A Locality Sensitive Hashing Filter for Encrypted Vector Databases

Kawamoto, Junpei
Graduate School of Systems and Information Engineering, University of Tsukuba

出版情報：Fundamenta Informaticae. 137 (2), pp.291-304, 2015-03-26. IOS Press
バージョン：
権利関係：

Fundamenta Informaticae 130 (2013) 1–15
DOI 10.3233/FI-2014-900
IOS Press

A Locality Sensitive Hashing Filter for Encrypted Vector Databases

Junpei Kawamoto[*†]
Graduate School of Systems and Information Engineering
University of Tsukuba
1-1-1 Tennodai, Tsukuba, Japan
junpei@mdl.cs.tsukuba.ac.jp

Abstract. We introduce a filtering methodology based on locality-sensitive hashing (LSH) and whitening transformation to reduce candidate tuples between which encrypted vector databases (EVDBs) must compute similarity for query processing. The LSH hashing methodology is efficient for estimating similarities between two vectors. It hashes a vector space using randomly chosen vectors. We can filter vectors that are less similar to the querying vectors by recording which hashed space each vector belongs to. However, if vectors in EVDBs are found locally, then most vectors are in the same hashed space, so the filter will not work. Because we can treat those cases using whitening transformation to distribute the vectors broadly, our proposed filtering methodology will work effectively on any vector space. We also show that our filter reduces the server's query processing cost.

Keywords: Query processing, Encrypted databases, Security and privacy

1. Introduction

Encrypted vector database (EVDB) is a secure version of *vector database (VDB)*. VDB is a database that manages tuples $(\boldsymbol{k}, v)$ consisting of a key vector $\boldsymbol{k}$ and an associated value v, and which is designed to find values corresponding to key vectors that are similar to a given querying vector. For example,

[*]This work is partly supported by the Nakajima Foundation.
[†]Address for correspondence: Graduate School of Systems and Information Engineering, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Japan

Received ?? ??; revised ?? 2013

picture databases and literature databases might be VDBs, with the picture database containing vectors of features for each picture and the literature database including vectors containing features of the texts. When database owners wish to deploy such VDBs on cloud services and share their data with colleagues for collaboration, encryption methods for VDBs are required. The database owner will expect cloud services to reduce their database management costs and to provide collaborative environments. However, we can readily imagine a situation in which the owner cannot perfectly trust those services and wants to keep their data secret from the cloud services. Their colleagues, i.e. users of their databases, should keep queries private as well. EVDB, which encrypts all tuples before establishing VDBs on cloud services, and which also encrypts all queries, is effective in this case. The encryption algorithms used by EVDB have the property that similarities between key vectors and query vectors will remain similar after encryption, so that cloud services can find encrypted key vectors that are similar to encrypted queries in the same way as if the key vectors and the queries were not encrypted.

All key vectors in EVDBs are encrypted such that we cannot assume that structures of plain vectors are retained in encrypted vectors. This encryption makes it difficult to construct search indexes for EVDBs. Particularly structure-based indexing methods such as R-tree [16] are apparently efficient. Existing EVDBs actually must compute similarities between a query vector and *all* key vectors in the EVDBs. Because queries are also encrypted and because even identical queries will be encrypted to different vectors, EVDBs cannot cache the results of previous queries. Combining these facts, query processing of EVDBs entails high computational costs.

In this paper, we introduce a filtering methodology based on *locality-sensitive hashing (LSH)* [4] and *whitening transformation* to reduce the number of candidate tuples between which the EVDB must compute similarity at the time of query processing. The LSH hashing methodology is efficient for filtering vectors based on their similarities. It is used for several purposes [10, 8]. LSH hashes a vector space using randomly chosen vectors that we designate as *base vectors*. We can filter vectors that are less similar to query vectors by recording which hashed space each vector belongs to. However, LSH assumes a uniform distribution of the distance between vectors. Therefore, if key vectors in EVDBs are found locally, meaning most vectors are in the same hashed space, our filter will not distinguish vectors. To treat these cases, we use the whitening transformation to make certain that key vectors are widely distributed.

The remainder of the paper is organized as follows. After we introduce related work in section 2, we define basic notions and formulate related methodologies with our notions in section 3. We then present our filtering methodology in section 4 and evaluate efficiency of proposed filter Section 5. Finally we conclude this paper with section 6.

2. Related work

Some studies have investigated encrypted vector databases [13, 14, 18] that support some operations over the encrypted vectors. For example, we introduce k-nearest neighbour ($k$nn) search over EVDBs. $k$nn search finds top k tuples having similar key vectors $\boldsymbol{k}$ from EVDBs to a given query vector $\boldsymbol{q}$. SCONEDB [18] is an EVDB supporting this $k$nn search. We introduce details in section 3.1. Another type of EVDB achieves some queries over relational databases with transformation of the relational databases into EVDBs. IPP method [9] is a methodology supporting range queries over encrypted databases, which

consists of a natural number key attribute. This method encrypts scalar key attribute values into vectors so that it is a kind of EVDB. We also introduce details of this methodology in section 3.2.

Many studies have examined query processing over encrypted database (EDB) [6, 17, 15]. An encrypted database is a secure version of a relational database. Similar to EVDB scenario, tuples in plain the relational database are encrypted before sending them to cloud services. A main research topic is how to find items matching search intentions over encryption efficiently so that many indexing methodologies are studied. Bucketization [6, 7] splits the database of each attribute into some labelled buckets. Those labels are used as queries. Although these approaches can work with tree-based indexing methodologies, they are not safe for frequency based attacks [9] because their encryption algorithms map same queries to same encrypted queries. Agrawal et al. introduce a methodology hiding frequencies of queries [1] based on an order-preserving encryption [2, 3]. By the order preserving encryption scheme reported by Agrawal et al., encrypted items retain the same order as plain items. For that reason, they are comparable after encryption. Their schema can also modify the distribution of items: for example, we might have plain items from a zipf distribution, but with encrypted items distributed uniformly. However, they emphasize not only the distribution of items but also the frequencies of queries. Therefore, adversaries who know the frequencies of queries are able to match encrypted queries to plain ones even if encrypted items have uniform distributions. Lu also introduces a query processing methodology over encrypted items using order-preserving encryption and producing tree structures [12]. It achieves $O(\log n)$ computational cost of servers, but it is also pointed out in the paper that the proposed methodology is not secure against frequency-based attacks. Therefore those results of studies suggest that tree-based indexing methodologies bring weakness even if we can apply them to our EVDB.

3. Basic notions

We denote a VDB that a database owner maintains as $VDB(Key, Value)$, where Key is a key attribute consisting of vectors and $Value$ is an attribute consisting of data associated with each key vector. For example, Key is a set of feature vectors of pictures or texts and $Value$ is a set of actual pictures or texts in a previous picture database or literature database. We also assume that database users send queries only over Key, and that each query means to find tuples consisting of key vectors of which similarities between the query vector are greater than or equal to a threshold α. In other words, all requests from users are to find tuples of which key vectors $\boldsymbol{k}$ satisfy $\mathrm{sim}(\boldsymbol{k}, \boldsymbol{q}) \geq \alpha$, where $\boldsymbol{q}$ is a query vector. We denote the domain of the query vector as $Query$ i.e. $\boldsymbol{q} \in Query$. For simplicity, we employ cosine similarity as the measure of similarity in this paper. Then we also assume that each vector is normalized.

We denote an EVDB associated with a VDB as $EVDB(Key_e, Value_e)$, where Key_e is an attribute of encrypted key vectors and $Value_e$ is an attribute of encrypted values. Each encrypted key vector $\boldsymbol{k}_e \in Key_e$ associated with a key vector $\boldsymbol{k} \in Key$ is computed by $\boldsymbol{k}_e = \mathrm{Enc}_k(\boldsymbol{k})$, where Enc_k is an encryption algorithm for key vectors. Each encrypted value $v_e \in Value_e$ associated with a value $v \in Value$ is also computed by $v_e = \mathrm{Enc}_v(v)$, where Enc_v is an encryption algorithm for values. This EVDB is the database deployed to cloud services, which we will refer to simply as database servers. The plain VDB is kept secret by the database owner. Queries from users are also encrypted.

When a user wants to send a query vector $\boldsymbol{q}$ and a threshold α as a query, an encrypted query vector $\boldsymbol{q}_e = \mathrm{Enc}_q(\boldsymbol{q})$ and the threshold α are sent to a database server on a cloud service instead, where Enc_q is an encryption algorithm for query vectors. We denote the domain of encrypted key vectors as $Query_e$

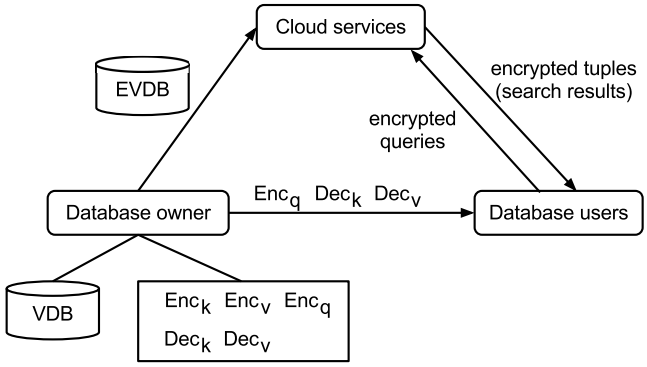


Figure 1. Stakeholders of EVDB scenario.

Table 1. Summary of important variables and algorithms.

symbol	description
Key	Domain of key attribute and $Key \subseteq \mathbb{R}^{d\dagger}$
$Query$	Domain of queries and $Query \subseteq \mathbb{R}^{d}$.
$\boldsymbol{k}$	Vector of the key attribute in VDB, i.e. $\boldsymbol{k} \in Key$.
$\boldsymbol{q}$	Query vector, i.e. $\boldsymbol{q} \in Query$.
Key_e	Domain of encrypted key attribute and $Key_e \subseteq \mathbb{R}^{d'\ddagger}$
$Query_e$	Domain of encrypted queries and $Query_e \subseteq \mathbb{R}^{d'}$.
$\boldsymbol{k}_e$	Encrypted vector associated with a plain key attribute $\boldsymbol{k}$, i.e. $\boldsymbol{k}_e \in Key_e$.
$\boldsymbol{q}_e$	Encrypted vector associate with a plain query vector $\boldsymbol{q} \in \mathbb{R}^d$.
Enc_k	Encryption algorithm for a key attribute defined as $\mathrm{Enc}_k : Key \to Key_e$.
Enc_q	Encryption algorithm for queries defined as $\mathrm{Enc}_q : Query \to Query_e$.

† d is the dimension of key attribute.
‡ d' is the dimension of encrypted key attribute.

so that $\boldsymbol{q}_e \in Query_e$. The database server searches a set of tuples $(\boldsymbol{k}_e, v_e)$ satisfying $\mathrm{sim}(\boldsymbol{k}_e, \boldsymbol{q}_e) \geq \alpha$ and returns them to the user. In this process, the database server cannot compute plain $\boldsymbol{k}, \boldsymbol{q}$, and v from $\boldsymbol{k}_e$, $\boldsymbol{q}_e$, and v_e. Finally, the user receives a set of $(\boldsymbol{k}_e, v_e)$ and obtains plain values by decrypting them $\boldsymbol{k} = \mathrm{Dec}_k(\boldsymbol{k}_e)$ and $v = \mathrm{Dec}_v(v_e)$, where Dec_k and Dec_v respectively denote decryption algorithms for key vectors and values. To run this protocol, the database owner knows Enc_k, Enc_q, Enc_v, Dec_k, and Dec_v, and notifies users of Enc_q, Dec_k, and Dec_v. Moreover, they know only this information, and database servers do not know the encryption and decryption algorithms above. Fig. 1 shows the stakeholders of this scenario. The database owner has the plain VDB and all encryption and decryption algorithms. The owner deploys the EVDB to cloud services (database servers) and notifies database users of encryption and decryption algorithms.

We argue efficient querying process in this paper so that we are interested in key vectors, query vectors, and their encryption algorithms. Table 1 presents a summary of those important variables and algo-

rithms. We examine how such variables and algorithms are defined in two related studies, SCONEDB [18] and IPP method [9], in the remainder of this section.

3.1. SCONEDB

In SCONEDB [18], the encryption algorithm Enc_k for key vectors $\boldsymbol{k}$. The encryption algorithm Enc_q for query vectors $\boldsymbol{q}$ are defined as

$$\mathrm{Enc}_k(\boldsymbol{k}) = \boldsymbol{M}^T(\boldsymbol{k}^T, -0.5\|\boldsymbol{k}\|^2)^T, \quad \mathrm{Enc}_q(\boldsymbol{q}) = r\boldsymbol{M}^{-1}(\boldsymbol{q}^T, 1)^T,$$

where $\boldsymbol{M}$ is a random regular matrix and works as a shard encryption key, and r is a positive random number. We denote the transposed matrix of a matrix $\boldsymbol{M}$ as $\boldsymbol{M}^T$. Those encryption schemes which SCONEDB proposes enable us to compare which vector between any two vectors $\boldsymbol{k}_1$ and $\boldsymbol{k}_2$ is more similar to a given query vector $\boldsymbol{q}$ after encrypting them. In other words, it is possible to compare $\mathrm{sim}(\boldsymbol{k}_1, \boldsymbol{q})$ and $\mathrm{sim}(\boldsymbol{k}_2, \boldsymbol{q})$ by comparing $\mathrm{sim}(\mathrm{Enc}_k(\boldsymbol{k}_1), \mathrm{Enc}_q(\boldsymbol{q}))$ and $\mathrm{sim}(\mathrm{Enc}_k(\boldsymbol{k}_2), \mathrm{Enc}_q(\boldsymbol{q}))$. Moreover, SCONEDB's encryption scheme does not enable us to compute any other operation after encryption to provide security. We can compare the similarity above between two vectors $\boldsymbol{k}_1$ and $\boldsymbol{k}_2$ as checking the following condition:

$$(\mathrm{Enc}_k(\boldsymbol{k}_1) - \mathrm{Enc}_k(\boldsymbol{k}_2)) \cdot \mathrm{Enc}_q(\boldsymbol{q}), \tag{1}$$

where operator $\cdot$ denotes the inner product. The first subtraction of two encrypted vectors is computed as

$$\begin{aligned} \mathrm{Enc}_k(\boldsymbol{k}_1) - \mathrm{Enc}_k(\boldsymbol{k}_2) &= \boldsymbol{M}^T(\boldsymbol{k}_1^T, -0.5\|\boldsymbol{k}_1\|^2)^T - \boldsymbol{M}^T(\boldsymbol{k}_2^T, -0.5\|\boldsymbol{k}_2\|^2)^T \\ &= \boldsymbol{M}^T\left((\boldsymbol{k}_1 - \boldsymbol{k}_2)^T, -0.5(\|\boldsymbol{k}_1\|^2 - \|\boldsymbol{k}_2\|^2)\right)^T, \end{aligned}$$

so that the condition (1) is evaluated as

$$\begin{aligned} &(\mathrm{Enc}_k(\boldsymbol{k}_1) - \mathrm{Enc}_k(\boldsymbol{k}_2)) \cdot \mathrm{Enc}_q(\boldsymbol{q}) \\ &= r\left((\boldsymbol{k}_1 - \boldsymbol{k}_2)^T, -0.5(\|\boldsymbol{k}_1\|^2 - \|\boldsymbol{k}_2\|^2)\right) MM^{-1}(\boldsymbol{q}^T, 1)^T \\ &= r\left((\boldsymbol{k}_1 - \boldsymbol{k}_2) \cdot \boldsymbol{q} - 0.5(\|\boldsymbol{k}_1\|^2 - \|\boldsymbol{k}_2\|^2\right) \\ &= 0.5r\left(\|\boldsymbol{k}_2 - \boldsymbol{q}\|^2 - \|\boldsymbol{k}_1 - \boldsymbol{q}\|^2\right). \end{aligned}$$

Because r is positive, checking a sign of the condition (1) is equal to checking which is greater between $\|\boldsymbol{k}_1 - \boldsymbol{q}\|^2$ and $\|\boldsymbol{k}_2 - \boldsymbol{q}\|^2$. We store vectors $\boldsymbol{k}_{ij} = \boldsymbol{k}_i - \boldsymbol{k}_j$ in EVDBs for any pair of vectors $\boldsymbol{k}_i$, $\boldsymbol{k}_j$. In query processes of those EVDBs, servers search vectors $\boldsymbol{k}_{ij}$ satisfying $\mathrm{sim}(\boldsymbol{k}_{ij}, \boldsymbol{q}) > 0$ for given query vectors $\boldsymbol{q}$, and then decide top k similar vectors for the given query vector $\boldsymbol{q}$.

3.2. IPP method

As explained in section 2, IPP method [9] encrypts scalar key attribute values into vectors. Let $DB(Key, Value)$ be a plain database and we encrypt them, where the key attribute $Key \subset \mathbb{N}$. Algorithm Enc_k which encrypts key attribute values $k \in Key$ to encrypted key vectors $\boldsymbol{k}_e$ is defined as

$$\mathrm{Enc}_k(k) = r_1\boldsymbol{M}^T\left((k + r_2)^3, (k + r_2)^2, k + r_2, 1\right)^T,$$

where r_1 and r_2 are positive random values, and where $\boldsymbol{M}$ is a regular matrix shared between a database owner and database users, i.e. a shared key. A range query that searches key attribute values k satisfying $a \leq k \leq b$ is described as an encrypted query vector by an algorithm Enc_q defined as

$$\mathrm{Enc}_q(a,b) = r_3 \boldsymbol{M}^{-1} \begin{pmatrix} 1 \\ -(a+b-r_4) \\ (a-\frac{1}{2})(b+\frac{1}{2}) - (a+b)r_4 \\ (a-\frac{1}{2})(b+\frac{1}{2})r_4 \end{pmatrix},$$

where r_3 is a positive random value, r_4 is also a random value satisfying $|r_4| < 1/2$, and M is the shared key used in the algorithm Enc_k. Query processing in the IPP method is represented by finding tuples having encrypted key vectors $\boldsymbol{k}_e$ satisfying $\mathrm{sim}(\boldsymbol{k}_e, \boldsymbol{q}_e) \geq 0$ for a given encrypted query vector $\boldsymbol{q}_e$. IPP method employs the inner product as the measurement of similarity so that $\mathrm{sim}(\boldsymbol{k}_e, \boldsymbol{q}_e) \geq 0$ is equal to $\boldsymbol{k}_e \cdot \boldsymbol{q}_e \geq 0$. This condition is evaluated as

$$\begin{aligned} \boldsymbol{k}_e \cdot \boldsymbol{q}_e &= r_1 r_3 \left((k+r_2)^3, (k+r_2)^2, k+r_2, 1\right) \boldsymbol{M} \boldsymbol{M}^{-1} \begin{pmatrix} 1 \\ -(a+b-r_4) \\ (a-\frac{1}{2})(b+\frac{1}{2}) - (a+b)r_4 \\ (a-\frac{1}{2})(b+\frac{1}{2})r_4 \end{pmatrix} \\ &= r_1 r_3 (k-a+r_4+\frac{1}{2})(k-b+r_4-\frac{1}{2})(k+r_2+r_4). \end{aligned}$$

Because we assumed $|r_4| < 1/2$, key attribute values k satisfying $\boldsymbol{k}_e \cdot \boldsymbol{q}_e \geq 0$ also satisfy $a \leq k \leq b$. In IPP method, encrypted query vectors contain random values so that it has tolerability against attacks using query distributions, which other existing methodologies [6, 1] do not have. As we described in section 1, we cannot assume that structures of plain vectors are retained in encrypted vectors because of those encryption schemes. Therefore, it is difficult to construct search indexes for EVDBs such as [16, 19].

4. LSH filter

We introduce a filtering methodology for querying similar vectors in encrypted vector databases (EVDB). Vectors in EVDB are encrypted and have a different structure from that of the plain vectors. In many protocols of EVDB, as we introduce into section 2, cosine similarities between encrypted key vectors and encrypted query vectors are preserved from before encryption, so that queries can be handled correctly. However, other operations such as cosine similarity calculations between two encrypted key vectors or between two encrypted query vectors will no longer have the same values as those before encryption. We specifically examine this desirable feature, that cosine similarities between key vectors and query vectors should not change in encryption, to construct our filter. LSH is a suitable data structure for this purpose.

In this section, we introduce two basic components of our filter, LSH and whitening transformation. Subsequently, we present our filter and its application to an EVDB.

4.1. Locality sensitive hashing

The locality-sensitive hashing (LSH) methodology has several variations [5, 4, 11]. We use Charikar's version [4] because it is suitable for computing cosine similarities. The LSH which we use consists of m hash functions. Each hash function h_i is made from a base vector $\boldsymbol{b}_i$ and is defined as

$$\mathrm{h}_i(\boldsymbol{v}) = \begin{cases} 1; & \boldsymbol{v} \cdot \boldsymbol{b}_i \geq 0, \\ 0; & \text{otherwise}, \end{cases}$$

where $\boldsymbol{v}$ and $\boldsymbol{b}_i$ must have the same dimensions. Using these m hash functions, the LSH value $\mathrm{lsh}(\boldsymbol{v})$ of a vector $\boldsymbol{v}$ is defined as a tuple that consists of m hash values, i.e.

$$\mathrm{lsh}(\boldsymbol{v}) = (\mathrm{h}_1(\boldsymbol{v}), \mathrm{h}_2(\boldsymbol{v}), \cdots, \mathrm{h}_m(\boldsymbol{v}))\,. \tag{2}$$

LSH values of two vectors $\boldsymbol{u}$, $\boldsymbol{v}$ have the following property under the assumption that vectors are distributed uniformly:

$$\Pr[\mathrm{lsh}(\boldsymbol{u}) = \mathrm{lsh}(\boldsymbol{v})] \approx 1 - \frac{\theta(\boldsymbol{u}, \boldsymbol{v})}{\pi},$$

where $\Pr[\mathrm{lsh}(\boldsymbol{u}) = \mathrm{lsh}(\boldsymbol{v})]$ means how many hash values of the two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ have the same values i.e. $\mathrm{h}_i(\boldsymbol{u}) = \mathrm{h}_i(\boldsymbol{v})$, and $\theta(\boldsymbol{u}, \boldsymbol{v})$ is the angle between the two vectors. This approximate equation suggests that we can estimate the angle between two vectors by computing their LSHs, and the cosine similarity of two vectors $\cos(\boldsymbol{u}, \boldsymbol{v})$ is approximated as

$$\cos(\boldsymbol{u}, \boldsymbol{v}) \approx \cos\left(\pi(1 - \Pr[\mathrm{lsh}(\boldsymbol{u}) = \mathrm{lsh}(\boldsymbol{v})])\right). \tag{3}$$

Because of the assumption of LSH, the accuracy of the approximation (3) depends on the number of base vectors m and the relations between the base vectors $\boldsymbol{b}_i$ and hashed vectors $\boldsymbol{v}$. Therefore, because we wish to apply LSH in our filtering methodology, we consider that the accuracy of the query results under LSH depends on m and on the distribution of encrypted key vectors and encrypted query vectors. In short, LSH splits a vector space into 2^m subspaces by m base vectors. Those base vectors are usually chosen randomly. Therefore, we also choose them randomly and do not consider the distribution of encrypted key vectors and encrypted query vectors. In practice, estimating the distribution of encrypted query vectors is unrealistic. However, if the distribution of encrypted vectors is lopsided, then LSH cannot split the vector space efficiently. In other words, many vectors will lie in the same subspaces. Therefore, we must reduce the skew of the vector space. We employ whitening transformation for this purpose.

4.2. Whitening transformation

We need to hash a vector space of randomly chosen base vectors efficiently to find similar vectors using LSH. As discussed above, if vectors exist in the same part of the vector space, then the hashed vector spaces cannot screen vectors. In the worst case, if all vectors lie in the same hashed space, our filter would be nonsensical. To combat such cases, we use whitening transformation, which is a decorrelation technique.

To construct a whitening matrix, we first compute the covariance matrix $\boldsymbol{\Sigma}$ of all vectors $\mathbf{v}$:

$$\boldsymbol{\Sigma} = E\left((\boldsymbol{v} - \boldsymbol{\mu})(\boldsymbol{v} - \boldsymbol{\mu})^T\right), \tag{4}$$

where $\boldsymbol{\mu}$ is the mean vector of $\boldsymbol{v}$ and $E(\boldsymbol{X})$ denotes the expected value of a matrix $\boldsymbol{X}$. We can compute the eigenvalue decomposition of $\boldsymbol{\Sigma}$ as $\boldsymbol{\Sigma} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^{-1}$, where $\boldsymbol{\Phi}$ is a square matrix whose i-th column is the eigenvector of $\boldsymbol{\Sigma}$ and $\boldsymbol{\Lambda}$ is a diagonal matrix of which the diagonal elements are the corresponding eigenvalues. We define our whitening matrix $\boldsymbol{W}_k$ as

$$\boldsymbol{W}_k = \boldsymbol{\Phi}\boldsymbol{\Lambda}^{-1/2}. \tag{5}$$

For any vector $\boldsymbol{v}$, the whitened vector $\boldsymbol{v}_w$ is computed by

$$\boldsymbol{v}_w = \boldsymbol{W}_k^T(\boldsymbol{v} - \boldsymbol{\mu}). \tag{6}$$

In this case, the covariance of the transformed vectors is

$$E(\boldsymbol{v}_w\boldsymbol{v}_w^T) = E(\boldsymbol{W}_k^T(\boldsymbol{v} - \boldsymbol{\mu})(\boldsymbol{v} - \boldsymbol{\mu})^T\boldsymbol{W}_k) = E(\boldsymbol{\Lambda}^{-1/2}\boldsymbol{\Phi}^T\boldsymbol{\Sigma}\boldsymbol{\Phi}\boldsymbol{\Lambda}^{-1/2}) = \boldsymbol{I},$$

where $\boldsymbol{I}$ represents an identity matrix. This equation means that the transformed vectors are no longer correlated.

Example 4.1. It is possible to imagine three vectors $\boldsymbol{a} = (2, 0)^T$, $\boldsymbol{b} = (1, 1)^T$, and $\boldsymbol{c} = (1, 2)^T$. Their covariance matrix is calculated as

$$\boldsymbol{\Sigma} = \begin{pmatrix} E\left((v_1 - \mu_1)^2\right) & E\left((v_1 - \mu_1)(v_2 - \mu_2)\right) \\ E\left((v_2 - \mu_2)(v_1 - \mu_1)\right) & E\left((v_2 - \mu_2)^2\right) \end{pmatrix} = \begin{pmatrix} \frac{1}{3} & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix},$$

where v_1 and v_2 respectively denote the first and second dimension values of $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$, and μ_1 and mu_2 denote the first and second dimension values of the average vector $\boldsymbol{\mu} = (2, 1/3)^T$ of these three vectors. The covariance matrix $\boldsymbol{\Sigma}$ suggests that those vectors have correlations. If they were to have no correlation, then the covariance matrix would be an identity matrix.

We can decompose the covariance matrix $\boldsymbol{\Sigma}$ as $\boldsymbol{\Sigma} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^{-1}$, where

$$\boldsymbol{\Phi} = \begin{pmatrix} -0.8816746 & 0.47185793 \\ -0.47185793 & -0.8816746 \end{pmatrix}, \quad \boldsymbol{\Lambda} = \begin{pmatrix} 0.06574145 & 0 \\ 0 & 1.26759188 \end{pmatrix},$$

and we compute the whitening matrix $\boldsymbol{W}_k$ for those three vectors:

$$\boldsymbol{W}_k = \boldsymbol{\Phi}\boldsymbol{\Lambda}^{-1/2} = \begin{pmatrix} -3.43865556 & 0.41910373 \\ -1.84031261 & -0.78310249 \end{pmatrix}.$$

Finally, we obtain the whitened vectors as shown below (6):

$$\begin{aligned} \boldsymbol{a}_w &= \boldsymbol{W}_k^T(\boldsymbol{a} - \boldsymbol{\mu}) = (-6.87731112, 0.83820747)^T, \\ \boldsymbol{b}_w &= \boldsymbol{W}_k^T(\boldsymbol{b} - \boldsymbol{\mu}) = (-5.27896817, -0.36399876)^T, \\ \boldsymbol{c}_w &= \boldsymbol{W}_k^T(\boldsymbol{c} - \boldsymbol{\mu}) = (-7.11928077, -1.14710125)^T, \end{aligned}$$

and their covariance matrix is calculated as

$$\boldsymbol{\Sigma}_w = \begin{pmatrix} E\left((v_{w,1} - \mu_{w,1})^2\right) & E\left((v_{w,1} - \mu_{w,1})(v_{w,2} - \mu_{w,2})\right) \\ E\left((v_{w,2} - \mu_{w,2})(v_{w,1} - \mu_{w,1})\right) & E\left((v_{w,2} - \mu_{w,2})^2\right) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

Table 2. Differences between original protocols and the proposed protocol.

(a) encrypted vector of $\boldsymbol{k}$

original protocol	proposed protocol
$\mathrm{Enc}_k(\boldsymbol{k})$	$\boldsymbol{W}_k^T\,(\mathrm{Enc}_k(\boldsymbol{k}) - \boldsymbol{\mu})$

(b) query condition of $\boldsymbol{q}$

original protocol	proposed protocol
find $\boldsymbol{k}_e$ s.t. $\boldsymbol{k}_e \cdot \mathrm{Enc}_q(\boldsymbol{q}) \geq \alpha$	find $\boldsymbol{k}_e$ s.t. $\boldsymbol{k}_e \cdot \boldsymbol{W}_k^{-1}\mathrm{Enc}_q(\boldsymbol{q}) \geq \alpha - \boldsymbol{\mu} \cdot \mathrm{Enc}_q(\boldsymbol{q})$

where $v_{w,1}$ and $v_{w,2}$ respectively denote the first and second dimension values of $\boldsymbol{a}_w$, $\boldsymbol{b}_w$, and $\boldsymbol{c}_w$, and where $\mu_{w,1}$ and $\mu_{w,2}$ denote the first and second dimension values of the average vector of these three vectors. This means the whitened vectors have no correlation; they are what we want.

We expect that this whitening approach will efficiently divide the vector space of encrypted key vectors. We will emphasize the effectiveness of this whitening in section 5.

4.3. Apply to EVDB

Existing EVDB protocols propose encryption and decryption algorithms; Enc_k, Enc_q, Enc_v, Dec_k and Dec_v are introduced in section 3. We use those algorithms for archiving our proposed methodology and extending Enc_k, Enc_q, and Dec_k, so that our proposal can function with those existing protocols. We respectively denote the extended algorithms Enc_k^*, Enc_q^*, and Dec_k^*. In this section, we introduce them first and then our filter.

Let us imagine that a database owner who maintains a $VDB(Key, Value)$ will deploy the database into cloud services with our methodology. The owner first encrypts all key vectors $\boldsymbol{k} \in Key$ in the plain VDB and then computes the mean vector $\boldsymbol{\mu}$ of encrypted key vectors. The owner then constructs a covariance matrix $\boldsymbol{\Sigma}$ of the encrypted key vectors by (4), i.e. $\boldsymbol{\Sigma} = E\left((\mathrm{Enc}_k(\boldsymbol{k}) - \boldsymbol{\mu})(\mathrm{Enc}_k(\boldsymbol{k}) - \boldsymbol{\mu})^T\right)$. The owner decomposes $\boldsymbol{\Sigma}$ into a diagonal matrix, i.e. $\boldsymbol{\Sigma} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^{-1}$, and obtains a whitening matrix $\boldsymbol{W}_k$ by (5). Our extended encryption algorithm for key vectors $\mathrm{Enc}_k^*(\boldsymbol{k})$ is defined by the whitening matrix:

$$\mathrm{Enc}_k^*(\boldsymbol{k}) = \boldsymbol{W}_k^T(\mathrm{Enc}_k(\boldsymbol{k}) - \boldsymbol{\mu}).$$

The other algorithms Enc_q^* and Dec_k^* are also defined as

$$\mathrm{Enc}_q^*(\boldsymbol{q}) = \boldsymbol{W}_k^{-1}\mathrm{Enc}_q(\boldsymbol{q}), \quad \mathrm{Dec}_k^*(\boldsymbol{k}_e) = \mathrm{Dec}_k((\boldsymbol{W}_k^T)^{-1}\boldsymbol{k}_e + \boldsymbol{\mu}).$$

The owner converts the plain VDB to an $EVDB(Key_e, Value_e)$ using the above encryption algorithm and informs users of Enc_q^*, Dec_k^*, Dec_v, and $\boldsymbol{\mu}$.

Queries using this extended encryption algorithm become the following. Presuming that a user asks to find similar vectors of a query vector $\boldsymbol{q}$ with a threshold α, i.e. the user searches tuples associating with key vectors $\boldsymbol{k}$ satisfying $\mathrm{sim}(\boldsymbol{k}, \boldsymbol{q}) \geq \alpha$, then we are employing cosine similarities and assuming that vectors are normalized so that we deem the condition is $\boldsymbol{k} \cdot \boldsymbol{q} \geq \alpha$. This request is changed for the EVDB and it becomes finding tuples having encrypted key vectors $\boldsymbol{k}_e^* = \mathrm{Enc}_k^*(\boldsymbol{k})$ satisfying $\boldsymbol{k}_e^* \cdot \boldsymbol{q}_e^* \geq \alpha - \boldsymbol{\mu} \cdot \mathrm{Enc}_q(\boldsymbol{q})$, so that the user sends $\boldsymbol{q}_e^* = \mathrm{Enc}_q^*(\boldsymbol{q})$ and $\alpha^* = \alpha - \boldsymbol{\mu} \cdot \mathrm{Enc}_q(\boldsymbol{q})$ as a query.

Table 3. Differences between data maintained in each database.

plain VDB	existing EVDB	proposed EVDB
$(\boldsymbol{k}, v)$	$(\mathrm{Enc}_k(\boldsymbol{k}), \mathrm{Enc}_v(\boldsymbol{v}))$	$(\mathrm{lsh}(\mathrm{Enc}^*_k(\boldsymbol{k})), \mathrm{Enc}^*_k(\boldsymbol{k}), \mathrm{Enc}_v(v))$

Proposition 4.2. The new condition $\boldsymbol{k}^*_e \cdot \boldsymbol{q}^*_e \geq \alpha - \boldsymbol{\mu} \cdot \mathrm{Enc}_q(\boldsymbol{q})$ is able to select correct tuples.

Proof:
The inner product of $\boldsymbol{k}^*_e$ and $\boldsymbol{q}^*_e$ is evaluated as

$$\begin{aligned}\boldsymbol{k}^*_e \cdot \boldsymbol{q}^*_e &= (\boldsymbol{W}_k^T(\mathrm{Enc}_k(\boldsymbol{k}) - \boldsymbol{\mu}))^T \boldsymbol{W}_k^{-1}\mathrm{Enc}_q(\boldsymbol{q}) \\ &= (\mathrm{Enc}_k(\boldsymbol{k}) - \boldsymbol{\mu})^T \boldsymbol{W}_k \boldsymbol{W}_k^{-1}\mathrm{Enc}_q(\boldsymbol{q}) = \mathrm{Enc}_k(\boldsymbol{k}) \cdot \mathrm{Enc}_q(\boldsymbol{q}) - \boldsymbol{\mu} \cdot \mathrm{Enc}_q(\boldsymbol{q})\end{aligned}$$

so that $\boldsymbol{k}^*_e \cdot \boldsymbol{q}^*_e \geq \alpha - \boldsymbol{\mu} \cdot \mathrm{Enc}_q(\boldsymbol{q})$ is equal to $\mathrm{Enc}_k(\boldsymbol{k}) \cdot \mathrm{Enc}_q(\boldsymbol{q}) \geq \alpha$. Those encryption algorithms proposed in existing work $\mathrm{Enc}_k(\boldsymbol{k})$ and $\mathrm{Enc}_q(\boldsymbol{q})$ have a property by which inner products between an encrypted key vector and an encrypted query vector are holding the same value as those between plain vectors of them; i.e. $\mathrm{Enc}_k(\boldsymbol{k}) \cdot \mathrm{Enc}_q(\boldsymbol{q}) = \boldsymbol{k} \cdot \boldsymbol{q}$. Therefore the new conditions work as expected. □

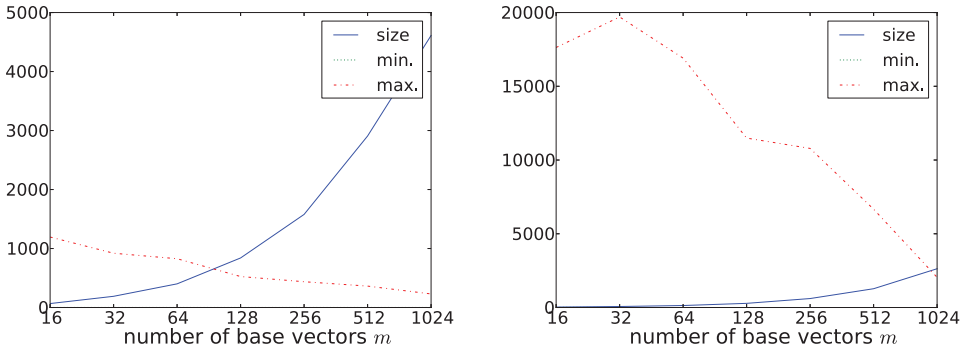
Table 2 presents a summary of the differences of encrypted key vectors and query conditions between original protocols of EVDB and our proposed protocol.

A database server (a cloud service) that hosts the EVDB generates m random vectors and constructs a hash function defined as (2). The server converts the database, which is deployed by the database owner in the following manner. First, the server computes an LSH value $\mathrm{lsh}(\boldsymbol{k}^*_e)$ for all encrypted key vector $\boldsymbol{k}^*_e$ in EVDB; then it adds the LSH values to each tuple. Therefore, the server maintains a database defined as $EVDB^*(LSH, Key_e, Value_e)$, where LSH is an attribute consisting of those LSH values. We denote the set of LSH values in the server as S_{lsh}. The server is able to compute that set easily by group queries, etc. Table 3 presents a summary of the difference between tuples maintained in plain VDB, existing EVDB, and our proposed EVDB.

The querying process using our filter is the following. When a user of the database sends a query, the server receives an encrypted query vector $\boldsymbol{q}^*_e$ and a threshold α^*, where $\alpha^* = \alpha - \boldsymbol{\mu} \cdot \mathrm{Enc}_q(\boldsymbol{q})$, as we described above. Then the server computes an LSH value of the query vector $h_q = \mathrm{lsh}(\boldsymbol{q}^*_e)$. Using this LSH value, the server finds a set of LSH values $S_{cand} \subset S_{lsh}$ s.t. $\forall h \in S_{cand}$

$$\cos\left(\pi(1 - \Pr[h = h_q])\right) \geq \alpha^*. \tag{7}$$

From (3), similarities between encrypted query vector $\boldsymbol{q}_e$ and encrypted key vectors that are associated with the LSH values satisfying the condition (7) are approximately greater or equal to α^*. Therefore, the server need not compute inner products between $\boldsymbol{q}_e$ and tuples of which LSH attribute values are not included in S_{cand}. This is the filtering method of our proposal. Finally, the server searches only tuples satisfying the LSH attribute values included in S_{cand} and satisfying $\boldsymbol{k}_e \cdot \boldsymbol{q}_e \geq \alpha^*$, which is the actual query condition. Then the server returns the tuples to the user as a query result.



(a) $n = 10000$ (b) $n = 100000$

Figure 2. The number of distinct LSH values, and the max. and the min. number of key vectors having the same LSH values.

5. Evaluation

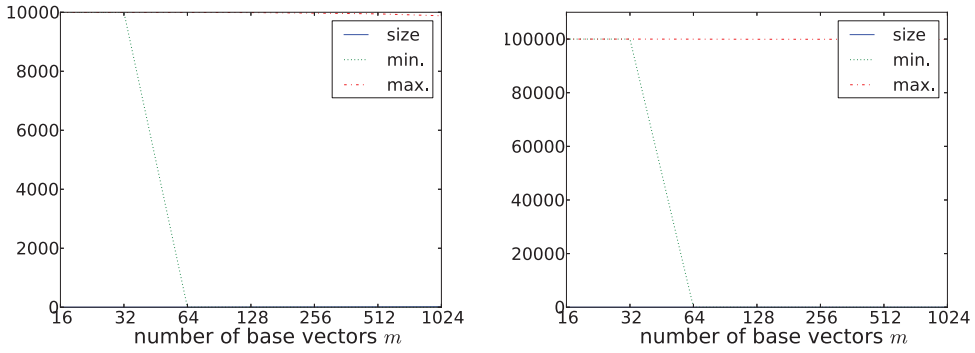
We have conducted some experiments to evaluate the efficiency of our proposed filtering methodology. We choose an EVDB using IPP method, which we introduced in section 2 for our evaluations. The EVDB is made from a plain database consisting of n-tuples. Using this EVDB, we evaluate the following:

1. how many sub spaces LSH splits a vector space of encrypted key vectors into and how many vectors are assigned in each sub space,
2. relation between LSH and recalls of queries,
3. relation between LSH and query processing times.

We developed all programs for this evaluation using Python 2.7. We run them on a PC that consists of Intel®Core™i7-860 Processor (8M Cache, 2.80 GHz), 8GB RAM, and Ubuntu 12.04 LTS.

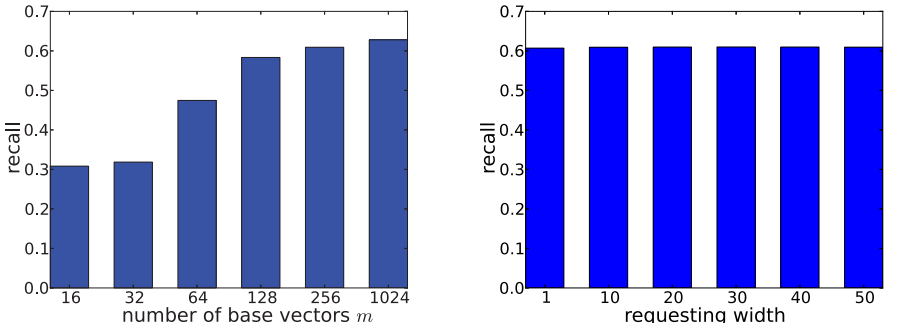
Fig. 2 shows data of LSH values in EVDBs managing $n = 10,000$ tuples and $n = 100,000$ tuples. The x-axes of those graphs show the number of base vectors m used to construct LSH. The *size* graphs show that the number of distinct LSH values defined by (2) is produced in the EVDB, which shows into how many sub spaces the encrypted key vectors space is divided. The *min.* and *max.* graphs respectively present the minimum and maximum number of key vectors assigned a sub space. The larger the *max.* value is, the more encrypted key vectors will be assigned into the same sub space, which means that LSH does not differentiate well and that our filter also does not prune unsimilar vectors. Minimum values, however, represent the minimum number of vectors that the LSH can distinguish. In those cases, the minimum values are 1 in all conditions. From Fig. 2, larger m provides larger *size* so that LSH can distinguish key vectors minutely. The maximum values become smaller when the m become larger. This fact also supports larger m provides good distinguishability for our filter. Finally, this tendency is independent of the total number of tuples n.

Fig. 3 shows statistics of LSH in the same condition as Fig. 2 but without whitening transformation, this figure shows the effectiveness of whitening transformation in our filter. Without whitening transformation, *size* graphs have small values (1 in almost cases) and are independent of m, which means that



(a) $n = 10000$ (b) $n = 100000$

Figure 3. Number of distinct LSH values, and the max. and min. numbers of key vectors having the same hash values w/o whitening transformation.

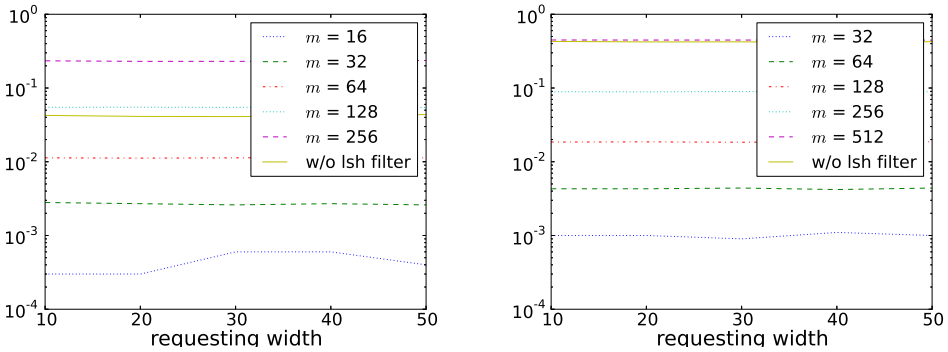


(a) number of base vectors vs. recall (b) requesting width vs. recall ($m = 256$)

Figure 4. Average recalls ($n = 1000$).

splitting a vector space does not work well and that this LSH cannot distinguish key vectors. Looking at the *max.* graphs, they are nearly identical to the total number of tuples, so that in these cases, almost all vectors are assigned into the same sub space and LSH has no effects for filtering. Actually, in the cases of $m < 64$, the minimum values are equal to n. which means there is only one sub space. However, in the cases of $m \geq 64$, the minimum values remain small, so that these LSH can distinguish some vectors. Nevertheless, that performance is insufficient. Comparison of Fig. 2 and Fig. 3 seems to show that the whitening transformation we use provides effective distinguishability for our filter.

Next, we evaluate recalls of queries. Our methodology computes original queries after filtering so that no query result contains false positives. Therefore, what we should consider is only recalls, i.e. false negatives. Fig. 4 shows recalls in several numbers of base vectors m and requesting widths. Fig. 4(a) shows relations between the number of base vectors m and recalls. It shows recalls depend on the number of base vectors; more base vectors achieve higher recalls. Fig. 4(b) portrays relevance between querying widths and recalls. In this figure, we choose $m = 256$. Recalls are independent of query widths.



(a) $n = 10,000$ (b) $n = 100,000$

Figure 5. Query processing time (s).

Finally we measure the query processing times in servers. Fig. 5 presents the results. In both graphs, the x-axis shows how many widths each request queries. The y-axis shows the query processing times in log scale. The query processing times do not include times in client times and communication times. We measure them with several numbers of base vectors m and also without using our filter. In both cases of $n = 10,000$ and $n = 100,000$, smaller m leads to smaller computational time. However, comparison to Fig. 4 seems to show that it is true because the LSH filters necessary tuples too much. For $n = 10,000$, Fig. 5(a) shows using LSH filter with $m > 128$ takes longer time than without a filter. That is true because if the amount of tuples in an EVDB is small, checking all tuples does not take such a long time and computing LSH filter requires some time. However, in the case of $n = 100000$, the worst case of LSH ($m = 512$) takes almost equal computational times to the case of without filtering. For smaller m, using the LSH filter gives us quick query processing. Thinking about the y-axis is in log scale, even $m = 256$ achieves fast query processing.

6. Conclusion

As described in this paper, we introduce a filtering methodology for query processing in encrypted vector databases (EVDBs). The proposed methodology employs locality-sensitive hashing (LSH) and whitening transformation to reduce the query processing cost in servers of EVDBs which have encrypted vectors and for which usual indexing methodologies such as R-tree are inapplicable. From the experimental evaluation, which is discussed in section 5, Fig. 2 and Fig. 3 suggest that the use of whitening transformation allows LSH to hash vector spaces efficiently even if the target vector spaces (spaces of encrypted key vectors) are skewed. Fig. 5 also shows that depending on how many base vectors m we choose for LSH, our methodology can reduce query processing times in servers for a suitable m for each EVDB.

Our filtering methodology uses an approximation (3), so that, as Fig. 4 shows, query results obtained using the proposed methodology might have false negative errors. Therefore, our filter is applicable to EVDBs in which users are not expecting perfect query results. We will modify our filter to increase the accuracy of query results as a subject for our future work.

References

[1] Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order Preserving Encryption for Numeric Data, *Proc. of the 23rd ACM SIGMOD International Conference on Management of Data*, ACM Press, New York, NY, USA, 2004, ISBN 1581138598.

[2] Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-Preserving Symmetric Encryption, *Proc. of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer-Verlag, Cologne, Germany, 2009.

[3] Boldyreva, A., Chenette, N., Neill, A. O.: Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions, *Proc. of the 31st International Cryptology Conference*, Santa Barbara, CA, USA, 2011.

[4] Charikar, M. S.: Similarity Estimation Techniques from Rounding Algorithms, *Proc. of the 34th Annual ACM Symposium on Theory of Computing*, ACM Press, Montreal, Quebec, Canada, 2002, ISBN 1581134959.

[5] Gionis, A., Indyk, P., Motwani, R.: Similarity Search in High Dimensions via Hashing, *Proc. of the 25th International Conference on Very Large Data Bases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[6] Hacigumus, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over Encrypted Data in the Database-Service-Provider Model, *Proc. of the 21st ACM SIGMOD International Conference on Management of Data*, ACM Press, Madison, WI, USA, 2002, ISBN 1581134975.

[7] Hore, B., Mehrotra, S., Tsudik, G.: A Privacy-Preserving Index for Range Queries, *Proc. of the 30th International Conference on Very Large Data Bases*, VLDB Endowment, Toronto, ON, Canada, 2004.

[8] Hua, Y., Xiao, B., Veeravalli, B., Feng, D.: Locality-Sensitive Bloom Filter for Approximate Membership Query, *IEEE Transactions on Computers*, **61**(6), June 2011, 817–830, ISSN 0018-9340.

[9] Kawamoto, J., Yoshikawa, M.: Private Range Query by Perturbation and Matrix Based Encryption, *Proc. of the Sixth IEEE International Conference on Digital Information Management*, IEEE Computer Society, Melbourne, Australia, 2011, ISBN 9781457715396.

[10] Kirsch, A., Mitzenmacher, M.: Distance-Sensitive Bloom Filters, *The 18th Workshop on Algorithm Engineering and Experiments*, Miami, FL, USA, 2006.

[11] Kulis, B., Grauman, K.: Kernelized Locality-Sensitive Hashing for Scalable Image Search, *Proc. of the 12th IEEE International Conference on Computer Vision*, IEEE Computer Society, Kyoto, Japan, 2009, ISBN 9781424444199.

[12] Lu, Y.: Privacy-Preserving Logarithmic-time Search on Encrypted Data in Cloud, *Proc. of the 19th Annual Network & Distributed System Security Symposium*, San Diego, CA, USA, 2012.

[13] Oliveira, S. R. M., Zaïane, O. R.: Privacy Preserving Clustering By Data Transformation, *Proc. of the 18th Brazilian Symposium on Databases*, UFAM, Manaus, AM, Brazil, 2003.

[14] Rane, S., Sun, W., Vetro, A.: Privacy-preserving approximation of L1 distance for multimedia applications, *In Proc. of the 2010 IEEE International Conference on Multimedia and Expo*, IEEE Computer Society, Singapore, July 2010, ISBN 978-1-4244-7491-2, ISSN 1945-7871.

[15] Wang, H., Lakshmanan, L. V. S.: Efficient Secure Query Evaluation over Encrypted XML Databases, *Proc. of the 32nd International Conference on Very Large Data Bases*, VLDB Endowment, Seoul, Korea, September 2006.

[16] Wang, J., Wu, S., Gao, H., Li, J., Ooi, B. C.: Indexing Multi-dimensional Data in a Cloud System, *Proc. of the 30th ACM SIGMOD International Conference on Management of Data*, ACM Press, Indianapolis, IN, USA, 2010.

[17] Wang, Z.-F., Dai, J., Wang, W., Shi, B.-L.: Fast Query over Encrypted Character Data in Database, *Computational and Information Science*, **4**(4), 2005, 1027–1033.

[18] Wong, W. K., Cheung, D. W.-L., Kao, B., Mamoulis, N.: Secure kNN Computation on Encrypted Databases Categories and Subject Descriptors, *Proc. of the 35th SIGMOD International Conference on Management of Data*, ACM Press, Providence, RI, USA, 2009.

[19] Zhang, Z., Ooi, B. C., Parthasarathy, S., Tung, A. K. H.: Similarity Search on Bregman Divergence: Towards Non-Metric Indexing, *Proc. of the 35th International Conference on Very Large Data Bases*, VLDB Endowment, Lyon, France, 2009.