Secure Metric-Based Index for Similarity Cloud

Stepan Kozak, David Novak, and Pavel Zezula

Masaryk University, Brno, Czech Republic
{xkozak1,david.novak,zezula}@fi.muni.cz

Abstract. We propose a similarity index that ensures data privacy and thus is suitable for search systems outsourced in a cloud. The proposed solution can exploit existing efficient metric indexes based on a fixed set of reference points. The method has been fully implemented as a security extension of an existing established approach called M-Index. This Encrypted M-Index supports evaluation of standard range and nearest neighbors queries both in precise and approximate manner. In the first part of this work, we analyze various levels of privacy in existing or future similarity search systems; the proposed solution tries to keep a reasonable privacy level while relocating only the necessary amount of work from server to an authorized client. The Encrypted M-Index has been tested on three real data sets with focus on various cost components.

1 Introduction

With more and more data being collected in all kinds of scientific processes (medicine, astronomy, etc.) or commercial applications such as social networking and on-line marketing, searching in large data sets became one of the key tasks performed these days. Such data often does not provide sufficient meta-data description, therefore in many applications similarity search is more important than an exact match or keyword search.

Since similarity search itself is a very resource demanding process, the trend is to outsource such services to 3rd party cloud providers. Outsourcing to a cloud provides many advantages such as low initial investments, low storage costs and a very good scalability (more storage or computational power can be added on the fly, when it's needed – so called pay-as-you-go principle).

We can see two possible scenarios of outsourcing similarity search. In the first case, user has their similarity search technology and wants to use the hardare of a cloud infrastructure provider. In the second scenario, a similarity search service provider makes the technology available for end users so they can use the engine without an actual knowledge of the technology. We observe an increasing trend of the latter case and we will refer to this as *similarity cloud*.

In both scenarios, users might not want to expose all their data which might be sensitive (e.g. medicine data) or valuable (e.g. data collected from a scientific research), to a 3rd party provider, which is, in general, untrusted. In these cases, privacy of the data is of high importance. Hence the similarity cloud has to provide mechanisms which allow applying privacy requirements of the end users.

W. Jonker and M. Petković (Eds.): SDM 2012, LNCS 7482, pp. 130–147, 2012.

The general objective of outsourcing is to move all the resource-demanding process to the cloud and allow authorized clients to run search queries and get answers. Intuitively, involving an encryption will negatively influence the efficiency of the service. The less the cloud servers know about the data, the less efficient indexing techniques can be used on the server side. A suitable balance between sufficient level of privacy and performance has to be found.

The topic of secured similarity search has been studied recently [1–4]. Some of these works focus on keyword similarity indexing [1–3] and the work of Yiu et al. [4] studies the indexing based on general metric principles. We consider the paper pioneering in the area that we expect to become more important and we consider this research direction promising.

In this work, we try to specify more precisely the application scenarios and goals of the secure similarity search; our analysis results in a taxonomy of privacy in similarity search services. We propose a novel metric-based technique that ensures reasonable level of privacy while relocating only the necessary amount of work from server to an authorized client. The proposed approach can exploit existing efficient metric indexes that are based on a fixed set of reference objects (pivots). Our method has been implemented utilizing M-Index [5, 6], an established indexing and searching approach that is currently being used in several real applications.

The M-Index and other similar approaches treat the data space as a *metric space* $(\mathcal{D}, d)$, where $\mathcal{D}$ is a *domain* of objects and d is a total *distance function* $d : \mathcal{D} \times \mathcal{D} \longrightarrow \mathbb{R}$ satisfying metric postulates (non-negativity, identity, symmetry, and triangle inequality) [7]. The set of indexed objects $X \subseteq \mathcal{D}$ is typically searched by the *query-by-example* paradigm, for instance by the *range query* $R(q,r) = \{o \in X \mid d(q,o) \leq r\}$, $q \in \mathcal{D}$ or by the *nearest neighbors query* k-NN(q) covering the k objects from X with the smallest distances to given $q \in \mathcal{D}$.

Contribution and Structure of This Paper

- In Section 2, we describe the goals of secure similarity search and we define the taxonomy of privacy in similarity clouds. In Section 3, we describe some of the existing approaches.
- We propose a mechanism for ensuring privacy in the M-Index and similar approaches in Section 4. Further, we discuss privacy and efficiency of the proposed solution and describe its prototype implementation.
- We performed experiments on three real-life datasets in order to measure the efficiency degradation caused by our privacy ensuring approach; we also analyzed individual cost components in real client-server environment (Section 5). The paper concludes in Section 6 with suggestions for future work.

2 Problem Definition

In this section, we establish the terminology used throughout this paper and define the main objectives of secured similarity search in an outsourced (client-server) setting.

2.1 Terminology

Raw Data. The original (sensitive) data objects to be indexed and searched. For example binary files of images or any other data collection.

Metric Space Objects (MS Objects). Metric space descriptors extracted from the raw data, each descriptor has a reference to the corresponding raw object; they are compared by a metric function. In some cases, the raw data and the MS objects are identical (for instance, gene sequences or other biomedical data); in other cases, several MS objects are extracted from one raw object.

Secret Key. Encryption key used to encrypt the raw data and/or MS objects. Encryption key is also used for authorized querying of the data.

Data Owner. Subject outsourcing the search service, owner of the data.

Server. Server(s) of the 3^{rd} party similarity cloud. From the data owner's point of view, server is not trusted because it is not fully controlled by the data owner (server can be attacked and data from it leaks to an attacker). Therefore server should not have the access to the original (unencrypted) data and should have as less as possible information about the MS objects.

Authorized Client. Client authorized by the data owner to use the search service (i.e. client having the secret key).

Attacker. Any potential malicious user with purpose of getting the data.

2.2 Objectives

In general, the process of outsourcing is the following: In the construction phase, the data owner creates the MS objects from the original raw data, sends these MS objects to a similarity cloud for indexing and the raw data to a data storage. In the search phase, any authorized client can query the similarity cloud to obtain IDs of the relevant objects referring to original data objects, that can be subsequently retrieved from the raw data storage. Scheme of such outsourced similarity system is depicted in Figure 1. General objectives of outsourced secure similarity search can be formulated as follows:

- Resource demanding process (the search itself) should be performed on the server side as much as possible (clients that query the server might be simple devices without big computational power).
- Communication cost between the client and the server should be as low as possible (in optimal case, client sends only initial search request and then receives result from the server).
- Data should be stored on the server in a secure way so that a potential attacker can gain as little information about the data as possible.

Intuitively, the security requirement goes against the efficiency objective. If most of the computations should be performed on server side, the server has to have enough information about the data to process such task efficiently. Hence, the right balance between the security and efficiency should be found for each specific application setting.

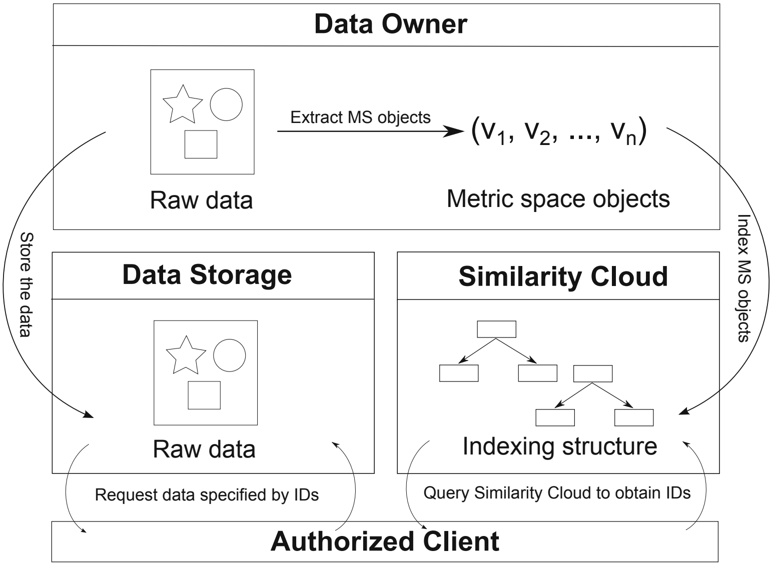

Fig. 1. General scheme of outsourced similarity search

2.3 Levels of Privacy

In this section, we introduce and discuss several general approaches to a (secure) outsourced similarity search as we see it.

No Encryption. This is the fundamental setting suitable for not sensitive data or data stored in a completely trusted environment (own servers, etc.). It is also the most efficient solution, because all the work can be done on the server and no additional encryption/decryption processes have to be employed. All advanced indexing techniques can be applied on the server side without loss of efficiency.

Raw Data Encryption. Another approach is to extract the MS objects from the raw data and build a standard indexing structure on these MS objects; then the raw data can be encrypted with some symmetric cryptosystem (AES or any other) and uploaded to the cloud data storage. The similarity search itself can be performed without any change, the whole search process can be done on the server (the index and MS objects are kept unencrypted). After the search, the raw data storage returns encrypted result data to the client for decryption.

This approach is good from the performance point of view, because the resource demanding process (similarity search with expensive distance computations) is done on the server side, while the client decrypts the final result (which can also be a time consuming process, but this can be hardly omitted since the authorized client is naturally the only party having the decryption key).

From the security point of view, this approach is not applicable if the MS objects are also sensitive or identical to the raw data. And even if not, the knowledge of the metric space (distances between objects) might be possibly exploited to get information about the raw data. This case is considered below.

MS Objects Encryption. In this setting, both raw data and MS objects are encrypted. However, once the indexing structure cannot fully access the MS objects, it cannot use pruning or filtering techniques which may reduce the system efficiency. Therefore, additional information about the MS objects has to be provided to the indexing structure, so that at least some filtering can be done on the server side (otherwise the server would degrade to a simple data storage).

MS Objects and Their Distribution Encryption. Encrypting individual metric space objects does not hide all the information from a potential attacker. Server has unencrypted index structure with distances between objects and other distance information between nodes within the indexing structure (the exact type and amount of information depends on specific indexing mechanism). From this, an attacker can possibly learn some information about distribution of the data. The aim in some scenarios might be to encrypt also this information.

Note: Since raw data can be stored (and encrypted) separately and indexing structures can only contain MS objects with a reference to the original resource, we further consider the raw data always encrypted and discuss only the process of securing MS objects within the indexing structures (therefore we focus on the last two approaches of the above list).

3 Existing Approaches

The topic of secured similarity search is being studied and there are several results in this field. Several techniques for the similarity (keyword) search in the text data has been proposed and analyzed [1–3]. However, all the solutions were designed only for specific type of data with specific distance function to measure the similarity. There is also a recent work concerning secure multidimensional interval queries in over outsourced attribute data [8]. On the other hand, general metric secure similarity search, where no additional information about the data or the distance function is known, is a relatively new research area.

Naturally, there exists a straightforward solution: The data owner can encrypt every object and send only the encrypted objects to the server without any additional information. During the search phase, an authorized client downloads all the objects from the server, decrypts them and performs the search. This solution achieves perfect security, however it is clear that it cannot be used in real applications because it has extremely high communication cost and it loses all the advantages of cloud environment (especially scalability).

Few advanced techniques for outsourcing similarity search (with general metric space approach) have been proposed. In this section, we provide brief overview of two of the techniques of Yiu et al. [4]. Both these solutions encrypt the MS objects and hide the data distribution (see Section 2.3).

3.1 Encrypted Hierarchical Index Technique

First approach is called Encrypted hierarchical index search (EHI) and it works as follows. Efficient indexing structure is build upon the MS objects, all the nodes of the indexing structure are encrypted with an arbitrary symmetric cipher, address of the root node is public. The search procedure logic is implemented on the client, which requests nodes of the structure from the server, decrypts the node, processes the objects stored in this node and requests other node until it completes the operation.

This approach has obvious advantage in its security, because a potential attacker cannot gain any information either about the data itself or about the metric space, since everything is stored encrypted. Another advantage is a straightforward implementation and robust design, because we can use many indexing structures practically without change of its internal structure.

However, there is a cost we have to pay for this high security level: communication costs (a lot of traffic is between client and the server) and efficiency of the search procedure. Since all the nodes of the indexing structure are encrypted, server cannot traverse through the structure and can only serve as a storage, sending the client what was requested. All the time-consuming search operations have to be implemented on the client side, and the client has to perform a lot of encryption/decryption operations which (in general) might be very resource demanding as well. This approach seems to be only slightly better than the trivial solution described above.

3.2 Metric-Preserving Transformation

EHI with the drawbacks mentioned above might not be applicable in some scenarios, because the client can be a device with limited resources (e.g. smart phone or even simpler device with less computational resources). Yiu et al. [4] propose Metric-Preserving Transformation (MPT) technique which uses an order-preserving encryption function. For details see the paper [4]. The goal of MPT was to reduce communication cost of EHI and pass part of the search work to the server while preserving sufficient privacy of the data.

However, to achieve sufficient security level, one has to have a representative sample of the data collection before the indexing structure is built and sent to the server (it is necessary for the order-preserving encryption function to work properly). This could be a problem in dynamic data sets where the collection is often changing.

4 Encrypted Metric-Based Index

In this section, we propose an approach that can add privacy to metric indexes that are based on distance permutations of a fixed set of reference objects [9–11, 6]. Our approach is introduced as an extension of a structure called M-Index [5, 6] (because it enables both precise and approximate similarity search and has other advantages) but it can be generalized straightforwardly to any

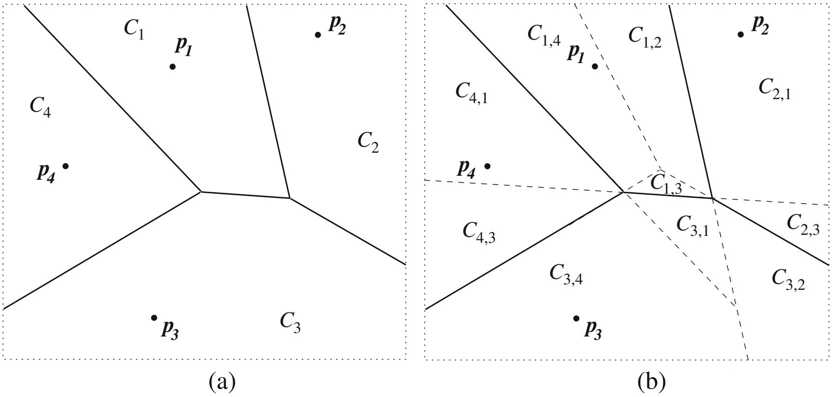

Fig. 2. Principle of recursive Voronoi partitioning: first level (a), second level (b)

other member of this class of metric indexes. Let us first briefly introduce the standard M-Index.

4.1 M-Index

The M-Index [5, 6] is a dynamic disk-efficient metric-based index that uses a set of reference objects (pivots) $p_1, \dots, p_n \in \mathcal{D}$ in order to partition the indexed set $X \subseteq \mathcal{D}$. Namely, the *recursive Voronoi partitioning* is used: at the first level, each object $o \in X$ is assigned to its closest pivot p_i creating Voronoi cell C_i; at the second level, data from C_i are partitioned further using pivots $p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_n$ forming cells $C_{i,j}$, etc. Figure 2 depicts an example of such partitioning with four pivots to the first and second level, respectively. This approach can also be formalized as *pivot permutations* [12, 13]: For each object $o \in X$, let $(\cdot)_o$ be permutation of indexes $\{1, \dots, n\}$ such that $\forall i, j \in \{1, \dots, n\}$:

$$(i)_o < (j)_o \Leftrightarrow d(p_{(i)_o}, o) < d(p_{(j)_o}, o) \vee d(p_{(i)_o}, o) = d(p_{(j)_o}, o) \wedge i < j.$$

Sequence $p_{(1)_o}, \dots, p_{(n)_o}$ is then ordered with respect to distances between the pivots and o. The M-Index uses prefixes of this permutation to index o.

Since neither this space partitioning nor the data distribution are uniform, the M-Index has a *dynamic* variant that further partitions only the cells that exceed certain data volume limit. The M-Index then maintains a dynamic *Voronoi cell tree* to keep track of actual depth for individual cells. The schema of this tree is sketched in Figure 3.

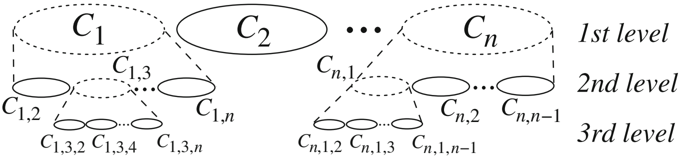

Fig. 3. M-Index Voronoi cell tree

The M-Index can evaluate the range and k-NN queries using several space pruning and filtering mechanisms [5] – only part of the Voronoi cells has to be accessed to ensure returning of all the required objects. Beside this *precise* query evaluation strategies that returns the precise answer A^P, the M-Index can also evaluate k-NN(q) queries in *approximate* manner [6] (not all objects from A^P are always returned while the search costs can be significantly reduced). In this case, the algorithm heuristically specifies a candidate set $S_C \subseteq X$ of the objects that are "close" to query q; this set S_C is then refined by evaluating distances $d(q, o)$, $\forall o \in S_C$ and selecting the k closest objects that form the approximate search result A. The size of the candidate set S_C can be parametrized. The *recall* is a common measure to quantify the quality of the result A with respect to precise answer A^P: $\mathit{recall}(A) = \frac{|A \cap A^P|}{|A^P|} \cdot 100\%$.

4.2 Encrypted M-Index

Our solution exploits the fact that only the prefix of the pivot permutation is used for indexing any object $o \in \mathcal{D}$ within M-Index. No additional distances need to be computed during insertion of o into the index. The M-Index search algorithms also need only query-pivot distances (or their permutation) in order to form the candidate set S_C. Therefore, the set of pivots can be part of the private information known only by the data owner (and shared with authorized clients). The final refining has to be done on the client, but S_C (created on the server) is supposed to be significantly smaller than the whole collection X and it can be pre-ranked. These operations are formalized in the following paragraphs and schematically depicted in Figures 4 and 5.

Data Insertion. In the construction phase, for each object $o \in X$, the data owner calculates pivot permutation, encrypts the object o (using arbitrary symmetric cipher) and sends encrypted object e along with its pivot permutation to the server. The server-side M-Index locates the leaf node of the dynamic cell tree that corresponds to given pivot permutation (see Figure 3); object e together with the pivot permutation is stored in this node and, if necessary, this leaf node is split (again, only the permutations stored with the objects are necessary for the split). This insert procedure is sketched in Figure 4 and its client part is formalized in Algorithm 1.

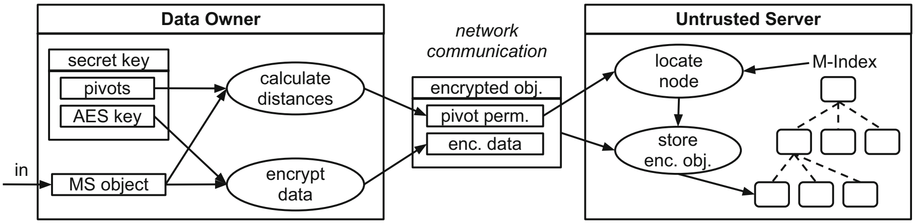

Fig. 4. Schema of the insert operation into encrypted M-Index

Algorithm 1. Encrypted M-Index insert algorithm

```
Input: $o \in \mathcal{D}$, secretKey containing pivots $p_1, \ldots, p_n$ and cipher key
calculate $d(o, \text{secretKey}.p_i)$, $\forall i \in \{1, \ldots, n\}$;
init new enc. object $e$;  /* e := struct {distances, permutation, data} */
if precise strategy is used then
    $e.distances \leftarrow d(o, \text{secretKey}.p_1), \ldots, d(o, \text{secretKey}.p_n)$;
else
    sort the distances to find permutation $(1)_o, \ldots, (n)_o$;
    $e.permutation \leftarrow (1)_o, \ldots, (n)_o$;
$e.data \leftarrow \text{secretKey}.encrypt(o)$;          /* store encrypted data only */
server.$insert(e)$;
```

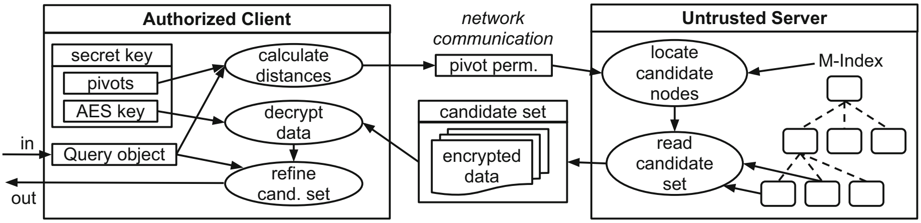

Fig. 5. Schema of the search operation in encrypted M-Index

Query Evaluation. Once the index is constructed, the data owner provides the clients with the private information (set of pivots and key for the symmetric cipher) and any client possessing this *secret key* can perform the search. To execute any search query, authorized client first computes distances to pivots (part of the secret key) and sends the search request to the M-Index on the server side (the search request consists of query object pivot permutation or distances to pivots, the query object itself is not a part of the request). Please, follow the diagram of this procedure in Figure 5. The server determines and sends back the candidate set S_C (set of encrypted objects), the client then decrypts the objects from S_C and finalizes the result (computes the distances of decrypted candidate objects to the query object). In this way, all basic types of search queries mentioned above can be evaluated (precise range and k-NN as well as the approximate k-NN). From the client point of view, the algorithm is very similar for all these types of queries and it is formalized in Algorithm 2.

Depending on the type of the search in Algorithm 2, either the query-pivot distances (line 5) or only the pivot permutations (line 9) are sent to the server. In either case, the candidate set is refined after decrypting the retrieved objects. For the approximate k-NN search, the client can choose the size CandSize of the candidate set S_C (line 10); S_C retrieved from the server is pre-ranked, therefore the client can choose to decrypt and compute distances only for candidates with the highest rank to speed up the search process.

Algorithm 2. Encrypted M-Index search algorithm (client)

```
Input: range $R(q,r)$ or $k$-NN$(q)$, secretKey with $p_1,\ldots,p_n$ and cipher key
Output: Answer set $A$
calculate $d(q, \text{secretKey}.p_i)$, $\forall i \in \{1,\ldots,n\}$;
init new encrypted object $e$ ;  /* e := struct {distances, permutation} */
initialize new encrypted object $e$;
if precise range query is used then
    $e.distances \leftarrow d(q, \text{secretKey}.p_1),\ldots,d(q, \text{secretKey}.p_n)$;
    $S_C \leftarrow$ server.rangeSearch$(e,r)$ ;        /* candidate set of enc. obj. */
else
    sort the distances to find permutation $(1)_q,\ldots,(n)_q$;
    $e.permutation \leftarrow (1)_q,\ldots,(n)_q$;
    $S_C \leftarrow$ server.approxKNN$(e, \text{CandSize})$ /* candidate set of enc. obj. */
$A \leftarrow \emptyset$;
foreach object $c$ in $S_C$ do
    $o \leftarrow$ secretKey.decrypt$(c)$;
    compute distance $d(q,o)$;
    if $o$ satisfies the query constraint then
        $A \leftarrow A \cup \{o\}$;
```

Algorithm 3 formalizes the range search algorithm on the server side. It uses the precomputed query-pivot distances and the query radius r to prune the index tree using several metric-based constraints [6] (line 3 of Alg. 3). Furthermore, because each indexed object contains distances to the pivots, the server can use *pivot filtering* [7, 6] to further reduce the candidate set size (lines 5–7).

Algorithm 4 describes the server-side procedure for the approximate k-NN search. As mentioned in Section 4.1, the candidate set of a given size can be determined only based on query pivot permutation or query-pivot distances: The M-Index can order the Voronoi cells by some "promise value" (line 3). The precise k-NN search can be realized as an approximate k-NN search, that determines distance ρ_k to the k-th nearest neighbor of q, and then subsequent precise range query $R(q,\rho_k)$ is executed [7].

Algorithm 3. Range query algorithm (server.*rangeSearch*)

```
  Input: encrypted query object $q$ with distances $q.distances$, radius $r$
  Output: Candidate set $S_C \subseteq X$
1 $S_C \leftarrow \emptyset$;
2 foreach Voronoi cell $C$ traversing the M-Index do
3     if $C$ cannot be pruned using $q.distances$ and radius $r$ then
4         $S_C \leftarrow S_C \cup C$;
5 foreach object $o$ in $S_C$ do
6     if $\max_{i=1}^{n} |q.distances[i] - o.distances[i]| > r$ then
7         $S_C \leftarrow S_C \setminus \{o\}$;
```

Algorithm 4. Approximate k-NN algorithm (server.*approxKNN*)

```
Input: encrypted $q$ with $q.permutation$, candidate set size CandSize
Output: Pre-ranked candidate set $S_C$
$S_C \leftarrow \emptyset$;
while $|S_C| <$ CandSize do
    $C \leftarrow$ next promising Voronoi cell from the index;
    $S_C \leftarrow S_C \cup C$;
trim $S_C$ to required size CandSize;
```

4.3 Security Analysis

The secret key of authorized clients consist of the set of pivots and key for symmetric cipher used to encrypt the data. The information which is stored on the server (and can possibly leak to an attacker) are the pivot permutations (or object-pivot distances) and the encrypted MS objects. Since the pivots are part of the private key and only authorized clients know them, only they can query the server by "meaningful" queries. An attacker can query the server index using an arbitrarily chosen pivot permutation not knowing to which query object(s) this permutation belongs; the received candidate objects are encrypted and without any similarity distances or other meta-data. If the server is compromised, the attacker could learn the index structure and thus the sets of clustered MS objects (encrypted); nevertheless, not knowing the pivots and the metric function, it would be difficult to learn specifics about the data set. Clearly, this approach belongs to the third privacy level defined in Section 2.3.

In order to even better hide information about the data set distribution, we would like to apply certain distance transformations that would preserve the utmost pruning and filtering efficiency of the server-side index. The aim is to provide the security level described in the forth paragraph of Section 2.3 and this belongs to our future work.

4.4 Implementation

The M-Index is implemented with the aid of MESSIF, a general and robust framework that supports the task of building prototypes of similarity search algorithms [14]. Encrypted M-Index is implemented as an encryption layer in MESSIF. Even though the algorithm was designed for the M-Index as the server algorithm, the architecture of encryption layer is general and robust. Therefore it can be used with any MESSIF-based algorithm.

Core of the encryption layer is an encryption client which can (if supplied with a secret key) provide a communication interface (layer) for a remote server running a similarity search algorithm (indexing structure running in a distributed cloud environment for example). Both client and server are Java processes communicating via TCP/IP.

5 Experimental Evaluation

The main objective of the experimental evaluation is to measure the price paid for ensuring privacy in terms of the efficiency loss – the computation demands placed upon the client due to the privacy protection and the client-server communication costs (volume of exchanged data and communication time). The client computations consist principally of (1) data encryption/decryption and (2) the search algorithm fragments relocated from the server (mainly computation of the distance function). All these cost components are quantified and their significance is compared for different types of data sets (small vs. large, simple distance function vs. complex distance function). In every experiment, the performance of the proposed secure M-Index is compared with the basic (non-encrypted) version of M-Index in order to properly quantify the cost paid for the security.

5.1 Data Sets and Settings

The Encrypted M-Index was tested on three real data sets, whose properties are summarized in Table 1.

YEAST[1] A gene expression data matrix obtained from a Microarray experiment on yeast. Each entry indicates the expression level of a specific gene (row/tuple) at a specific condition (column/attribute) [4].

HUMAN[2] A gene expression data matrix obtained from a Lymphoma/Leukemia Molecular Profiling Project.

CoPhIR[3] Collection of one million images downloaded from Flicker photo site. From each image, five MPEG-7 visual descriptors were extracted and the distance combines them [15]. We test scalability of our solution on this set.

Table 1. Data sets summary

Name	# of records	Data type	Distance function
YEAST	2,882	17-dim. num. vectors	L_1
HUMAN	4,026	96-dim. num. vectors	L_1
CoPhIR	1,000,000	280-dim num. vectors	combination of L_p

We performed all the experimental evaluations on a machine with 8 CPU cores (double quad core) with 8GB RAM, 4 high-speed disks in RAID-5. To reduce an influence of network communication time, both encryption client and M-Index server were running on the same machine communicating via loopback interface. Standard symmetric cipher AES with 128 bit key was used for encryption.

[1] `http://arep.med.harvard.edu/biclustering/`

[2] `http://arep.med.harvard.edu/biclustering/`

[3] `http://cophir.isti.cnr.it/`

Evaluation parameters of the M-Index running on the server side for each data set are summarized in Table 2. The pivots used were chosen at random from within the data set. The experimental evaluation is divided into two phases: the construction phase (data insertion) and the search phase.

Table 2. M-Index parameters

Name	**Bucket capacity**	**Storage type**	**# of pivots**
YEAST	200	Memory storage	30
HUMAN	250	Memory storage	50
CoPhIR	1,000	Disk storage	100

5.2 Index Construction

For the construction phase, we used bulk insert operations to insert the data into M-Index running as the server via encryption client. The size of each bulk was 1,000. Measures taken for each data set are the following (they are used for both construction and search phases):

client time the overall client computation time: data encryption/decryption, distance computations (object-pivot distances), and processing overhead,
server time the time to store objects in the M-Index (and to build the M-Index cell tree) or, in the search phase, to prepare the candidate set,
communication time time spent on communication between server and client,
overall time sum of client, server and communication times.

Results for the construction phase are summarized in Table 3. We can see that the relative importance of the client, server and communication times differs for the three data sets: for the small YEAST and HUMAN data sets, the server time is more than 50 % of the client time and the communication time is very important, and, for CoPhIR, the server and communication times are marginal in comparison with the time spent by client-side computations. This is caused by the fact that the CoPhIR distance function is more demanding than the YEAST and HUMAN and the distances are computed on client. By analogy, the encryption time is relatively more important for YEAST and HUMAN.

Table 3. Index construction of encrypted M-Index

		YEAST	**HUMAN**	**CoPhIR**
Client time [s]		0.208	0.324	1,584.5
	Encryption time [s]	0.117	0.155	32.2
	Dist. comp. time [s]	0.026	0.101	1,541.4
Server time [s]		0.116	0.188	70.2
Communication time [s]		0.182	0.288	52.9
Overall time [s]		0.506	0.800	1,707.7

Let us compare these results with the basic non-encrypted M-Index. All the settings (computational power, network, M-Index parameters, client-server architecture, etc.) were the same, the only difference was the absence of the encryption layer. The index construction results for non-encrypted M-Index are summarized in Table 4. For the small YEAST and HUMAN data sets, we can see that the overall time increase caused by the encryption (Table 3) was about 60 % (e.g. 0.315 s vs 0.506 s, for YEAST). In the case of CoPhIR, the overall time is practically the same for both variants because the encryption time is marginal comparing to the distance computation time.

Table 4. Index construction of the basic (non-encrypted) M-Index

	YEAST	**HUMAN**	**CoPhIR**
Client time [s]	0.001	0.009	0.300
Server time [s]	0.144	0.216	1,563.4
Dist. comp. time [s]	0.026	0.101	1,541.4
Communication time [s]	0.170	0.265	141.4
Overall time [s]	0.315	0.490	1,705.2

5.3 Approximate Search

For the evaluation of the search efficiency, we used the approximate k-NN search algorithm on one hundred query objects randomly chosen from the data set. We varied the parameter k but the results were similar and we present only results for $k = 30$. All presented results are averaged over the 100 queries. For each data set, we varied the size of the candidate set provided by the server (see Section 9) which influences the quality of the result and efficiency. Besides the measures introduced in the previous section, the following values are presented:

decryption time time spent on deserialization and decryption of the candidate objects received from the server,
recall quality of the result (see Section 4.1),
communication cost amount of data sent between the server and client.

The search results for the Encrypted M-Index are summarized in Table 5 (YEAST) and Table 6 (CoPhIR). Results for the HUMAN data set are not presented – the trends do not differ from YEAST (the sizes of the collections are very similar and the character of data and distance function is the same).

For the two small data sets (YEAST and HUMAN) with a simple distance function, server time is approximately 50% of the client time. For the CoPhIR data set, the ratio of server/client time is approximately 1/5 due to resource demanding distance computations on client side. The relative significance of decryption time (which includes also deserialization of the objects) is the same for all the data sets (and candidate sizes) and it is relatively high; this cost component can be hardly moved from the client, provided the system is secure (decryption can be done only by an authorized client having a secret key).

Table 5. Approximate 30-NN evaluation using the Encrypted M-Index (YEAST)

Candidate set size		**150**	**300**	**600**	**1,500**
Client time [s]		0.002	0.003	0.006	0.014
	Decryption time [s]	0.001	0.002	0.004	0.010
	Dist. comp. time [s]	0.001	0.001	0.002	0.003
Server time [s]		0.001	0.002	0.004	0.009
Communication time [s]		0.003	0.003	0.004	0.008
Overall time [s]		0.006	0.008	0.014	0.031
Recall [%]		59.80	82.87	91.3	91.6
Communication cost [**kB**]		25.805	51.643	103.308	258.314

Naturally, the decryption, distance computation and communication time values are linearly proportional to the size of the candidate set S_C, which also influences the recall. For the YEAST data set, we can observe that the $|S_C| = 600$ (about 20 % of the collection size) results in recall over 90 % and further increase of the S_C size does not lead to a significant recall improvement. For CoPhIR, we can achieve almost 90% recall with the S_C size of 50,000 (5 % of the collection).

Table 6. Approximate 30-NN evaluation using the Encrypted M-Index (CoPhIR)

Candidate set size		**500**	**1,000**	**5,000**	**10,000**	**20,000**	**50,000**
Client time [s]		0.034	0.047	0.224	0.446	0.899	2.429
	Decryption time [s]	0.019	0.026	0.132	0.264	0.534	1.451
	Dist. comp. time [s]	0.009	0.018	0.082	0.180	0.356	0.969
Server time [s]		0.005	0.016	0.059	0.110	0.207	0.498
Communication time [s]		0.006	0.008	0.029	0.054	0.106	0.290
Overall time [s]		0.045	0.071	0.312	0.610	1.212	3.217
Recall [%]		7.619	10.952	36.667	55.238	71.905	87.143
Communication cost [**kB**]		460	921	4,605	9,211	18,423	46,058

For comparison, we present the results of the search procedure on the basic (non-encrypted) M-Index – see Table 7 (YEAST) and Table 8 (CoPhIR). In this setting, practically all the work is done on the server side. The most significant difference between encrypted and non-encrypted M-Index is in the communication cost and time, because the non-encrypted variant directly returns the answer set with 30 objects, not the candidate set (which can have up to 50,000 objects in case of CoPhIR). The fixed set of 30 objects transferred between sever and client implies same communication time for all S_C sizes. The amount of work on the client is negligible, therefore there are no values in the search result tables.

To summarize this section, the price paid for privacy in the search phase on Encrypted M-Index is mainly the communication cost (which grows linearly with the candidate set size) and the time of decrypting the candidate objects. This results in approximately three times longer overall search time measured on the encrypted variant. The distance computations, performed during the necessary candidate set refinement on the client, constitute between 1/10 and 1/3 of the overall search time, depending on the complexity of the distance function.

Table 7. Approx. 30-NN evaluation using basic (non-encrypted) M-Index (YEAST)

Candidate set size	**150**	**300**	**600**	**1,500**
Client time [s]	–	–	–	–
Server time [s]	0.001	0.001	0.002	0.003
Dist. comp. time [s]	0.001	0.001	0.002	0.003
Communication time [s]	0.002	0.002	0.002	0.002
Overall time [s]	0.003	0.003	0.004	0.005
Communication cost [kB]	5.161	5.164	5.162	5.161

Table 8. Approx. 30-NN evaluation using basic (non-encrypted) M-Index (CoPhIR)

Candidate set size	**500**	**1,000**	**5,000**	**10,000**	**20,000**	**50,000**
Client time [s]	–	–	–	–	–	–
Server time [s]	0.023	0.032	0.116	0.215	0.416	1.029
Dist. comp. time [s]	0.009	0.018	0.082	0.180	0.356	0.969
Communication time [s]	0.006	0.006	0.005	0.005	0.005	0.006
Overall time [s]	0.029	0.038	0.121	0.220	0.421	1.023
Communication cost [kB]	26.421	26.174	26.403	26.325	26.196	26.254

5.4 Comparison with Other Solutions

In order to compare our solution with the referenced approaches, we evaluated 1-NN queries on the YEAST data set, which corresponds with the setting of Yiu et al. [4]. To be more specific, we adjusted the approximate 1-NN strategy so that the server-side M-Index was limited to access only one M-Index Voronoi cell which then forms the candidate set (this leads to candidate sets of average size 42). Again, we ran these 1-NN queries for 100 randomly chosen query objects (they were excluded from the indexed set). In Table 9, we report average values of the collected measures. The recall value says how many queries (out of 100) resulted in the actual nearest neighbor.

In comparison with results presented by Yiu et al. [4], the Encpryted M-Index outperformed all the techniques in the communication cost. It also outperformed FDH [4] (technique which uses also approximate search) in CPU time. On the other hand, it takes more time to construct the index (the Encrypted M-Index was approximately twice slower than FDH).

However, it is always difficult to compare the wall-clock CPU times, as they strongly depend on specific implementation and hardware. Moreover, the referenced paper does not specify technical details about the symmetric cipher used (and its key length), the computational power and network setting. Also, the referenced approaches [4] modify the distance function for indexing, therefore they belong to the fourth privacy level whereas our approach fulfills conditions of the third level (see Section 2.3).

Table 9. Approximate 1-NN search evaluation results for the YEAST data set

Client time [ms]		0.509
	Decryption time [ms]	0.160
	Dist. comp. time [ms]	0.210
Server time [ms]		1.001
Communication time [ms]		1.180
Overall time [ms]		2.690
Recall [%]		94.0
Communication cost [kB]		2.368

6 Conclusions and Future Work

We proposed a method that can be used to ensure data privacy in similarity search systems outsourced in a cloud. The proposed solution exploits existing efficient metric indexes based on a fixed set of pivots. This set is part of the secret key controlled by authorized clients, while the server itself cannot compute the similarity distance function, nor it can access any data in an unencrypted form. A potential attacker can only learn encrypted object data and pivot permutations.

Our approach has been implemented as a real client-server “similarity cloud” system exploiting an existing mature implementation of the M-Index (disk-efficient, parallel, potentially distributed). The performance of the system was experimentally evaluated on several real data sets focusing on individual components of the search time (server, communication, data decryption, client data operations). The Encrypted M-Index has the intention of providing required privacy while preserving the server-side efficiency as much as possible and relocate only the necessary computations to the client (data decryption and computation of query-data distances).

In the future, we would like to analyze the precise range and k-NN evaluation strategies of Encrypted M-Index in comparison to the approximate strategy and to other possible solutions. Further, we would like to study various types of distance transformations (i.e. transform the distances to pivots stored on the server for precise strategies); such transformation could better hide information about the data set distribution and thus further restrict possible attacks.

Acknowledgments. This work was supported by national Czech Research Foundation project GACR P103/12/G084.

References

1. Park, H.A., Kim, B.H., Lee, D.H., Chung, Y.D., Zhan, J.: Secure similarity search. In: 2007 IEEE International Conference on Granular Computing (GRC 2007), pp. 598–598. IEEE (2007)
2. Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W.: Fuzzy keyword search over encrypted data in cloud computing. In: Proceeding of the 29th Conference on Information Communications, pp. 441–445 (2010)
3. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. In: 2011 Proceedings IEEE INFOCOM, pp. 829–837. IEEE (2011)
4. Yiu, M.L., Assent, I., Jensen, C.S., Kalnis, P.: Outsourced Similarity Search on Metric Data Assets. IEEE Transactions on Knowledge and Data Engineering 24(2), 338–352 (2012)
5. Novak, D., Batko, M.: Metric index: an efficient and scalable solution for similarity search. In: Second International Workshop on Similarity Search and Applications (SISAP 2009), pp. 65–73. IEEE (2009)
6. Novak, D., Batko, M., Zezula, P.: Metric Index: An Efficient and Scalable Solution for Precise and Approximate Similarity Search. Information Systems 36(4), 721–733 (2011)
7. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. In: Advanced Database Systems, vol. 32. Springer (2006)
8. Hore, B., Mehrotra, S., Canim, M., Kantarcioglu, M.: Secure multidimensional range queries over outsourced data. The VLDB Journal 21(3), 333–358 (2011)
9. Chávez, E., Figueroa, K., Navarro, G.: Effective Proximity Retrieval by Ordering Permutations. IEEE Transactions on Pattern Analalysis and Machine Intelligence 30(9), 1647–1658 (2008)
10. Amato, G., Savino, P.: Approximate similarity search in metric spaces using inverted files. In: Proceedings of the 3rd International Conference on Scalable Information Systems (2008)
11. Esuli, A.: PP-Index: Using permutation prefixes for efficient and scalable approximate similarity search. In: Proceedings of LSDS-IR 2009 (2009)
12. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (2001)
13. Skala, M.: Counting distance permutations. Journal of Discrete Algorithms 7(1), 49–61 (2009)
14. Batko, M., Novak, D., Zezula, P.: MESSIF: Metric similarity search implementation framework. Digital Libraries Research and Development 4877(102), 1–10 (2007)
15. Bolettieri, P., Esuli, A., Falchi, F., Lucchese, C., Perego, R., Piccioli, T., Rabitti, F.: CoPhIR: A Test Collection for Content-Based Image Retrieval. CoRR abs/0905.4 (2009)