Privacy Preserving Ranked Soft Similarity Search over Encrypted Cloud Data

Shiyu Ji
shiyu@cs.ucsb.edu

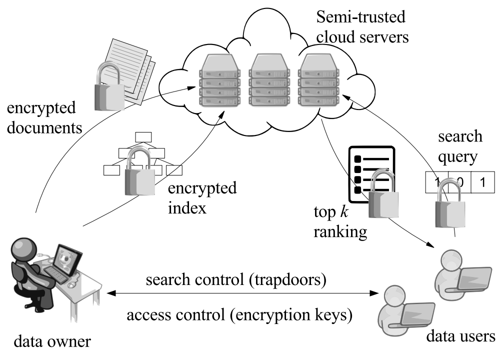

Fig. 1. The architecture of ranked similarity search over encrypted cloud data.

I. Introduction

We summarize our major contributions as follows:

1) We are the first to formalize the privacy of query unlinkability for searchable symmetric encryption over cloud data. We also propose an improved encryption and search scheme that can achieve reasonable bounds of privacy leakage up to our definition.
2) We also investigate the possibility to do soft similarity search over encrypted cloud data. We find that it is possible for the cloud server to search and rank over encrypted documents, without knowing the exact parameters of soft similarity, e.g., the entries in the similarity matrices.

II. Problem Formulation

A. The System and Threat Model

As in Fig. 1, we consider a cloud data hosting service which consists of three entities: the data owner, the data users and the semi-trusted cloud data server. The data owner has a collection of documents that need to be outsourced to the semi-trusted cloud server. The outsourced documents must be encrypted since the data owner does not want to reveal the documents to the semi-trusted server. The encryption scheme will be given later. On the other hand, the data users want to search over the encrypted documents. To enable this, the data owner prepares a search index of the outsourced documents, and then uploads the encrypted index to the server. On a similarity search request, an authorized data user firstly gets the corresponding trapdoors and keys from the data owner, and uses the keys to encrypt the query, and then sends the encrypted search query to the cloud server. Upon receiving the encrypted search query, the server should securely search on the index (i.e., without revealing any information about the documents), and return the top k most similar documents in the encrypted form to the user.

We assume the cloud server is *semi-trusted*, or *honest-but-curious*, i.e., the server will honestly do as described above, but it also wants to know any information of the outsourced documents and the submitted search queries. We consider the *Ciphertext-Only Attack*, in which the server only has the submitted encrypted documents and queries, but it initially has no knowledge about their plaintext. We will also consider another stronger attack, in which we assume the server has some knowledge about the outsourced content, e.g., the frequency distribution. Thus some statistical attacks may be launched.

B. Our Goals

We want to design an efficient and secure ranked search scheme over encrypted cloud data that satisfies the requirements above, and we also want it to keep privacy to some extent.

Efficiency. The algorithms should be efficient, e.g., 1) encryption over the documents, index and queries, 2) search over the encrypted index, etc.

Dynamic updating support. The data owner should be able to update the documents and index efficiently at any time.

Confidentiality. The content of the documents, the entries in the index and the submitted queries should not be revealed to the cloud server.

Privacy preserving. We want to prevent the cloud server from learning anything about the outsourced documents, index or queries submitted by users. We will investigate the cases as follows:

- **Keyword privacy**. The server should not be able to know any information about the keywords or terms in the search request.
- **Query unlinkability**. The server should be difficult to tell whether two encrypted queries are from the same search request (e.g., the same keywords or index terms) if these two queries are generated by identical or similar search requests. Note that the server can easily tell apart the queries from very different requests, since the returned top k documents could be very different.

C. Notations and Definitions

- $\mathbf{F}$ — The collection of the documents to be outsourced: $\mathbf{F} = \{f_1, f_2, \cdots, f_n\}$ where f_i denotes the i-th document.

- $\mathbf{C}$ — The collection of the encrypted documents in $\mathbf{F}$: $\mathbf{C} = \{c_1, c_2, \cdots, c_n\}$ where each c_i is the encrypted form of f_i.
- I — The index built for $\mathbf{C}$: $I = \{I_1, I_2, \cdots, I_n\}$. Denote by I_i the entry of c_i. Each I_i is a vector determined by the keywords and index terms in f_i.
- D — The encryption of index I. Each D_i in D is the encryption of entry I_i.
- $\mathbf{W}$ — The set of the keywords and terms: $\mathbf{W} = \{W_1, W_2, \cdots, W_m\}$.
- r — The search request r, which contains the keywords and index terms that are interesting to the user.
- q — The query vector determined by the keywords and index terms in the search request r.
- Q — The encryption of query q.
- $\hat{F}$ — The top k documents: $\hat{F} = \{\hat{f}_1, \hat{f}_2, \cdots, \hat{f}_k\}$ with the highest similarities to the query q.
- $V[i]$ — The i-th component of the vector V. For example, if $V = (0, 1, 2)^T$, then $V[1] = 0$, $V[2] = 1$ and $V[3] = 2$.

Cosine Similarity [1], [2], [3], [4] is a popular measure in information retrieval community to evaluate the similarity between two vectors in vector space model [1], [2], [3]. Given a set of vectors $v_i = (v_{i,1}, v_{i,2}, \cdots, v_{i,m})^T$ where each vector is normalized to a unit length, and has at most m features (if v_i does not have feature k, we can set $v_{i,k}$ to be zero), the cosine similarity between vectors is defined as follows:

$$Sim(v_i, v_j) = \sum_{k=1}^{m} v_{i,k} \cdot v_{j,k}.$$

Note that the cosine similarity of two vectors equals to their dot product. We say two vectors are similar if their cosine similarity exceeds some threshold τ. For a ranked search over the documents $\mathbf{F}$, given the query vector q, we want the cloud server to return the top k documents $\{\hat{f}_i\}$ with the highest cosine similarities $Sim(\hat{f}_i, q)$. In post-processing, the search user can decrypt the returned k documents and check whether among them there are some similar ones, i.e., with similarities larger than the threshold τ.

Soft Cosine Similarity [5] is a generalized version of cosine similarity. The major difference is that for soft similarity, we assume there can be similarity between features. For example, in medical patient profiles, age and height are two different features, but their values often share some similarities: adults usually have more heights than young people. However, age and gender are two features that are almost orthogonal. With the above observation, we can use a real number s_{ij} between 0 and 1 to represent the similarity between the features i and j. Considering all the combinations, we have a $m \times m$ matrix $\mathbf{S}$ called similarity matrix, where m is the number of features, and each entry s_{ij} of $\mathbf{S}$ is the similarity between features. The soft cosine similarity between the vectors u and v is defined as follows:

$$SoftSim(u, v) = \frac{\sum_{i,j} s_{ij} u_i v_j}{\sqrt{\sum_{i,j} s_{ij} u_i u_j} \sqrt{\sum_{i,j} s_{ij} v_i v_j}}. \quad (1)$$

Usually we assume $s_{ii} = 1$ along the diagonal of $\mathbf{S}$. If the similarity matrix $\mathbf{S}$ is a unit matrix, then the soft cosine similarity is the same as the classical cosine similarity.

Similarity Distance between two normalized vectors X and Y is defined as the minimum value d such that

$$|Sim(X, q) - Sim(Y, q)| < d$$

for any normalized vector q. Here we can choose (soft) cosine similarity. For other similarity measure, a distance concept can be similarly established.

Statistical Distance [6], [7] defines a distance measure between two distributions. Consider two random variables X and Y defined over domain Ω. The statistical distance between X and Y is defined as

$$\Delta(X, Y) = \frac{1}{2} \sum_{\alpha \in \Omega} |\Pr[X = \alpha] - \Pr[Y = \alpha]|.$$

If X and Y are continuous random variables, then the summation above should be replaced by integration. An important property of statistical distance is that for any randomized algorithm A, we have $\Delta(A(X), A(Y)) \leq \Delta(X, Y)$. That is, statistical distance cannot be increased by any randomized algorithm.

Unlinkable Privacy. We define unlinkable privacy in terms of the ability to distinguish two queries from possibly different search requests. The idea is similar to Definition 1 in [8].

Definition 1. *Let $\Delta(X, Y)$ be the statistical distance between two probability distributions X and Y. Let $A(X)$ be the beliefs on X of a randomized algorithm A. Given two query distributions X and Y generated by our scheme with similarity distance d, we say our scheme preserves the (δ, d)-unlinkable privacy of queries against A if*

$$\Delta(A(X), A(Y)) \leq \delta$$

for some bound $\delta > 0$. Here X and Y are independent from each other.

We will use the above privacy notion to study query unlinkability. If two queries X, Y are independently generated from the same request, it is clear that X and Y are also identically distributed, and thus their distance is zero. Hence we only need to consider the case when X and Y have some similarity distance $d > 0$ between them, i.e., $|Sim(X, Z) - Sim(Y, Z)| \leq d$ for any normalized query vector Z.

Secure ranked similarity search scheme can be summarized as follows:

- **Setup**(1^λ): the data owner takes a security parameter λ as input and generates a private key sk that will be used to encrypt the outsourced documents.
- **BuildIndex**(**F**, sk): using the key sk, the data owner builds an encrypted index I given the documents $\mathbf{F}$. Each entry I_i is an encrypted vector determined by the keywords and index terms contained in the document f_i.
- **Query**(**W**, sk, r): the authorized user computes the query vector q based on the keywords and terms in the search request r, and sends Q (the encryption of q by key sk) to the cloud server. We want two similar queries Q's can achieve unlinkable privacy for some upper bound.

- **Search**(Q, k, **C**): the cloud server searches over the encrypted documents **C** and determines the top k documents $\hat{F}$ that are the most similar to the query vector q. We want the server can figure out the most similar k documents as accurate as possible given the encrypted q and index entries I_i. The similarity computation can be based on (soft) cosine similarity.

III. Our Proposed Scheme

In this section we firstly recall the secure dot product encryption [9] (which was adapted from [10]). Then we propose our scheme for ranked similarity search.

A. Secure Dot Product Encryption

In Cao et al's scheme [9], the cloud server can approximately compute the dot product between an index entry I_i and the query vector q. The private key sk consists of a $(m+2)$-bit string S and two $(m+2) \times (m+2)$ invertible matrices M_1 and M_2, where m is the number of features in each vector. To encrypt, we add two more dimensions to both I_i and q, and we denote the extended vectors as D_i and Q respectively. The components of the two added dimensions are given as follows:

1) For all D_i, $D_i[m+1]$ is set to a random number ε_i, and $D_i[m+2]$ is set to 1, and
2) $Q[m+1]$ is set to 1, and then all the first $(m+1)$ components are scaled by a non-zero random number r, and finally $Q[m+2]$ is set to another independent random number t.

Then D_i is randomly split into two random vectors D_i' and D_i'', and Q is also randomly split into Q' and Q''. The string S is a splitting indicator: if the k-th bit of S is 0, then the $D_i'[k]$ and $D_i''[k]$ are the same as the $D_i[k]$, while the sum of $Q'[k]$ and $Q''[k]$ should be $Q[k]$; if the k-th bit of S is 1, then the process is similar except that the positions of D_i and Q are exchanged. The split index entry is encrypted as $(M_1 D_i', M_2 D_i'')$, and the split query vector is encrypted as $(M_1^{-1} Q', M_2^{-1} Q'')$. Upon an encrypted query Q, the cloud server computes the approximate dot product given encrypted D_i as follows:

$$\begin{aligned}
&(M_1 D_i', M_2 D_i'') \cdot (M_1^{-1} Q', M_2^{-1} Q'') \\
=&D_i' \cdot Q' + D_i'' \cdot Q'' \\
=&\sum_{S[k]=0} D_i'[k]Q'[k] + D_i''[k]Q''[k] \\
&+ \sum_{S[j]=1} D_i'[j]Q'[j] + D_i''[j]Q''[j] \\
=&\sum_{S[k]=0} D_i[k](Q'[k] + Q''[k]) + \sum_{S[j]=1} (D_i'[j] + D_i''[j])Q[j] \\
=&\sum_{S[k]=0} D_i[k]Q[k] + \sum_{S[j]=1} D_i[j]Q[j] \\
=&D_i \cdot Q = (I_i, \varepsilon_i, 1) \cdot (rq, r, t) \\
=&r(I_i \cdot q + \varepsilon_i) + t.
\end{aligned}$$

Clearly if the bounds of the random ε_i and t are small enough, the result is approximately proportional to $Sim(I_i, q)$. In [9] ε_i is considered when computing similarity between I_i and any query q. However in this paper we use cosine similarity, and thus we can set each ε_i to be zero or some very small value. Then the result is close to $rI_i \cdot q + t$. Note that only r and t are not enough to preserve privacy, since they can be easily reverse-engineered: suppose the cloud server collects extensive $rI_i \cdot q + t$ values by fixing q, then

- often there exists an index entry I_k that is very distinct from q. I_k gives an almost zero similarity and thus t is completely compromised by finding the lowest value of $rI_i \cdot q + t$. And
- the highest value of them can probably be used to estimate r since the maximum similarity $I_i \cdot q$ can usually be learned from historic search experience.

Thus we need to introduce more randomness to better preserve privacy, e.g, the similarity, the query unlinkability, etc.

B. Improved Scheme

We observe that the above scheme is not secure since all the "safeguard" random numbers are associated with the query side, but none of them is chosen by the index side. Hence the basic idea of our improved scheme is to introduce more randomness during the encryption of index entries.

- **Setup**(1^λ): the data owner takes a security parameter λ as input and generates a private key sk, which consists of a $(m+2)$-bit string S and two $(m+2) \times (m+2)$ invertible matrices M_1 and M_2. All of these are randomly chosen. Here m should be proportional to λ. If m is more than the number of features in each vector, we can extend the dimensions by adding some dummy keywords, which always contribute zero when computing similarities.
- **BuildIndex**(**F**, sk): each m-dimensional entry I_i associated with f_i is first extended to a $(m+2)$-dimensional vector D_i. Note that $D_i[m+1]$ is set to a random number ε_i and $D_i[m+2] = 1$. Then based on S, D_i is split into D_i' and D_i''. The splitting procedure based on S in sk has been described above.
  After all the entries have been encrypted, we divide the entries into partitions by using dissimilarity detection [3]. For each partition, we build an index tree by using MD algorithm [1] ([4] gives a similar index tree building solution). All the interior nodes in the index tree are also encrypted using sk.
- **Query**(**W**, sk, r): the authorized user computes the query vector q based on the keywords and terms in the search request r, and sends Q', Q'' (the encryption of q) to the cloud server. The encryption procedure of q is similar as above, except that $Q[m+2]$ is set to $r\sigma$, where σ is a random number chosen by the user (it will be discussed later). Finally the vector of Q is now $(rq, r, r\sigma)$.
- **Search**(Q, k, **C**): the cloud server securely searches over the encrypted index of each partition, and determines the top k documents $\hat{F}$ with the highest values of $D_i \cdot Q$. At last the server returns the documents $\hat{F}$ to the user.

It is not difficult to see now the cloud server can compute the value as follows:

$$\begin{aligned}&(M_1 D_i', M_2 D_i'') \cdot (M_1^{-1} Q', M_2^{-1} Q'')\\ =&D_i' \cdot Q' + D_i'' \cdot Q''\\ =&D_i \cdot Q = (I_i, \varepsilon_i, 1) \cdot (rq, r, r\sigma)\\ =&r(I_i \cdot q + \varepsilon_i + \sigma).\end{aligned} \tag{2}$$

Clearly if the bound of $\varepsilon_i + \sigma$ is small enough, the value above can be used to rank the similarities, since it is proportional to $Sim(I_i, q)$. However, we will use ε_i and σ to preserve the privacy of similarity and query unlinkability.

IV. Privacy Analysis

In this section we assume the attacker has more knowledge than only ciphertext. In particular, we assume the attacker has a precise estimation of r and t as above, and we also assume the attacker has information of keyword frequency distribution based on historic experience. We make such an assumption since we have no way to consider all the strategies the attacker could possibly take. If our scheme can achieve security up to a higher standard, we can make sure it is also secure in the real world. We can show that our scheme can achieve privacy under certain settings by using well known computation-theoretical arguments.

A. Similarity and Keyword Privacy

Since we assume the attacker knows the value of r, $I_i \cdot q + \varepsilon_i + \sigma$ is exposed to the cloud server. However the cloud server is not sure about the value of $\varepsilon_i + \sigma$, which is chosen at random and different for every query and every index entry. This randomness can hide the true similarity value $I_i \cdot q$. Hence the cloud server cannot learn the exact similarity between I_i and q. On the other hand, the bound of $\varepsilon_i + \sigma$ cannot be too large; otherwise it will decrease the accuracy of our ranked search scheme. If the bound of randomness is very small, then the accuracy is high but the similarity privacy can be leaked. If the bound is very large, then the privacy is well preserved, but the accuracy can be significantly decreased.

B. Query Unlinkability

Consider two query data sets E_1 and E_2 (see Section II-C), which are either from the same search request, or from two requests with similarity distance d between them. Based on the assumptions above, for the query pair $(q_{i,1}, q_{i,2})$ in E_i, the cloud server can learn the values $I_i \cdot q_{i,j} + \varepsilon_i + \sigma_j$ where j is 1 or 2, given any encrypted index entry I_i. Note that the part of ε_i is fixed for both queries, and σ_1, σ_2 are independently drawn. Define random variables $\{X_i\}$ as follows:

$$\begin{aligned}X_i =&(I_i \cdot q_{i,1} + \varepsilon_i + \sigma_1) - (I_i \cdot q_{i,2} + \varepsilon_i + \sigma_2)\\ =&I_i \cdot (q_{i,1} - q_{i,2}) + (\sigma_1 - \sigma_2).\end{aligned}$$

Note that the term $I_i \cdot (q_{i,1} - q_{i,2})$ is zero if the two queries are from the same request, and is no more than d otherwise. Let each σ_j be drawn from normal distribution $N(0, \nu^2)$. Note that the density always increases as we get closer to the mean value of normal distribution. Thus if the two queries in X_i are from the same request, X_1 and X_2 are exactly identically distributed. Otherwise, both of them are normally distributed with variance $2\nu^2$, but the distance between their mean values is not zero. Clearly for any point in the real numbers $\mathbb{R}$, if it is closer to the mean value of X_1, then the probability density of X_1 at that point is higher than that from X_2, and vice versa. We assume the queries in X_1 are from the same request, while those from X_2 are from two requests with similarity distance d. Then the statistical distance between X_1 and X_2 can be computed as follows:

$$\begin{aligned}\Delta(X_1, X_2) =& \frac{1}{2} \int_{-\infty}^{+\infty} \Big| p[X_1 = x] - p[X_2 = x] \Big| dx\\ =&\frac{1}{2} \int_{|x - \mathbb{E}[X_1]| < |x - \mathbb{E}[X_2]|} p[X_1 = x] - p[X_2 = x] dx\\ &+ \frac{1}{2} \int_{|x - \mathbb{E}[X_1]| > |x - \mathbb{E}[X_2]|} p[X_2 = x] - p[X_1 = x] dx,\end{aligned}$$

where $p(\cdot)$ is the probability density. Clearly if $I_i \cdot (q_{i,1} - q_{i,2}) = d$, the two normal random variables X_1 and X_2 are the most distinctly distributed, and thus have the highest statistical distance. Then

$$\begin{aligned}&\Delta(X_1, X_2)\\ \le&\frac{1}{2} \cdot \left\{ \int_{-\infty}^{d/2} + \int_{d/2}^{+\infty} \right\} \Big| p[X_1 = x] - p[X_2 = x] \Big| dx\\ &\quad \text{s.t. } X_1 \sim N(0, 2\nu^2), X_2 \sim N(d, 2\nu^2)\\ =&\int_{-\infty}^{d/2} p[X_1 = x] - p[X_2 = x] dx\\ =&\Pr[X < d/2] - \Pr[X > d/2] \quad \text{s.t. } X \sim N(0, 2\nu^2)\\ =&2 \Pr[X < d/2 : X \sim N(0, 2\nu^2)] - 1.\end{aligned}$$

We know that applying any randomized algorithm A on the distributions X_1 and X_2 cannot increase their statistical distance $\Delta(X_1, X_2)$, i.e.,

$$\Delta(A(X_1), A(X_2)) \le \Delta(X_1, X_2).$$

Note that we have found an upper bound for $\Delta(X_1, X_2)$. Thus we have bounded the advantage to distinguish these two queries for *any* adversary. Now the remaining question is to see whether the value of the upper bound is small enough.

Fig. 2 gives some upper bounds under different settings of ν and d. Note that if the similarity distance d are small, the achieved upper bound of distinguishing advantage is close to zero, implying any attacker cannot well detect the relations between queries. Thus by setting appropriate ν, it is possible to guarantee that the advantage of *any* attacker must be upper bounded if the similarity distance between two requests is within d. For two queries with too large similarity distance between them, it is clearly impossible to hide their difference for privacy preserving in our scheme, since otherwise the utility of ranked search cannot be satisfied. Hence we have established an upper bound of how much degree of privacy our scheme can preserve.

V. Secure Ranked Soft Similarity Search

In this section we propose another secure search scheme that is based on soft cosine similarity. The major challenge is that

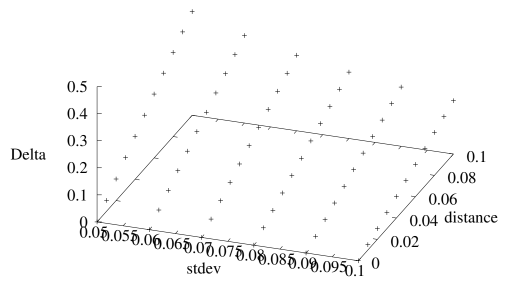

Fig. 2. The upper bounds of statistical distance $\Delta(X, Y)$. Here ν is the standard deviation of noise, and d is the similarity distance between two search requests of X and Y.

each user can have a different belief in the similarities between features. Thus each user may submit queries based on different similarity matrices. However the cloud server has only one encryption version of the index entries, but the search scheme still has to satisfy all the queries from different similarity matrices. Our solution idea is to compute the bounds of dot product between two encrypted vectors. Then based on the computation, we let the cloud server return a little more than k documents to the user, and we shall make sure the user can find the most similar k documents among the returned ones, no matter which similarity matrix this user chooses.

A. Our Scheme for Soft Cosine Similiarity

Usually the similarities between different features are quite less than 1 [5]. For example, if the similarity is defined as $s_{ij} = 1/(1 + |j - i|)$ (Eq. (27) in [5]), then $s_{ij} \leq 0.5$ for any $i \neq j$. Thus we introduce a bound parameter ϵ to represent the maximum off-diagonal entry in the similarity matrix. Without exceeding the bound ϵ, each search user can have the freedom to choose different similarity matrices, e.g., for experimental use or for different beliefs.

Our new scheme is given as follows:

- **Setup**(1^λ): the data owner takes a security parameter λ as input and generates a private key sk, which consists of a random $(m+2)$-bit string S and two random $(m+2) \times (m+2)$ invertible matrices M_1 and M_2. Similarly, m should be proportional to λ.
- **BuildIndex**(**F**, sk): each m-dimensional entry I_i associated with f_i is first extended to a $(m+2)$-dimensional vector D_i. Note that $D_i[m+1]$ is set to a random number ε_i and $D_i[m+2] = 1$. Then based on S, D_i is split into D'_i and D''_i. The splitting procedure based on S in sk has been described above.
  After all the entries have been encrypted, we divide the entries into partitions, and for each partition, we build an index tree by using MD algorithm. All the interior nodes in the index tree are also encrypted using sk.
- **Query**(**W**, sk, r): the authorized user computes the query normalized vector q based on the keywords and terms in the search request r, and sends Q', Q'' (the encryption of q) to the cloud server. The encryption procedure of q is similar as Section III-B, except that there is an additional step at the beginning: we calculate
$$\mathbf{q} = \mathbf{S}q/\sqrt{q^T \mathbf{S} q},$$
  where $\mathbf{S}$ is the similarity matrix chosen by the user, and then encrypt the vector $\mathbf{q}$ as what we did for q in Section III-B.
- **Search**(Q, k, **C**): the cloud server securely searches over the encrypted index of each partition. For each index entry D_i, it will compute two values: 1) an upper bound
$$U_i = D_i \cdot Q,$$
  and 2) a lower bound
$$L_i = U_i/\sqrt{1 + (n-1)\epsilon}.$$
  Then the server will sort all the L_i in descending order, and find the k-th largest L_i (denoted by $\hat{L}_k$), and then include every document, whose index entry D_i satisfies $U_i > \hat{L}_k$, into the set $\hat{F}$. Finally the server returns the documents $\hat{F}$ to the user.

B. Correctness

We argue that if privacy preservation is not required (i.e., $\varepsilon_i = \sigma = 0$), in our scheme the cloud server always returns a set including the k documents which are the most similar to the query based on the similarity matrix S chosen by the user.

Clearly it suffices to show that for each document f_i in $\hat{F}$, its soft cosine similarity is between U_i/r and L_i/r, where r is the random scaling factor chosen by the user when encrypting the query. If this holds, then $\hat{F}$ must contain the k documents with the highest L_i's (called the k leading ones), and all the documents whose similarities can be possibly larger than any of the k leading documents. This ensures the completeness of top k similar documents among $\hat{F}$.

For any document f_i in the returned $\hat{F}$, the user first decrypts it and obtains the index entry I, which is a normalized vector, and then computes the scaling factor in the denominator of Eq. (1):

$$\begin{aligned} I^T \mathbf{S} I &= \sum_{i,j} s_{ij} I[i] I[j] = \sum_i I[i] \sum_j s_{ij} I[j] \\ &\leq \sum_i I[i] \left((1-\epsilon) I[i] + \sum_j \epsilon I[j] \right) \\ &\leq \sum_i I[i] \left((1-\epsilon) I[i] + \epsilon\sqrt{n} \right) \\ &= \sum_i (1-\epsilon) I[i]^2 + \epsilon\sqrt{n} \sum_i I[i] \\ &\leq 1 + (n-1)\epsilon. \end{aligned}$$

If $\varepsilon_i = \sigma = 0$, similarly to Eq. (2), the cloud server can compute

$$\begin{aligned} U_i &= (M_1 D_i', M_2 D_i'') \cdot (M_1^{-1} Q', M_2^{-1} Q'') \\ &= rI \cdot \mathbf{q} = \frac{r I^T \mathbf{S} q}{q^T \mathbf{S} q} = r \frac{\sum_{i,j} s_{ij} I_i q_j}{\sqrt{\sum_{i,j} s_{ij} q_i q_j}} \\ &= \sqrt{I^T \mathbf{S} I} \cdot r \cdot SoftSim(I, q) \\ &\geq r \cdot SoftSim(I, q). \end{aligned}$$

Thus the lower bound satisfies

$$L_i = U_i / \sqrt{1 + (n-1)\epsilon} \leq r \cdot SoftSim(I, q).$$

Hence we have the bounds as

$$L_i \leq r \cdot SoftSim(I, q) \leq U_i.$$

C. Discussions on Privacy

Clearly if we introduce some normal random variables such as ε_i and σ, then the cloud server is difficult to tell the link between two queries from the same request: the two bounds U_i and L_i may be perturbed each time so that the queries become indistinguishable if their requests are close enough. Similarly as before, with non-zero random variables, our scheme can prevent more privacy leakage, but the accuracy of the results may be decreased. Thus it is important to evaluate the decrease in accuracy while using randomness.

References

[1] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. ACM, 2013, pp. 71–82.

[2] M. Alabduljalil, X. Tang, and T. Yang, "Cache-conscious performance optimization for similarity search," in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2013, pp. 713–722.

[3] X. Tang, M. Alabduljalil, X. Jin, and T. Yang, "Load balancing for partition-based similarity search," in *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. ACM, 2014, pp. 193–202.

[4] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 340–352, 2016.

[5] G. Sidorov, A. Gelbukh, H. Gómez-Adorno, and D. Pinto, "Soft similarity and soft cosine measure: Similarity of features in vector space model," *Computación y Sistemas*, vol. 18, no. 3, pp. 491–504, 2014.

[6] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[7] O. Goldreich, "Foundations of cryptography: fragments of a book," 1995.

[8] D. Kifer, "Attacks on privacy and definetti's theorem," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 127–138.

[9] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on parallel and distributed systems*, vol. 25, no. 1, pp. 222–233, 2014.

[10] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 139–152.