International Journal of Research Studies in Science, Engineering and Technology
Volume 1, Issue 7, October 2014, PP 44-54
ISSN 2349-4751 (Print) & ISSN 2349-476X (Online)

# Outsourced Similarity Search on Metric Data Assets

**Vishal R. Shinde**
Asst. Prof., Computer Dept.
SSJCET, Asangaon
Thane, India
*mailme.vishalshinde@rediffmail.com*

**Sandesh N.Rangale**
Student (BE Comp).
SSJCET, Asangaon,
Thane, India
*rangales@gmail.com*

**Prathamesh P Sansare**
Student(BE Comp).
SSJCET, Asangaon,
Thane, India
*psansare8291@gmail.com*

**Mitesh T.Gavale**
Student (BE Comp).
SSJCET, Asangaon,
Thane, India
*mgavale8@gmail.com*

**Abstract:** *This paper considers a cloud computing setting in which similarity querying of metric data is outsourced to a service provider. The data is to be revealed only to trusted users, not to the service provider or anyone else. Users query the server for the most similar data objects to a query example. Outsourcing offers the data owner scalability and a low-initial investment. The need for privacy may be due to the data being sensitive (e.g., in medicine), valuable (e.g., in astronomy), or otherwise confidential. Given this setting, the paper presents techniques that transform the data prior to supplying it to the service provider for similarity queries on the transformed data. Our techniques provide interesting trade-offs between query cost and accuracy. They are then further extended to offer an intuitive privacy guarantee. Empirical studies with real data demonstrate that the techniques are capable of offering privacy while enabling efficient and accurate processing of similarity queries.*

**Keywords:** *query processing; security; integrity; protection; data owner; service provider; trusted client*

## 1. INTRODUCTION

Advances in digital measurement and engineering technologies enable the capture of massive amounts of data in fields such as astronomy, medicine, and seismology. The effort for data collection and processing, as well as its potential utility for research or business, create value for the data owner. He wishes to store them and allow access by himself, colleagues, and other (trusted) scientists or customers. Cloud computing services enable individuals and organizations to outsource the management of their data with ease and at low cost, even if they lack IT expertise. Cloud computing enables scalability with respect to storage and computational resources as the number of service requests grows, without the need for costly investments in hardware and maintenance. Consider the example of a real-estate company that owns a large database with descriptions of properties and their locations. The company (i.e., the private data owner) wishes to allow authorized users (e.g., paying customers) to query for properties situated within a certain geographical region. To save on hardware investments and maintenance costs, the data owner outsources the management of its dataset to a service provider (SP) that specializes in data storage and query processing. However, the SP may not be fully trusted, and could sell the data to a competitor. Furthermore, even if the SP is trusted, a malicious attacker can compromise the SP and gain unauthorized access to the data. To prevent such attacks, the data owner first encrypts the dataset according to a secret transformation and then uploads the encrypted data to the SP. Only authorized users who know the transformation are able to learn the property locations.

This can be supported by outsourced servers that offer low storage costs for large databases. For instance, outsourcing based on cloud computing is becoming increasingly attractive, as it promises pay-as-you go, low storage costs as well as easy data access. However, care needs to be taken to

safeguard data that is valuable or sensitive against unauthorized access. In this context, we call any item in a data collection an object, individuals with authorized access query users, and the entity offering the storage service the service provider. We illustrate the sensitivity issues with several scenarios. First, consider space programs such as the NASA Apollo program on the Earth's Moon1 or the ESA Mars Express2 that collect scientifically valuable and rare data. The NASA data is known to be private before it is released to the public. For example, time series data is collected from sensors to study the atmosphere's density. Such data is usually analyzed by the scientists involved in setting up the instruments, prior to being made available to the general community. At the early stage, access is restricted to authorized scientists for first analysis, because of the substantial efforts invested in building, testing, and deploying instruments, and in refining the data prior to use. Such valuable data needs protection when outsourced, to ensure that the investments by scientific groups are decently rewarded. To analyze the data, authorized scientists may search for similar patterns in collected time series, such as certain daily or hourly sub-sequences that indicate interesting phenomena. In this scenario, time series can be represented as vectors of values in chronological order. At query time, a user specifies an example time series q and wishes to obtain those time series most similar to q; the system then retrieves the time series p in the database with the minimum distance to q. As a second scenario, consider biologists analyzing DNA microarray data to understand the functioning of genes or gene groups, for instance from the Stanford Microarray Database3.A DNA microarray is a matrix obtained by subjecting gene samples (rows in the matrix) to different experimental conditions (columns in the matrix). Genes that follow the same expression pattern on all or a subset of the experiments might be part of a common control mechanism. For a given gene, its expression values form a query vector. Biologists query the database of experiments to identify those genes that are most similar to this specific expression pattern and that are therefore most likely to be linked to this gene. Generating DNA data is very costly due to the material and time invested.

## 2. RELATED WORK AND EXISTING MODEL

### 2.1 Related Work

*Indexing and NN Search in Metric Space*

We review metric indexing because our proposed methods provide metric indexing on the server for efficient processing. The R*-tree and the X-tree are well-known disk based indexes for multi-dimensional objects, where each object is modeled as a vector of coordinate values. Complex data objects (e.g., DNA sequence, time series) cannot be effectively represented by coordinate values. Instead, we model them in metric space, where a (black-box) distance function dist($p_i$; $p_j$) is used to compute the dissimilarity between objects $p_i$ and $p_j$ . The distance function dist(_) is said to be a metric if it satisfies symmetry, non-negativity, and the triangle inequality. Interested readers are referred to two excellent surveys , on metric space indexing. In this section, We only describe three representative indexing methods for a set P of metric space objects. They are the vantage-point tree (VP-tree),the multi-vantage-point (MVP-tree), and the M-tree.

The VP-tree is a binary tree built on P by utilizing the mutual distances among the objects in P. First, we choose an arbitrary object a 2 P as the root object, and then we determine the median distance r among the distances dist(a; p) from a to the objects p 2 P. Each object p 2 P satisfying dist(a; p) _ r is inserted into the left subtree of a, whereas the others are inserted into the right subtree of a. The tree is built in a top-down manner by applying the above construction procedure recursively to the subtrees of a. The MVP-tree is an extension of the VP-tree, so that each index node stores two anchor objects and has m2 subtrees (m being a parameter). The VP-tree (and MVP-tree) supports insertion/deletion of objects at the risk of an unbalanced tree.

The most popular metric space index is the M-tree (and its variant) due to its efficient support of object insertion/deletion. Each index entry e stores a minimum bounding sphere consisting of

(i) an anchor object e:a as the sphere center,

(ii) a covering radius e:r = maxfdist(e:a; p) j p 2 sub(e)g as the maximum distance from e:a to any object in the subtree of e. In addition, the entry stores

(iii) a pointer to its child node

Given a query object q and a set P of objects, the nearest neighbor (NN) query retrieves the object p 2 P such that dist(q; p) is minimized. The best-first paradigm is the state-of-the art method for performing NN search on a hierarchical metric space index (e.g., the M-tree). Given a query object q and an index entry e, the function mindist(q; e) is used to compute the (conservative) minimum distance between q and any object indexed by the subtree of e. The best-first search employs a min-heap H for organizing its encountered entries in ascending order of mindist(q; e). Initially, the entries in the root node of the tree are inserted into H. When an index entry e is deheaped from H, we access its child node and insert all entries of the node into H. The first object p that is deheaped from H, is guaranteed to be the NN of q.

Hashing techniques have also been proposed to answer the NN query efficiently. These techniques do not guarantee exact NN retrieval, but they return objects close enough to the NN in practice. The locality-sensitive hashing technique (LSH) is specifically designed for the Lx norm in the multidimensional space Rd; it is inapplicable to arbitrary metric spaces (e.g., edit distance over the domain of strings).

The distance-based hashing technique (DBH) is an extension of LSH for metric spaces. It takes as input two parameters:

i) the number A of bits, and

ii) the number C of hash tables. Let HT j be the j-th hash table, for indexing objects p 2 P based on their A-length bitmaps $BM_j(p)$. To compute the i-th bit of the bitmap $BM_j(p)$, we pick two anchor objects $a_i$; $b_i$ 2 P, and define the projection function as:

$$PJF_{a_i;b_i}(p) = \frac{dist^2(p; a_i) + dist^2(a_i; b_i) - dist^2(p; b_i)}{2 .dist(a_i; b_i).}$$

Then, we determine the value $r_i$ as the median value of $PJF_{a_i;b_j}(p)$. The i-th bit of $BM_j(p)$ is set to 0 if $PJF_{a_i;b_i}(p)$ _ $r_i$; otherwise, the bit is set to 1.

During the construction phase, we insert each object p 2 P into the hash table HT j according to the bitmap $BM_j(p)$.This step is repeated for all C hash tables. At query time, the user requests the hash table HT j to return all objects having the same bitmap as the bitmap $BM_j(q)$ of the query object q. Again, this step is repeated for all hash tables and eventually the closest of them (to q) is reported as the result.

Nevertheless, DBH has two limitations. First, it is possible that no hash table HT j contains any object with the same bitmap as the query bitmap $BM_j(q)$, leading to an empty result. Secondly, once the DBH structure is built (i.e., values of A and C are fixed), its query accuracy cannot be dynamically optimized by the user. We will address the above problems by developing a flexible hashing technique in that

(i) prevents empty results, and

(ii) allows the user to boost the query accuracy online by trading off communication cost.

## 2.2 Existing Model

### *2.2.1 Brute Force Algorithm*

This brute-force solution is the one we mentioned in the Introduction. In the offline construction phase, the data owner applies conventional encryption (e.g., AES) on each object and then uploads those encrypted objects to the server. At query time, the user needs to download all encrypted objects from the server, decrypt them and then compute the actual result. As mentioned, it is perfectly secure, but its query and communication cost are both prohibitively high, and pay-as you- go is not supported . In computer science, **brute-force search** or **exhaustive search**, also known as **generate and test**, is a very general problem-solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement. A brute-force algorithm to find the divisors of a natural number *n* would enumerate all integers from 1 to the square root of n, and check whether each of them divides *n* without remainder. A brute-force approach for the eight queens puzzle would examine all possible arrangements of 8 pieces on the 64-square chessboard, and, for each arrangement, check whether each (queen) piece can attack any other. While a brute-force search is simple to implement, and will always find a solution if it exists, its cost is proportional to the number of candidate solutions – which in many practical problems tends to grow very quickly as the size of the problem increases. Therefore, brute-force search is typically used when the problem size is limited, or when there are problem-specific heuristics that can be used to reduce the set of candidate solutions to a manageable size. The method is also used when the simplicity of implementation is more important than speed

### *2.2.2 Anonymization-based Solution (ANONY)*

This anonymization-based solution achieves data privacy by means of k-anonymity — the objects are generalized in such a way that each generalized object cannot be distinguished from k -1 other generalized objects. In the context of similarity search, it is able to confuse the ranking of transformed objects because k-1 of them have the same rank as the transformed object of the actual nearest neighbor. Thus, we still consider this solution as a competitor, even though k-anonymity is not a suitable privacy guarantee for our applications.

In the offline construction phase, the data owner applies a K-D tree partitioning technique on the dataset to obtain disjoint buckets such that each bucket contains at least k objects. For each bucket e, the data owner uploads to the server:

(i) e.MBR, the minimum bounding rectangle (MBR)of all objects inside the bucket and

(ii) encrypted strings of the tuples assigned to that bucket.

Let mindist(q; e:MBR) and maxdist(q; e:MBR) represent the minimum and maximum distance from the query object q to e:MBR (the MBR of the bucket e). At query time, the query user first obtains the MBRs of all buckets from the server, then computes the maximum distance from q to each bucket, and determines the smallest maximum distance (say, $x = \min_e \text{maxdist}(q; e)$). Next, the query user requests from the server all encrypted tuples from buckets e that satisfy mindist(q; e) <= x . Eventually, the query user decrypts those tuples in order to obtain the actual result.

Observe that the anonymization technique of LeFevre et al. is applicable only to multi-dimensional data. For arbitrary metric space data, the clustering-based anonymization technique of Aggarwal et al. can be applied. It represents each bucket as a minimum bounding sphere MBS) consisting of an anchor object and a covering radius, similar to M-tree index entry as described in Indexing and NN search in metric space.

The above anonymization-based solution has two limitations.First, the MBRs/MBSs of the buckets contain substantial empty space, due to the curse of dimensionality. Thus, the derived upper NN distance bound x can be loose, triggering a large number of buckets to be retrieved. Second, the solution still allows the server to know the MBRs/MBSs of the buckets,which are located in the same

space as the original objects.In contrast, our proposed methods in Section MPT and FDH do not permit the server to know any information in the original space; the anchors are converted to IDs and distance information is transformed to numbers or bitmaps.

R-tree; however, no solutions were proposed for the NN query on those encrypted indexes.

## 3. MODULES

### 3.1 Outsourcing Data

It consists of three entities: a data owner, a trusted query user, and an untrusted server. On the one hand, the data owner wishes to upload his data to the server so that users are able to execute queries on those data. On the other hand, the data owner trusts only the users, and nobody else (including the server). The data owner has a set P of (original) objects (e.g., actual time series, graphs, strings), and a key to be used for transformation. First, the data owner applies a transformation function (with a key) to convert P into a set P0 of transformed objects, and uploads the set P0 to the server . The server builds an index structure on the set P0 in order to facilitate efficient search. In addition, the data owner applies a standard encryption method (e.g., AES) on the set of original objects; the resulting encrypted objects (with their IDs) are uploaded to the server and stored in a relational table (or in the file system).

### 3.2 Nearest Neighbor Query

In this module, our research objective is to design a transformation method that meets the following requirements:

1) Even in the worst case where the attacker knows the inverse of the transformation function, the attacker can only estimate the original object p from the transformed object p' with bounded precision.

2) It enables high query accuracy.

3) It enables efficient query processing in terms of communication cost.

4) It supports the insertion and deletion of objects.

*A. Brute-force Secure Solution (BRUTE):-*

This brute-force solution is the one we mentioned in the Introduction. In the offline construction phase, the data owner applies conventional encryption (e.g., AES) on each object and then uploads those encrypted objects to the server. At query time, the user needs to download all encrypted objects from the server, decrypt them and then compute the actual result. As mentioned, it is perfectly secure, but its query and communication cost are both prohibitively high, and pay-as you- go is not supported.

### 3.3 Anonymization - Based Solution (ANONY)

This anonymization-based solution achieves data privacy by means of k-anonymity — the objects are generalized in such a way that each generalized object cannot be distinguished from k - 1 other generalized objects. In the context of similarity search, it is able to confuse the ranking of transformed objects because k - 1 of them have the same rank as the transformed object of the actual nearest neighbor. Thus, we still consider this solution as a competitor, even though k-anonymity is not a suitable privacy guarantee for our applications.

## 4. PROPOSED WORK

We introduce approaches that shift search functionality to the server. The proposed Metric Preserving Transformation (MPT) stores relative distance information at the server with respect to a private set of anchor objects. This method guarantees correctness of the final search result, but at the cost of two

rounds of communication. The proposed Flexible Distance-based Hashing (FDH) methods finishes in just a single round of communication, but does not guarantee retrieval of the exact result.

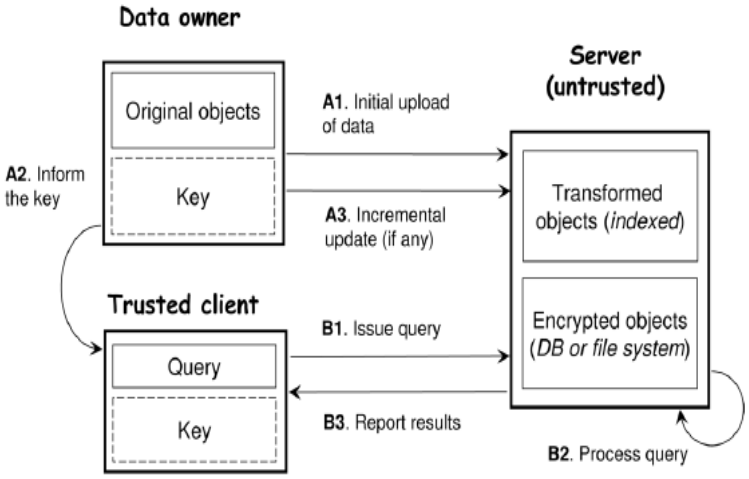



**Fig-** *System Architecture*

### 4.1 Encrypted Hierarchical Index Search (EHI)

This section presents a client algorithm, called encrypted hierarchical index (EHI), for performing NN search on an encrypted hierarchical index stored at the server. This method offers perfect data privacy for the data owner, but it incurs multiple communication round trips during query processing. In the literature, algorithms have been developed for processing range queries on encrypted B+-tree and encrypted R-tree; however, no solutions were proposed for the NN query on those encrypted indexes.

**Algorithm** EHI-Search ( Query object $q$, Encryption Key $\mathcal{CK}$, Integer $\lambda$ )
1: request the server for the (encrypted) root node $L_{root}$;
2: $H$:=new min-heap; $p_{nn}$:=NULL;
3: $\gamma$:=min$_{e \in L_{root}}$ $maxdist(q, e)$; ▷ derive NN distance bound
4: **for each** entry $e \in L_{root}$ such that $mindist(q, e) \leq \gamma$ **do**
5: insert the entry $\langle e, mindist(q, e) \rangle$ into $H$;
6: **while** $H$ is not empty and its top entry's key $\leq \gamma$ **do**
7: pop next $\lambda$ entries from $H$ and insert them into a set $S$;
8: request the server for each (encrypted) child node of $S$;
9: **for each** retrieved node $L_{cur}$ **do**
10: **if** $L_{cur}$ is a leaf node **then** ▷ check for closer objects
11: update $\gamma$ and $p_{nn}$ by using objects in $L_{cur}$;
12: **else** ▷ expand the entries of $L_{cur}$
13: $\gamma$:=min$\{\gamma,$ min$_{e \in L_{cur}}$ $maxdist(q, e)\}$;
14: **for each** $e \in L_{cur}$ such that $mindist(q, e) \leq \gamma$ **do**
15: insert the entry $\langle e, mindist(q, e) \rangle$ into $H$;
16: **return** $p_{nn}$ as the result;

**Fig:** *Algo for searching indexes*

### 4.2 Metric Preserving Transformation (MPT)

In this section, we develop a method, called metric preserving transformation (MPT), for evaluating the NN query. Unlike the EHI method, MPT incurs only 2 rounds of communication during the query phase. The basic idea behind MPT is to pick a small subset of the dataset P as the set of anchor objects and then assign each object of P to its nearest anchor. For each object p, we compute its distance dist( ai;p) from its anchor ai and then apply an order-preserving encryption function OPE on the distance value. These order-preserving encrypted distances will be stored in the server and utilized for processing NN queries. A function OPE : R !R is said to be order preserving if it guarantees that OPE( x) > OPE( x') for any values x;x' that satisfy x > x'.

**Algorithm** MPT-Build ( Dataset $P$, Encryption Key $\mathcal{CK}$, Integer $A$ )

use a heuristic of Ref. [11] to select a set of $A$ anchor objects from $P$;
Integer $B$:=$\lceil |P|/A \rceil$;
use a heuristic of Ref. [11] to assign each data object of $P$ to an anchor object, subject to the capacity constraint $B$;
**for** $i$:=1 to $A$ **do**
    let $a_i$ be the $i$-th anchor object;
    let $a_i.S$ be the set of objects assigned to the anchor $a_i$;
    $r_i$:=$\max_{p \in a_i.S} dist(a_i, p)$; ▷ compute covering radius
    **for each** object $p \in a_i.S$ **do**
        send the tuple $\langle p.id, \mathcal{OPE}(dist(a_i, p)), \mathcal{ECR}(p, \mathcal{CK}) \rangle$ to the server;

**Fig:** *Algo for Data-Owner*

**Algorithm** MPT-Search ( Query object $q$, Encryption Key $\mathcal{CK}$, Integer $\theta$, Integer $A$, Pairs $\{(a_i, r_i)\}$ )
*/* Distance bounding phase */*
$\gamma$:=$\min_{i \in [1,A]} dist(q, a_i)$; ▷ initial bound of NN distance
let $a_{near}$ be the anchor leading to the $\gamma$ value;
request the server for $\theta$ random tuples whose anchor ID equals to that of $a_{near}$;
let $S_{samp}$ be the set of decrypted objects from the received tuples;
**for each** $p \in S_{samp}$ **do**
    $\gamma$:=$\min\{\gamma, dist(q, p)\}$; ▷ refined bound of NN distance
*/* Candidate retrieval phase */*
**for** $i$:=1 to $A$ **do**
    **if** $mindist(q, (a_i, r_i)) \leq \gamma$ **then**
        request the server for all tuples (with anchor ID as $a_i$) whose $\mathcal{OPE}(dist(a_i, p))$ falls into the range $[\mathcal{OPE}(dist(q, a_i) - \gamma), \mathcal{OPE}(dist(q, a_i) + \gamma)]$;
let $S_{cand}$ be the set of decrypted objects from the received tuples (of the above request);
**return** the object $p \in S_{cand}$ with the minimum $dist(q, p)$ value as the final result;

**Fig:** *Algo for Data-Client*

### 4.3 Flexible Distance-based Hashing (FDH)

In this section, we propose a hashing-based technique, called flexible distance-based hashing (FDH), for processing the NN query. The main advantage of this technique is that the server always returns a constant-sized candidate set (in one communication round). The client then refines the candidate set to obtain the final result. Even though FDH is not guaranteed to return the exact result, the final result is very close to the actual NN in practice.

**Algorithm** FDH-Build ( Dataset $P$, Encryption Key $\mathcal{CK}$, Integer $A$ )

**for** $i$:=1 to $A$ **do** ▷ key generation
    choose an object randomly from $P$ as an anchor object $a_i$;
    find the distance value $r_i$ such that half of objects $p \in P$ satisfy $dist(a_i, p) \leq r_i$;
**for each** object $p \in P$ **do**
    compute the encryption $\mathcal{ECR}(p, \mathcal{CK})$;
    compute $\mathcal{BM}(p)$;
    send the tuple $\langle p.id, \mathcal{BM}(p), \mathcal{ECR}(p, \mathcal{CK}) \rangle$ to the server;

**Fig:** *Algo for Data-Owner*

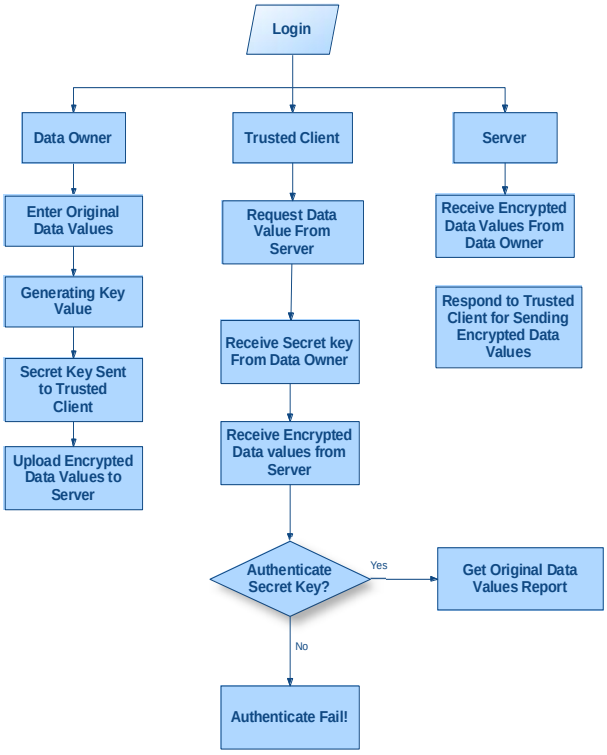
**Algorithm** FDH-Search ( Query object $q$, Encryption Key $\mathcal{CK}$, Integer $\theta$, Integer $A$, Pairs $\{(a_i, r_i)\}$ )

compute the query bitmap $\mathcal{BM}(q)$;
request the $\theta$ tuples $\langle p.id, \mathcal{BM}(p), \mathcal{ECR}(p, \mathcal{CK}) \rangle$ with the lowest Hamming distance from $q$ from the server (ties are resolved arbitrarily);
let $S$ be the set of decrypted objects from the received tuples;
**return** the object $p \in S$ with the minimum $dist(q, p)$ value as the final result;

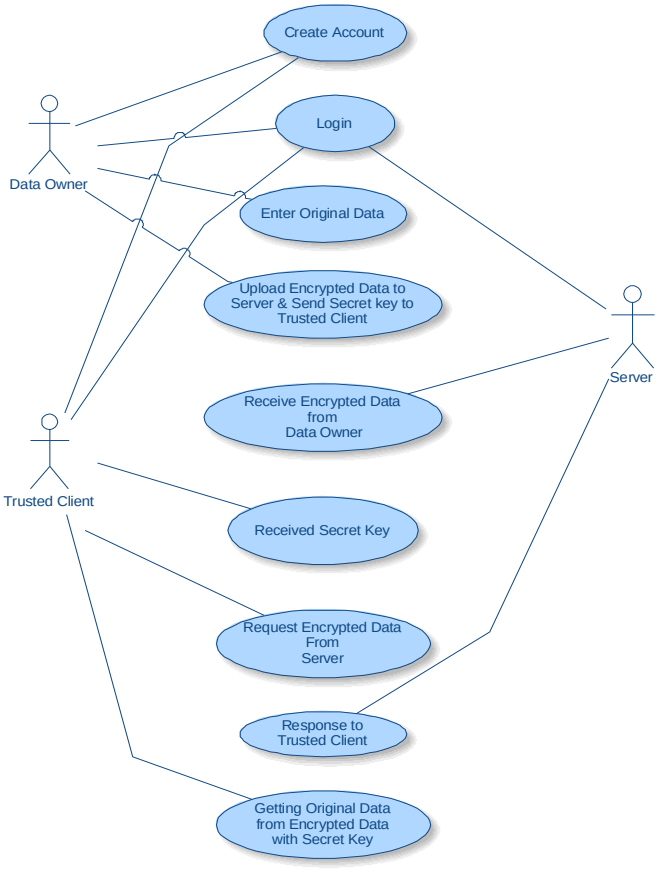
**Fig:** *Algo for Data-Client*

# 5. SYSTEM DESIGN

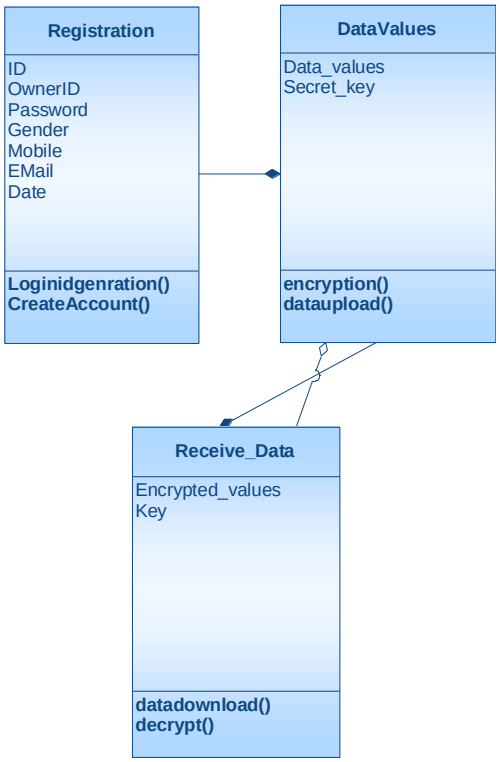
## 5.1 Data Flow Diagram



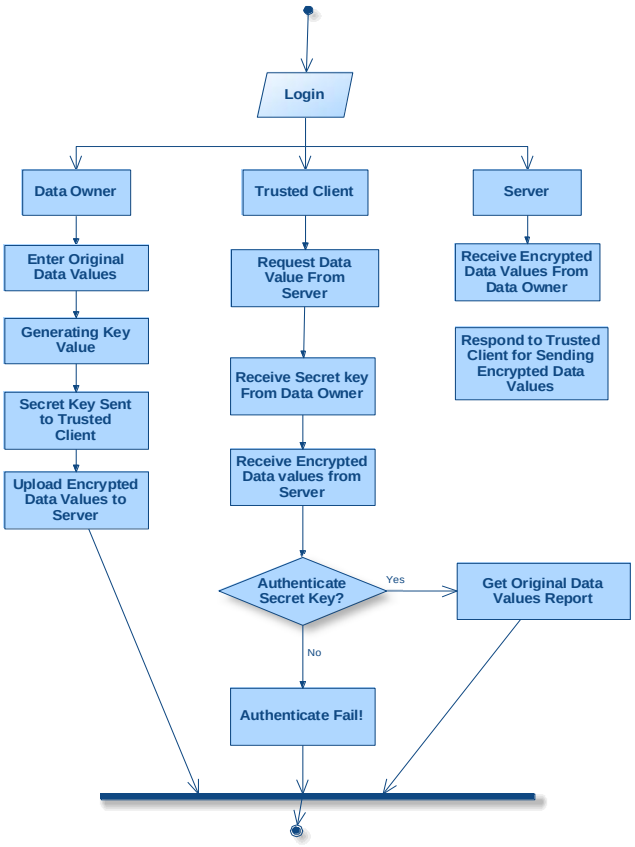

## 5.2 Use Case Diagram



## 5.3 Class Diagram




## 5.4 Activity Diagram



## 6. CONCLUSION

Existing solutions either offer query efficiency at no privacy, or they offer complete data privacy while sacrificing query efficiency. It is attractive to be able to maintain data confidentiality with respect to untrusted parties, including the service provider. The paper presents methods to encode a dataset such that only authorized users can access the content, while the service provider "blindly" evaluates queries,without seeing the actual data.It is important for the data owner to choose an appropriate transformation method that best matches the requirements. We are proposing three transformation methods. The first method is encrypted hierarchical index search algorithms gives the final result multiple rounds of communication. The second method is Metric Preserving Transformation method guarantees correctness of the final search result, but at the cost of two rounds of communication. The third proposed method is Flexible Distance-based Hashing methods finishes in just a single round of communication, but does not guarantee retrieval of the exact result. But actual result is very close to the exact result. This transformation methods achieve different trade-offs between the data privacy and query efficiency.

## REFERENCES


[1] G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy,D. Thomas, and A. Zhu. Achieving Anonymity via Clustering. In PODS, pages 153–162, 2006.

[2] R. Agrawal, P. J. Haas, and J. Kiernan. Watermarking Relational Data: Framework, Algorithms and Analysis. VLDB J., 12(2):157–169, 2003.

[3] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-Preserving Encryption for Numeric Data. In SIGMOD, pages 563–574, 2004.

[4] R. Agrawal and R. Srikant. Privacy-Preserving Data Mining. In SIGMOD, pages 439–450, 2000.

[5] C. A. Ardagna, M. Cremonini, E. Damiani, S. D. C. di Vimercati, and P. Samarati. Location Privacy Protection Through Obfuscation-Based Techniques. In DBSec, pages 47–60, 2007.

[6] V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios. Nearest Neighbor Retrieval Using Distance-Based Hashing. In ICDE, pages 327–336, 2008.

[7] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In SIGMOD, pages 322–331, 1990.

[8] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree : An Index Structure for High-Dimensional Data. In VLDB, pages 28–39, 1996.

[9] T. Bozkaya and Z. M. ¨Ozsoyoglu. Indexing Large Metric Spaces for Similarity Search Queries. TODS, 24(3):361–404, 1999.

[10] E. Ch´avez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroqu´ın. Searching in Metric Spaces. ACM Comput. Surv., 33(3):273–321, 2001.

[11] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In VLDB, pages 426– 435, 1997.

[12] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Sama-rati. Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs. In CCS, pages 93–102, 2003.

[13] M. Dunham. Data Mining: Introductory and Advanced Topics. Prentice Hall, 2002.

[14] C. Faloutsos and K.-I. Lin. FastMap: A Fast Algorithm for Indexing Data-Mining and Visualization of Traditional and Multimedia Datasets In SIGMOD, pages 163–174, 1995.

[15] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K. L. Tan Private Queries in Location Based Services: Anonymizers are not Necessary. In SIGMOD, pages 121–132, 2008.

[16] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In VLDB, pages 518–529, 1999.

[17] H. Hacig ¨um ¨us, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over Encrypted Data in the Database-Service-Provider Model. In SIGMOD pages 216–227, 2002.

[18] H. Hacig ¨um ¨us, S. Mehrotra, and B. R. Iyer. Providing Database as a Service. In ICDE, pages 29–40, 2002.

[19] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What Is the Nearest Neighbor in High Dimensional Spaces? In VLDB, pages 506–515, 2000

[20] G. R. Hjaltason and H. Samet. Index-Driven Similarity Search in Metric Spaces. TODS, 28(4):517–580, 2003.

[21] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Z. 0003. iDistance: An Adaptive B+-Tree Based Indexing Method for Nearest Neighbor Search. TODS, 30(2):364–397, 2005.

[22] C. T. Jr., A. J. M. Traina, B. Seeger, and C. Faloutsos. Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes. In EDBT, pages 51–65, 2000.

[23] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the Privacy Preserving Properties of Random Data Perturbation Techniques. In ICDM, pages 99–106, 2003.

[24] A. Khoshgozaran and C. Shahabi. Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy. In SSTD, pages 239–257, 2007.

[25] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian Multidi- mensional K-Anonymity. In ICDE, page 25, 2006.

[26] T. Seidl and H. P. Kriegel. Optimal Multi-step k-Nearest Neighbor Search. In SIGMOD, pages 154–165, 1998.

[27] L. Sweeney. k-Anonymity: A Model for Protecting Privacy. IJUFKS, 10(5):557–570, 2002.

[28] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis. Secure k-NN Computation on Encrypted Databases. In SIGMOD, pages 139–152, 2009.

[29] P. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In SODA, pages 311–321, 1993.

[30] M. L. Yiu, I. Assent, C. S. Jensen, and P. Kalnis. Outsourced Similarity Search on Metric Data Assets. DB Technical Report TR-28, Aalborg University, 2010.