location distance — an immature stop leads to an aggressive lower bound that will probably be unfolded later on. For distance range queries, the adding up can be stopped when the lower bound reaches the distance threshold. For kNN queries, however, the decision to fold a distance can relate to the processing status. Specifically, we keep only the topmost L items in the priority queue as "unfolded" while the distances of all rest items are folded as they are. Before a folded distance is to be inserted into the queue, it must be unfolded by at least one dimension or until it is no longer among the topmost L items. This strategy is called is "L-unfolded", where L dictates how aggressive the strategy is. Obviously, $L \geq k$ for kNN queries; moreover, for the best adaption to a specific dataset, L can be runtime-adjustable by its performance as follows. When a query is complete, the saving can be calculated by counting all entries in the queue whose distances are still folded, whereas the overhead can be calculated by counting the number of unfolding operations during processing. L should be increased when the overhead dominates the saving, and vice versa.

C. Entry Folding

While distance can be folded by ignoring some dimensions, the same rationale can be applied to the entries in an index node. The key observation is that, a node i usually has a large number (typically over 100) of entries to fit into one disk page, and it is unnecessary to compute the local distance from q to each entry. As such, some remote entries can be "folded" and represented by a super entry. As there are fewer entries in i, the computational cost of distance access for i can be significantly reduced.

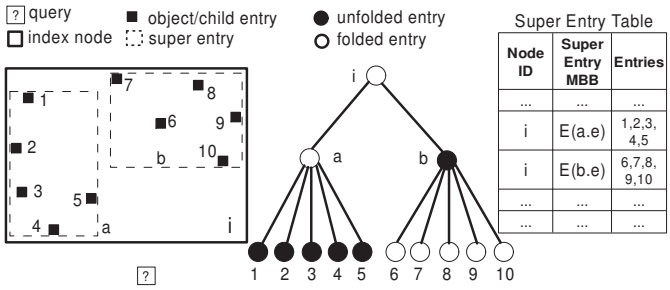


Fig. 5. Super Entry and Entry Folding

Fig. 5 illustrates the notion of super entry and entry folding. The node i contains 10 objects or child entries which form two super entries a and b. When i is accessed, it will be treated as if there are only entries a and b. Later when entry a is accessed (as it is closer to q than b), the same node i will be used and a will be unfolded into entries 1-5. b, on the other hand, remains folded and entries 6-10 can be waived from the distance access.

The entry folding process is conducted offline at the data owner after the index is built. For each node, the set of entries are recursively partitioned into two subsets by dimensional axes like the kd-tree index until each subset contains only one entry. Upon service initialization, besides the shadow index, an auxiliary table is sent to the client that stores the super entry information of all nodes (see Fig. 5). Each table record corresponds to a super entry and has three fields: the node affiliation, minimum bounding box (encrypted by E), and the associated entries. The table size depends on how we regulate the size of a super entry. Based on its query demands, the client should initialize a desirable threshold W for the minimum number of entries in each super entry. For example, W should be set higher than the largest k for kNN queries. In Fig. 5, $W = 5$. Note that entry folding is not equivalent to reducing the fanout of the index, as the latter is query independent. The tradeoff of entry folding lies in the wasted distance computation of super entries that are unfolded later and the saved distance computation of folded entries.

IX. Security Analysis

In this section, we analyze the security aspects of our solution, from both the client and cloud/owner perspectives. We first show the data security of the proposed framework, based on the theoretical results from ASM-PH [7]. We then study the query security, in particular the security of scrambling process and the one-off optimization for distance computation.

A. Data Security

The security of the data is based on two factors — the security of the secret keys in the ASM-PH and distance recoding scheme.

1) Key Security: In [7], Domingo-Ferrer showed the ASM-PH is secure against the compromise of a fixed number h of known cleartext-ciphertext pairs. Specifically, he first showed that the size of the subset of keys consistent with the known

pairs grows exponentially with $s-h$ where $s=\log_{m'} m$. In fact, the expected size of this subset is $max\{6(m')^{s-h}/\pi^2, 1\}$. Then he showed that the probability of two keys (r_1, m_1') and (r_2, m_2') yield the same cleartext from the same ciphertext is $O((\log m)/m)$, which is extremely low. As such, the best attacking strategy of ASM-PH is randomly guessing a key and verifying its consistency with the known pairs, until a key is found that is consistent to all known cleartext-ciphertext pairs. Even if the adversary were not bounded by the computing power, the success rate of such attack is at most equal to $\pi^2(m')^{h-s}/6$. This probability can be extremely small as long as $h < s$, because m' is always large in a practical application domain. However, if $h \geq s$, i.e., the number of known pairs exceeds $\log_{m'} m$, the key that is consistent with the h pairs will be almost unique. Nonetheless, the acquisition of this key is still bounded by the adversary's computational power. Since the key is a pair of (r, m'), the enumeration of all possible pairs in a practical application domain is almost infeasible, let alone the verification process, which requires modular multiplication.

In our framework, the adversary can easily obtain all ciphertexts because the shadow index is stored at the client side. However, he/she cannot obtain any cleartext through our framework, except for the cleartext-ciphertext pairs in the seeds. As the number of seeds is usually small, $h < s$ can be trivially satisfied. As such, we come to the following proposition on key security.

Proposition 9.1: The secret key (r, m') is secure against the client if $h < s$, where h is the number of known cleartext-ciphertext pairs, including the seeds, and $s = \log_{m'} m$.

2) Distance Recoding: The distance recoding on the cloud modifies a scrambled local distance x into x'. To show x' cannot be reverted to x, we show the distance recoding scheme guarantees the expected value of x' is independent of x.

Property 9.2: As the u-th distance to be recoded, the expected value x' for x is independent of x.

Proof: Let $\overline{x'}(u)$ denote the expected recoded value, $\overline{\Phi}'(u)$ and $\overline{\phi}'(u)$ denote the expected values of the maximum and minimum recoded distances among u distances, $\overline{\Phi}(u)$ and $\overline{\phi}(u)$ denote the corresponding distances before recoding, and ζ denote the expected value of the $rand()$ function in the recoding scheme. Then we have recursive formulae as follows.

$$\overline{x'}(u+1) = \overline{x'}(u) * \frac{u-1}{u+1} + (\overline{\Phi'}(u) + x - \overline{\Phi}(u) + \zeta) * \frac{1}{u+1} + (\overline{\phi'}(u) - x + \overline{\phi}(u) - \zeta) * \frac{1}{u+1} \quad (4)$$

$$\overline{\Phi'}(u+1) = \overline{\Phi'}(u) + \zeta * \frac{1}{u+1} \quad (5)$$

$$\overline{\phi'}(u+1) = \overline{\phi'}(u) - \zeta * \frac{1}{u+1} \quad (6)$$

Since x is canceled out in Eqn. 4, and $\overline{\Phi}'(u)$, $\overline{\Phi}(u)$, $\overline{\phi}'(u)$, $\overline{\phi}(u)$ are independent of x, $\overline{x'}(u)$ is independent of x. ■

B. Query Privacy

The security of the query is based on two factors: the security of the scrambling and "untraceable root access". The latter means that the cloud should not be able to pinpoint or narrow down the query during the first node access, when the cloud knows it is always the root node.

1) Scrambling Security: The deviation is based on initial seeds and composite seeds that are derived from initial seeds through arithmetic operations. In what follows, we show that the set of composite seeds can be extremely large by only a few steps of arithmetic operations.

Proposition 9.3: Given g initial seeds, whose ciphertexts all have t components, then the number of composite seeds by h operations and whose ciphertexts have at most wt components is at least $g^h(\sum_{i=0}^{w} 2^{h-i} \cdot \binom{h}{i})$, where $w < h < g$.

Although the above number may include duplicates, deviation based on these composite seeds are still sufficient to prevent the disclosure of original distances individually. In addition, the permutation further disconnects the one-to-one correspondence between the distances and entries.

2) Untraceable Root Access: Since the scrambling process turns genuine distances into relative distances and destroys their correspondence to entries, the cloud is unable to narrow down the query point during the root access, using techniques such as Voronoi Diagram. However, for the one-off approach in Section VIII, the cloud is able to know the relative position of q_i and segment $[l_i, u_i]$, for any dimension i. The following privacy threat exploits this knowledge.

Security Threat 9.4: For the root node and the one-off approach, the cloud or data owner may try to narrow down the query point in any dimension i. Specifically, l_i and u_i of all n entries are projected on a 1D axis, which splits this axis into at most $2n+1$ segments. For any segment seg, if the query point q_i is in it, the number of occurrences when $q_i > u_i$, $l_i \leq q_i \leq u_i$, and $q_i < l_i$ can be determined. If any of these three values are different from what the one-off approach tells, q_i cannot be in this seg and can be filtered.

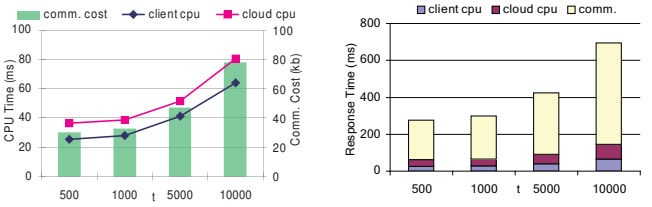
This threat can be alleviated by sending scrambled $E(u-q)$ and $E(l-q)$ values in random dimension order. As such, the above attack cannot be launched for a particular dimension. Rather, all dimensions must be considered as a whole, so the entire space needs to be split into subspaces for filtering. This not only costs more computations for this attack, but also results in more possible subspaces for the query point.

X. Performance Analysis

In this section, we analyze the performance of our proposed framework. While no existing work can be directly compared with it, the purpose is to show the feasibility and study its computation and communication costs under various query types and parameter settings. We build an R-tree index on a real dataset of 123,593 postal addresses in New York, Philadelphia and Boston. The longitude and latitude coordinates of these objects are scaled up to integers and all significant figures are preserved. The resulted cleartext domain is $[0, 10^6]$, which means the m' of ASM-PH must exceed 4×10^{12}. The resulted R-tree has a size of 7.28 MB and the shadow index has a size of 19.08 MB. As the transmission of the shadow index is a one-time operation, such cost is small and can be amortized

TABLE I
PARAMETER SETTINGS FOR EXPERIMENTS

Parameter	Symbol	Value
# of R-tree records	N	$123,593$
page size	–	4KB
encryption key	(m',r)	$[2\text{x}10^{15}, 4\text{x}10^{15}]$, $[10^{15}, 2\text{x}10^{15}]$
threshold for range query	τ	$500 - 10,000$
k of kNN query	k	$1 - 50$



(a) CPU Time and Comm. Cost (b) Query Response Time

Fig. 6. Distance Range Query Performance

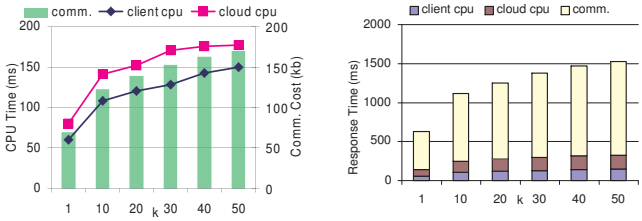
over a series of queries. Further, this size of index can easily fit into the main memory of most modern computers.

The client is set up on a desktop computer with Intel Core Duo T2500 processor and 2GB RAM, running Windows XP SP3 64-bit edition, and the cloud is set up on an IBM eserver xSeries 335, with Dual 4-cores Intel Xeon X5570 2.93GHz CPU and 32GB RAM, running GNU/Linux. The code of our experiments is implemented and executed in Java JDK 1.6.0. As for the ASM-PH encryption scheme, we use $m \in [2 \times 10^{17}, 8 \times 10^{17}]$, $m' \in [2 \times 10^{15}, 4 \times 10^{15}]$, $r \in [10^{15}, 2 \times 10^{15}]$, $t = 3$, and the encryption key is (m', r). The number of initial seeds is set to 10 and any composite seed is derived by 1-3 operations on the initial seeds. For performance evaluation, we measure the computational cost (as the client and cloud CPU time), the communication cost (as the exchanged kilobytes between client and cloud), and the response time (as the total CPU time plus the communication time through a typical network at 2Mbps download rate and 1Mbps upload rate). For each experiment, 1,000 queries are executed and their average measurements are reported. Table I summarizes the parameter settings used in the experiments.

A. Basic Query Performance

For distance-range queries, we vary the distance threshold τ from 500 to $10,000$. Fig. 6(a) plots the client CPU, cloud CPU and the communication cost. The CPU times of both parties increase moderately as τ increases, and even for the largest distance $\tau = 10,000$ (which returns 40 objects on average), both times are below 100 ms. A similar trend is observed in the communication cost, where the total transmitted data are fewer than 100 kb in all τ settings. As a result, the query response time, shown in Fig. 6(b), increases moderately as τ increases. The same figure also shows the breakdown of the response time, and we find that the CPU times are dominated by the communication time, and their ratios of contributions are stable regardless of τ. This justifies the use of privacy homomorphism as an efficient encryption scheme.

For kNN queries, we vary the number of nearest neighbors k from 1 to 50. Fig. 7(a) plots the client CPU, cloud CPU and the communication cost. Similar to distance-range queries, the



(a) CPU Time and Comm. Cost (b) Query Response Time

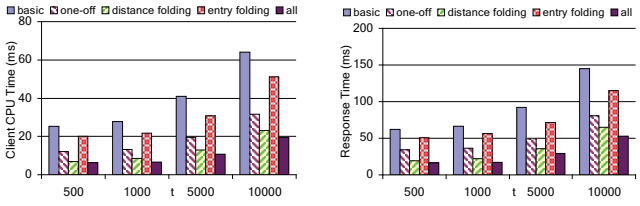
Fig. 7. kNN Query Performance

CPU times of both parties increase sub-linearly as k increases, and they never exceed 200 ms even for the largest k value. The trend in the communication cost coincides with those in CPU times, where the total transmitted data are fewer than 200 kb in all k settings. As a result, the query response time, shown in Fig. 7(b), increases moderately as k increases. The breakdown of the response time also shows the dominance of the communication time over CPU times, regardless of k.

B. Performance Optimizations

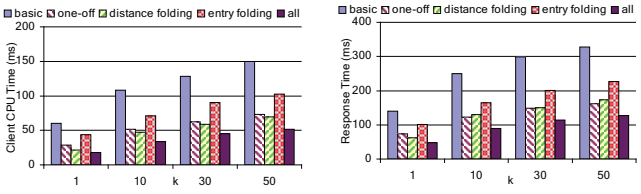
In this set of experiments, we implement the performance optimization techniques proposed in Section VIII, namely, the one-off local distance computing, distance folding and entry folding. For distance folding, L for the L-unfolded strategy is set to k; for entry folding, the minimum super entry size W is set to 5. Fig. 8 shows their individual performance gain as well as the total gain from the basic approach for distance-range queries. In Fig. 8(a), the client CPU time has been reduced by about 50% due to the one-off approach alone, which is in line with our analysis in Section VIII that the saving is approximately 60%. However, the distance folding alone (not shown in this figure) does not produce noticeable improvement. This can be explained as follows. The saving of distance folding on the basic two-phase approach only occurs in the first phase when $E(u-q)$ and $E(l-q)$ of those folded dimensions are saved from being sent; and for a 2D dataset, at most one dimension can be folded. On the other hand, the unfolding penalty is always a repeat of the second phase. Fortunately, the distance folding shows further improvement when implemented on top of the one-off approach. As is depicted in Fig. 8(a), they totally reduce the client CPU time by more than 70%. It is also noteworthy that as τ increases, only the gain of distance folding decreases slowly while the others are kept constant. This is because a larger τ makes fewer entries qualified for distance folding. Entry folding alone produces about 20% cost saving consistently, which justifies our motivation that a disk-based index node has more entries than necessary for efficient query processing. The combination of all three optimizations always achieves the top performance, which is around 75% saving. This further verifies the fact that all of them are effective. Similar trend and observations are found in the cloud CPU time and communication cost, so we omit these results in the interest of space. The query response time is shown in Fig. 8(b), which concludes that for a normal duty distance-range query ($\tau < 10,000$), the response time can keep below 50 ms.

Fig. 9 shows the performance gains for kNN queries.



(a) Client CPU Time (b) Query Response Time

Fig. 8. Performance Optimizations for Distance Range Query



(a) Client CPU Time (b) Query Response Time

Fig. 9. Performance Optimizations for kNN Query

Although most results are similar to those in the distance-range query, we observe that distance folding gains less and becomes even less as k increases, until eventually when $k = 50$, its response time is even worse than applying the one-off approach alone. This is due to the folding strategy used in different queries. For distance-range queries, it is naturally set to distance exceeding τ; for kNN queries, we use a simple heuristic $L = k$ for all k settings. However, as k increases, this heuristic seems less viable — a nonlinear estimation of L should be used. On the other hand, entry folding gains more than in distance range queries, and becomes even better as k increases. This shows that kNN queries normally needs fewer entries in a single node than distance-range queries.

XI. Conclusion

In this paper, we study the problem of processing private queries on indexed data for mutual privacy protection in a cloud environment. We present a secure index traversal framework, based on which secure protocols are devised for classic types of queries. Through theoretical proofs and performance evaluation, this approach is shown to be not only feasible, but also efficient and robust under various parameter settings. We believe this work steps towards practical applications of privacy homomorphism to secure query processing on large-scale, structured datasets. As for future work, we plan to extend this work to other query types, including top-k queries, skyline queries and multi-way joins. We also plan to investigate mutual privacy protection for queries on semi- or unstructured datasets.

Acknowledgment

This work was supported by Research Grants Council, Hong Kong SAR, China, under Projects HKBU FRG/08-09/II-48, FRG2/09-10/047, HKBU210808, HKBU211510.

References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. In *Proc. SIGMOD*, 2004.

[2] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Proc. CRYPTO*, 2007.

[3] Elisa Bertino, Beng Chin Ooi, Yanjiang Yang, and Robert H. Deng. Privacy and ownership preserving of outsourced medical data. In *Proc. ICDE*, 2005.

[4] Ernest F. Brickell and Yacov Yacobi. On privacy homomorphisms. In *Proc. 6th annual international conference on Theory and application of cryptographic techniques*, 1987.

[5] Chi-Yin Chow, Mohamed F. Mokbel, and Walid G. Aref. Casper*: Query processing for location services without compromising privacy. *ACM Transactions on Database Systems*, 34(4), 2009.

[6] Tran Khanh Dang. Privacy-preserving search and updates for outsourced tree-structured data on untrusted servers. In *Proc. iTrust*, 2005.

[7] Josep Domingo-Ferrer. A provably secure additive and multiplicative privacy homomorphism. In *Proc. 5th International Conference on Information Security*, 2002.

[8] W. Du and M. Atallah. Privacy-preserving cooperative statistical analysis. In *Proceedings of the 17th Annual Computer Security Applications Conference*, 2001.

[9] M. A. Soderstrand et al. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. New York: IEEE Press, 1986.

[10] R. Gallant, R. Lambert, and S. Vanstone. Faster point multiplication on elliptic curves with effient endomorphisms. In *Advances in Cryptology-Crypto 2001, LNCS 2139, Springer-Verlag*, 2001.

[11] P. Gastaldo, G. Parodi, and R. Zunino. Enhanced montgomery multiplication on dsp architectures for embedded public-key cryptosystems. *EURASIP Journal on Embedded Systems*, 2008(April), 2008.

[12] Bugra Gedik and Ling Liu. Location-privacy in mobile systems: A personalized anonymization model. In *Proc. ICDCS*, 2005.

[13] G. Ghinita, P. Kalnis, and S. Skiadopoulos. Prive: Anonymous location-based queries in distributed mobile systems. In *WWW*, 2007.

[14] Gabriel Ghinita, Panos Kalnis, Ali Khoshgozaran, Cyrus Shahabi, and Kian-Lee Tan. Private queries in location based services: Anonymizers are not necessary. In *Proc. of SIGMOD*, 2008.

[15] Oded Goldreich. *The Foundations of Cryptography – Volume 2*. Cambridge University Press, 2004.

[16] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proc. MobiSys*, 2003.

[17] Gisli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM TODS*, 24(2):265 – 318, 1999.

[18] Haibo Hu and Jianliang Xu. Non-exposure location anonymity. In *Proc. ICDE*, 2009.

[19] M. Kantarcioglu and C. Clifton. Privacy preserving k-nn classifier. In *Proc. ICDE*, 2005.

[20] A. Khoshgozaran and C. Shahabi. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In *Proc. SSTD*, 2007.

[21] Yaping Li and Minghua Chen. Privacy preserving joins. In *Proc. ICDE*, 2008.

[22] K. Liu, C. Giannella, and H. Kargupta. An attacker's view of distance preserving maps for privacy preserving data mining. In *Proc. PKDD*, 2006.

[23] Yinian Qi and Mikhail J. Atallah. Efficient privacy-preserving k-nearest neighbor search. In *Proc. ICDCS*, 2008.

[24] Mark Shaneck, Yongdae Kim, and Vipin Kumar. Privacy preserving nearest neighbor search. In *Proceedings of ICDM - Workshops*, 2006.

[25] E. Onur Turgay, Thomas B. Pedersen, Yucel Saygin, Erkay Savas, and Albert Levi. Disclosure risks of distance preserving data transformations. In *Proc. SSDBM*, 2008.

[26] Jaideep Vaidya and Chris Clifton. Privacy-preserving top-k queries. In *Proc. ICDE*, 2005.

[27] Hui Wang and Laks V. S. Lakshmanan. Efficient secure query evaluation over encrypted xml databases. In *Proc. VLDB*, 2006.

[28] Wai Kit Wong, Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. Secure knn computation on encrypted databases. In *Proc. SIGMOD*, 2009.

[29] X. Xiao and Y. Tao. Anatomy: simple and effective privacy preservation. In *Proc. VLDB*, 2006.

[30] Li Xiong, Subramanyam Chitti, and Ling Liu. Preserving data privacy in outsourcing data aggregation services. *ACM Transactions on Internet Technology*, 7(3), 2007.

[31] M. L. Yiu, G. Ghinita, C. S. Jensen, and P. Kalnis. Outsourcing search services on private spatial data. In *Proc. ICDE*, 2009.