ZERBER^{+R}: Top-k Retrieval from a Confidential Index

Sergej Zerr Daniel Olmedilla Wolfgang Nejdl
L3S Research Center & Leibniz University of Hanover
Appelstr. 9a, Hanover 30167, Germany
{zerr, olmedilla, nejdl}@L3S.de

ABSTRACT

Privacy-preserving document exchange among collaboration groups in an enterprise as well as across enterprises requires techniques for sharing and search of access-controlled information through largely untrusted servers. In these settings search systems need to provide confidentiality guarantees for shared information while offering IR properties comparable to the ordinary search engines. Top-k is a standard IR technique which enables fast query execution on very large indexes and makes systems highly scalable. However, indexing access-controlled information for top-k retrieval is a challenging task due to the sensitivity of the term statistics used for ranking.

In this paper we present ZERBER^{+R} – a ranking model which allows for privacy-preserving top-k retrieval from an outsourced inverted index. We propose a relevance score transformation function which makes relevance scores of different terms indistinguishable, such that even if stored on an untrusted server they do not reveal information about the indexed data. Experiments on two real-world data sets show that ZERBER^{+R} makes economical usage of bandwidth and offers retrieval properties comparable with an ordinary inverted index.

1. INTRODUCTION

The number of access-controlled documents shared over enterprise intranets is growing rapidly. Collaboration groups within enterprises require facilities to support effective and efficient retrieval of the top-k documents most relevant to the query while shielding those documents from others' eyes. In the enterprise settings users can participate in a number of collaboration groups and need to obtain the most relevant top-k results from the whole document collection accessible to them.

Top-k is a standard IR technique which enables fast query execution on very large indexes and makes systems highly scalable. Top-k prevents information overload by returning only highly ranked documents most relevant to the user query and allows reducing bandwidth in case group members use mobile devices to access the enterprise's search facilities.

Whereas top-k retrieval of publicly available documents is well-studied, indexing access-controlled information for top-k processing remains a challenging task. Even within a single enterprise (and especially by collaborations within virtual enterprises), competitive working groups are not likely to agree on a single trusted server hosting the index or fully trusted system administrators. Therefore an index created over a set of confidential documents requires specific protection.

Inverted indexes are the standard choice for keyword (full-text) top-k document search. An inverted index is a sequence of *posting lists*, each of which contains the *posting elements* (*elements* for short). Every posting element represents a document which contains particular term and encloses the relevance score used for ranking. Figure 1 shows an inverted index with two posting lists.

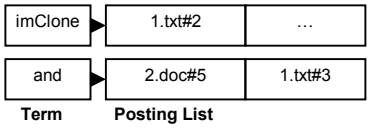

Figure 1: Inverted Index

Unfortunately, an inverted index, which is a highly efficient data structure for the top-k query processing, does not preserve confidentiality of the indexed documents. The content of a document can be easily reconstructed by a straight forward posting lists scan. Relevance scores within the posting elements disclose the number of the indexed documents highly relevant to a specific term. Even if the exact content of the elements is obscured, the number of highly ranked documents can give an industrial spy insides in the compounds which are used in the development of a new chemical process [6].

In general there is a tradeoff between the retrieval efficiency of the index and confidentiality it can provide. On the one hand in order to efficiently answer a query, an index server requires possibly complete ranking information enclosed in its posting elements. On the other hand this information gives undesirable insides into the content of the indexed documents.

In this paper we present ZERBER^{+R}, a novel ranking model which allows for top-k retrieval from a confidential outsourced inverted index without information leakage. This paper makes the following contributions: (i) we introduce the problem of the confidential top-k retrieval from an outsourced inverted index; (ii) we propose ZERBER^{+R}, a ranking model which minimizes information leakage by top-k retrieval from a confidential outsourced inverted index; (iii) we propose a novel relevance score transformation function (***RSTF*** for short) – a heuristic which hides term specific distribution of the relevance score values, making scores of different terms indistinguishable. This

EDBT'09 St.-Petersburg, Russia

heuristic enables including (transformed) relevance scores in the posting elements on an untrusted server to allow the server answering top-k queries; (iv) we demonstrate retrieval efficiency of the index on two real-world data sets.

2. BACKGROUND

In the literature different ways of protecting outsourced information were proposed. While several approaches considered protecting outsourced data by encryption [10, 13], encryption of posting elements in an inverted index does not hide document frequency (the number of documents containing specific term), which can be used by an adversary to reverse-engineer the terms [6]. Probabilistic-based index protection techniques suppress statistical data introducing a controlled amount of uncertainty by including false positive elements in the index [2]. Thereby the lack of precision in search results from the central index represents a tradeoff between search efficiency and confidentiality preservation.

Zerber [22] protects an inverted index by combining the benefits of the both, probabilistic and encryption techniques. It allows obtaining precise search results from an outsourced encrypted index while providing confidentiality guarantees for the indexed documents. Zerber introduces the concept of *r-confidentiality* as a measure of the information degree that can leak from an index.

In order to avoid information leakage if an index server is compromised and provide tunable resistance to statistical attacks Zerber supplements encryption of posting elements with a novel probabilistic term merging scheme. This scheme selectively combines posting lists representing different terms into one posting list (as shown in Figure 2) until a certain probabilistic threshold is met. Thereby the probability of a particular term being related to a concrete posting element does not exceed a certain value r. The combination of the posting lists is called posting list merging. The system allows for tunable index confidentiality/efficiency.

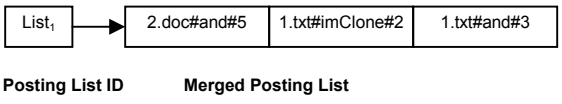

Figure 2: Merged Unencrypted Posting Lists

Unfortunately, neither of existing index protecting approaches supports server-side top-k retrieval, which is essential for retrieval efficiency and economical usage of network resources, as the user is typically not interested in all available search results, but only in the most relevant ones. μ-Serv[2] does not support centralized ranking at all; Zerber randomly distributes posting elements containing encrypted ranking information within the merged posting list, which only allows performing top-k on the client side after the querying client downloads the whole posting list.

2.1 r-Confidentiality

The concept of *r-confidentiality* was introduced in [22]. *r-confidentiality* is a measure of the degree of information that can leak from an index about inaccessible documents, given an adversary's background knowledge of the document corpus or general language statistics. *r-confidentiality* bounds the ability of an adversary Alice to make probabilistic claims about the contents of a document collection. From her background knowledge ***B*** and the parts of the index structure ***I*** that she can access, Alice will know a-priori that a term ***t*** is contained in document ***d*** with a probability $P(t \in d)$. For example, for a set of emails, ***B*** should include $P("Subject" \in d) = 1$, for all ***d*** and ***I***. The probability estimate ***P(X|B)*** about fact ***X*** that Alice can make based on ***B*** can not be controlled, but her ability to refine that estimate when she computes ***P(X|I,B)*** can be limited. In the following, only facts ***X*** of the form "term **t** is in document ***d***" and "term **t** is not in document ***d***" will be considered.

Definition 1: An indexing scheme is *r-confidential* iff

$$\frac{P(X \mid B, I)}{P(X \mid B)} \leq r. \qquad (1)$$

Here, ***r*** is the factor of maximal probability amplification for term ***t*** in ***d*** given ***I***. The indexing scheme offers maximal protection when ***P(X|B) = P(X|I,B)***, i.e. ***I*** does not provide any additional knowledge about ***X***.

Zerber [22] is an *r*-confidential global inverted index for sensitive documents. Zerber relies on a centralized set of *largely untrusted* index servers that hold encrypted posting list elements. To provide tunable resistance to statistical attacks, Zerber employs a novel term merging scheme. In this scheme posting lists are selectively merged together (as shown in Figure 2), such that the probability of a particular term being related to a concrete posting element does not exceed ***r*** times the probability of the term in the document corpus. The probability p_t of occurrence of a term ***t*** in the document corpus ***D*** is represented by its normalized document frequency.

.

Definition 2: a merged posting list is ***r***-confidential iff the term probability amplification inside of the merged posting list does not exceed ***r***. That is, a merging scheme is r-confidential iff

$$\sum_{t_i \in S} p_{t_i} \geq 1/r \qquad (2)$$

where p_t is the probability of occurrence of the term ***t*** in the document corpus ***D***, ***S*** is the set of terms in a merged posting list and ***r*** is the confidentiality parameter. Note that in order to preserve the r-confidentiality property the posting elements representing different terms are randomly distributed within the merged posting list, to resist statistical attacks.

Zerber stores ranking information as well as term and document identifiers within each posting element in an encrypted form. As ranking information is not accessible to the server, Zerber does not support server-side ranking. Thereby the whole merged posting list needs to be retrieved by the querying client to obtain the top-k results.

2.2 Outsourcing Relevance Scores

A naïve approach to provide confidential ranking in an outsourced index is to arrange posting elements in the posting list on the client side before outsourcing them. For instance, an inserting client could ensure that the most relevant top-k posting elements are contained in the head of the posting list. However, this approach does not fit to the collaboration groups' scenario, as the index contains posting elements with different access rights and cannot be rearranged by a single user. Moreover, this approach is impractical in case of frequent index updates.

To allow for top-k retrieval in an ordinary inverted index relevance scores of posting elements are made available to the server. However, the scores increase the amount of information available to the server and thus decrease confidentiality provided by the outsourced inverted index.

In the vector space model [18], which is most widely used in search engines, scoring information is typically based on the term frequency (the number of term occurrences in the document, TF hereafter), which distribution is term specific and can allow adversary to reverse-engineer the terms.

In case of probabilistic-based index protection (e.g. [2]) an index contains a controlled amount of false positive elements. Adding relevance scores to the posting elements in this case would require generation of the realistic relevance scores for the false positive elements to prevent statistical attacks.

Sorting posting elements by their relevance score in a Zerber's merged posting list (Figure 3) may amplify the probability of a particular posting element to be related to a specific term, violating the r-confidentiality guarantees provided by the index. In the worst case it may allow an adversary to overcome the merging i.e. to find out which of the merged terms corresponds to a particular posting element. Consider a merged posting list containing terms "and" and "imClone". By sorting posting elements according to their term frequency, "imClone" would more probably appear in the tail of the list, as it is less frequent, and its relevance score is lower.

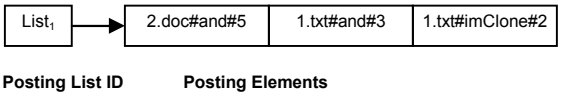

Figure 3: Merged Posting Lists sorted by Term Frequency

2.3 Relevance Score Calculation

In order to allow for efficient top-k retrieval and support index updates in a collaborative environment relevance score needs to be included in each posting element in a way the index server can access. In this section we provide a definition for *scoring function*, and discuss its factors relevant to the confidential ranking.

Powerful top-k server-side ranking techniques were introduced in the literature [18]. The Vector Space Model is the most widely used in search engines for determining document relevance within a collection to a particular query term. In this model, a document ***d*** is represented as a vector, where each term is assigned a specific weight indicating the importance of the term in representing the semantics of the document. Two factors are of importance in the weight assignment: the normalized term frequency, which is the number of term occurrences in the document divided by the document length, and the inverse document frequency (***IDF***) which represents the query term selectivity. At the query time the relevance score (***rscore*** hereafter) of a document ***d*** to a query ***Q*** is computed as follows:

$$rscore(Q,d) = \sum_{q \in Q} \left(\frac{TF_q}{|d|} \cdot IDF_q \right) = \sum_{q \in Q} \left(\frac{TF_q}{|d|} \cdot \log \frac{\sum_{t_i \in D} n_d(t_i)}{n_d(q)} \right) \quad (3)$$

where: TF_q is the number of occurrences of the query term ***q*** in ***d,*** $|d|$ is the document length measured in terms and $n_d(t)$ is the number of documents containing term t.

The term frequency based weighting factor is responsible for the correct ordering of documents with respect to a single query term. Normalization applied to the term frequency in Formula 3 prevents longer documents from being highly ranked. IDF weighting factor is responsible for making relevance scores of different terms comparable in case of multi-term queries. IDF calculation requires knowledge of collection statistics, such as total number of documents as well as the number of documents containing particular query term. As the global index contains posting elements with different access rights, revealing such global IDF in the relevance score leaks critical statistical data about inaccessible documents [6].

In ZERBER^{+R} we focus on confidential top-k query processing for single-term queries. In this case IDF factor is constant and relevance score calculation can be calculated as:

$$rscore(q,d) = \frac{TF_q}{|d|} \quad (4)$$

Results of a single-term query can be accurately ranked based only on the information contained in a single document using Formula 4. Processing of multi-term queries can then be performed by executing a number of single-term queries. By this procedure, the retrieval accuracy of a multi-term query slightly decreases representing a tradeoff between confidentiality of the collection statistics and retrieval accuracy of the index [21].

2.4 TF-Distribution

Confidential document ranking is a challenging task. On the one hand, encrypting relevance scores to protect them on an untrusted index server does not allow the server to perform any ranking. On the other hand, if accessible to the index server, the term frequency information required for the computation of relevance scores is term specific, and therefore can reveal the actual term in an index even if the term itself is encrypted, hence breaking confidentiality guarantees provided by the index.

Term frequency distribution among the documents in a collection follows a power law distribution (as shown by a log-log plot). Figure 4 shows the term frequency distributions for the frequent term (in German language) "nicht" and the less frequent term "management" in the test collection described in Section 5.1.1.

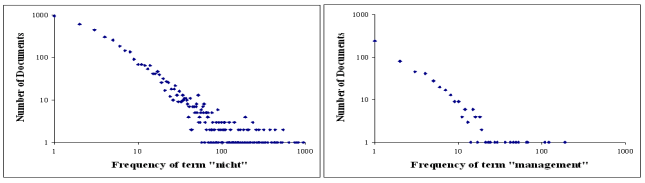

Figure 4: Log-Log Plot of TF Distributions

The terms can be differentiated by the slope and value range of the TF distribution. State-of-the–art IR techniques mostly use term frequency normalized by the document length [18] in order to avoid that large documents are ranked higher simply because they contain more terms.

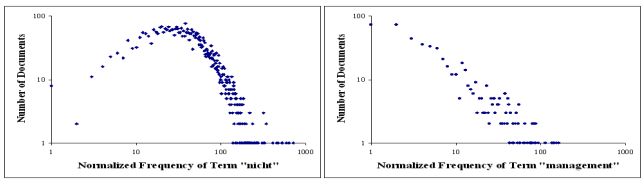

Figure 5: Log-Log Plot of Normalized TF Distributions

Normalized TF represents the fraction of the term in a document, which depends on the document topic, assigning higher ranks to the topic specific terms. Normalized TF distributions, as extracted from our study on our test collections (Figure 5 illustrates an example for the aforementioned terms), are not power law but still term specific, that is, distinguishable and therefore they may reveal their corresponding term.

Since most state-of-the–art IR techniques use normalized TF for ranking [18] our aim is to store the relevance scores accessible to the untrusted server while hiding the term specificity of the normalized TF distribution, therefore making posting elements representing different terms indistinguishable in a merged posting list. At the same time we need to preserve the relevance order of posting elements in a merged list to allow for top-k retrieval.

3. PROBLEM STATEMENT

We target the problem of providing confidential top-k retrieval from an outsourced inverted index. In the ideal case the server can order posting elements regarding to their relevance without violating the confidentiality of the index.

In an r-confidential index the probability of a particular element in the posting list of being related to a specific term is r-bounded. All posting elements are placed randomly inside the posting list and therefore their positions do not reveal any additional information to the adversary. A *scoring function* will order the terms according to their relevance and may increase the probability of a specific term being within particular positions or intervals of the posting list, given the adversary has knowledge of specific TF distribution as well as the public scoring function.

The ideal indexing scheme restricts an adversary's ability to increase her available knowledge, even if she takes over an index server and can examine the ranking information of posting elements and the stream of incoming queries and updates. Assuming posting elements contain relevance scores and they are accessible by the index server, then the elements can be ordered (sorted) by the server in order to retrieve top-k results for a given query. In the following, we refer to a merged index whose posting elements contain relevance score values as an *ordered index*. An *ordered posting list* is a merged posting list in an *ordered index*.

The *ordered index* offers maximal protection in case relevance score values does not reveal any additional knowledge about the index content.The ideal ordered index will be unattainable in practice, but we can identify the factors which have impact on the confidentiality of an *ordered index* and quantify the degree of their impact.

3.1 Threat Model

To execute a keyword query, the user first authenticates herself to an index server and supplies the query terms to the server as well as *k* - the desired number of the top-k documents. The index server determines user's access rights, identifies posting list containing query terms and returns the highest ranked elements from the requested list.

In order to enable the server to identify the top-k elements relevance scores of each posting element must be visible to the index server. This additional knowledge can compromise confidentiality of the index. Specifically, we want to bound the ability of an adversary to perform the following attacks:

1) *Identify terms represented by the posting elements by analyzing relevance score values stored in the index.*

An adversary Alice could use relevance score distribution statistics to extract specific features like score ranges, or score distribution patterns for each particular term. Alice could compare extracted features with the relevance score distribution in the posting lists to find correlations. If the terms are encrypted Alice could use these statistics to break the encryption. In a merged index, in case a simple term frequency based scoring function is used, she could claim that "frequent terms are more probably located in the head of the merged posting list" and even undo the posting list merging.

2) *Determine query terms of other users by observing queries and query results.*

In case of a merged ordered posting list, the number of requests required for obtaining top-k elements for a rare or a frequent term may differ. Alice can also know the k-value which is requested by the client application. As document frequency is term specific, Alice could guess the term by observing the number of follow-up requests required to fill the top-k results.

To guard against these attacks ZERBER^{+R} proposes novel techniques making relevance scores and number of follow-up requests for different terms indistinguishable for the server while preserving retrieval accuracy of server-side top-k processing.

3.2 Requirements

Server side top-k processing requires the server to access relevance scores of each posting element. However, as explained before, the distribution of posting elements within a merged posting list according to their plain relevance scores can reveal the corresponding term of the posting element (as shown on Figure 3). Since the scoring function presented in Formula 4 is monotonic, it is possible to make transformations to such function without affecting the ranking process as long as the ordering within posting elements representing each particular term is preserved. This section introduces the requirements for building a *relevance score transformation function (**RSTF** hereafter)*, which makes relevance score distributions of different terms indistinguishable. Specifically, it uniformly distributes posting elements representing different terms within the merged posting list while the order of posting elements related to each single term remains unaffected. This transformation function assigns a *transformed relevance score* (***TRS*** for short) to each posting element, therefore replacing the original relevance score. To preserve confidentiality of the index a posting element related to any term ***t*** in an ordered posting list should have equal probability to obtain a given ***TRS*** score.

RSTF has to fulfil the following properties:

- ***RSTF*** maps the relevance scores of different terms to a range ***R*,** which will be the same for all ***RSTF***s.

- ***RSTF*** uniformly distributes the ***TRS*** values over ***R.***
- ***RSTF*** preserves the order of the relevance score values.

In the next section we propose a heuristic for the construction a ***RSTF*** for arbitrary term independent of its score distribution.

4. ZERBER^{+R} DESIGN

Processing in ZERBER^{+R} can be split in two phases: an offline pre-computing phase performed once at the time of index initialization and an online insertion and query phase. In the pre-computing phase, ZERBER^{+R} initializes and publishes the ***RSTF*** for each term in the training document set, such that in the online insertion phase this function can be used by an inserting client. To index a document, its owner extracts the document's terms, builds their elements, encrypts them, calculates ***TRS*** values, and sends encrypted posting elements to the server along with the IDs of the merged posting list that the new element belongs to, the document's group and the ***TRS*** value. The index server authenticates the user, checks his group membership and accepts the update if appropriate. Finally, the server inserts posting elements into the specified merged posting list. Upon query the server has access to the ***TRS*** values and can identify the top-k most relevant query answers.

4.1 *RSTF* Construction

We define the target ***RSTF*** range to map the relevance score input values (calculated using Formula 4) as $\boldsymbol{R}=[a_1, a_2]$.

In order to explain our approach, we first consider the case where the input values are already uniformly distributed over some range $[b_1, b_2]$. In this case ***RSTF*** will be a straight line with a slope $(b_1-b_2)/(a_1-a_2)$ and an offset $-b_1$, and is a projection of $[b_1,b_2]$ on ***R***.

For example f(x) = 2.5*x-1.25 will map a range [0.5, 0.9] to [0, 1]

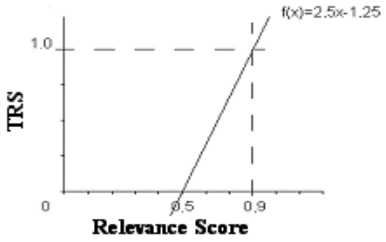

Figure 6: Relevance Score Transformation: Linear Projection

The slope of the projection function is responsible for the scale of the input range in the output range. Given an input range, the greater the slope of the projection function, the wider is the output range. Unfortunately, the relevance score distribution in the document set is not uniform and a linear projection does not change the distribution of the input relevance score values. To uniformly distribute the input values over the output range, the slope of the ***RSTF*** at each particular point has to reflect the probability density of the relevance score values at this point, i.e. it should create a wider output range in the more crowded areas. Thus the probability density function of the relevance score values is a derivative of the ***RSTF*** and respectively the ***RSTF*** is an integral over the probability density function of the relevance score values.

The values of the integral over a probability density function are always contained within the common range [0, 1]. Moreover, an integral is monotonically increasing, which preserves the ordering of the input values. Thus an ***RSTF*** created in this way would possess the first and third required properties discussed above.

In the following we model the probability density of the relevance score distributions and compute a ***RSTF*** that approximates the relevance score distribution to a uniform distribution.

4.1.1 Modelling Relevance Score Distribution

To model the relevance score distribution, ZERBER^{+R} requires as input a training set of documents. This set must be a representative sample of the corpus, such that the distribution of the relevance scores will hold for the whole corpus as well. From the training set ZERBER^{+R} extracts the relevance scores for each term-document pair. Terms found later, which were not contained in the training set are assumed to be rare and can therefore be assigned a random ***TRS***. We base our model on the central limit theorem [7] stating that the sampling distribution of the sample mean is approximately normal, even if the distribution of the population from which the sample is taken is not normal.

We consider each relevance score value from the training set to be a sample mean of the relevance score. The continuous probability density function of the *normal distribution* is the Gaussian function. We model the probability distribution of the relevance score values around each sample mean as a Gaussian curve. We take the sum of the Gaussian curves over all samples as an approximation of the relevance score distribution for a given term over the whole document corpus. Thereby we make use of the fact that a probability density function can be arbitrarily closely approximated by a weighted sum of Gaussian curves [1].

A Gaussian function can be defined by two parameters, *location* and *scale*: the mean ("average", μ) and variance (standard deviation squared) $\boldsymbol{\sigma}^2$, respectively. The probability density of a term ***t*** over the whole document corpus given ***N*** training points is calculated as:

$$f_{\vec{\mu},\sigma}(x) = \frac{1}{N}\sum_{i=0}^{N}\left(\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu_i)^2}{2\sigma^2}}\right) \qquad (5)$$

where μ_i – is the i^{th} value from the training set and $\boldsymbol{\sigma}$ – is the scale of the Gaussian function.

A more frequent term results in several training values. In this case the sum of the Gaussian bells, one for each input value, will reflect the probability distribution over the input interval. The probability of unseen values being in particular region is reflected through the density of training points in that region.

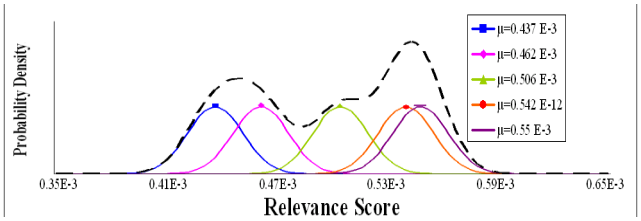

Figure 7: Probability Distribution from 5 Training Values

Figure 7 shows the sum of the probability density functions over five input values. The X-axis shows the relevance score of a training value. The Y-axis shows the probability density. Solid lines represent probability density of each training value. The

dashed line represents the probability density accumulated using several training values.

4.1.2 Calculating the *RSTF*

RSTF(x) is an integral of the probability distribution of the input training values within the range $[-\infty, x]$.

Given ***N*** training points ***RSTF***(x) can be calculated as:

$$RSTF_{\vec{\mu},\sigma}(x) = \sum_{i=0}^{N} \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{(u-\mu_i)^2}{2\sigma^2}} du \qquad (6)$$

where x is a relevance score to be transformed, μ_i is the i^{th} value from the training set, N is a number of the values in the training set and σ is the scale of the Gaussian function.

Therefore, ***RSTF***(x) represents a projection of the relevance score distribution that approximates the relevance score distribution into a uniform distribution over the range [0, 1]. An integral of the Gaussian function within the range $[-\infty, x]$ can be estimated with the standard error function:

$$\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{(u-\mu)^2}{2\sigma^2}} du \approx \frac{1}{1+e^{-\sigma(x+\mu)}} \qquad (7)$$

Thus the ***RSTF*** can be calculated as:

$$RSTF_{\vec{\mu},\sigma}(x) \approx \frac{1}{N} \sum_{i=0}^{N} \left[\frac{1}{(1+e^{-\sigma(x+\mu_i)})} \right] \qquad (8)$$

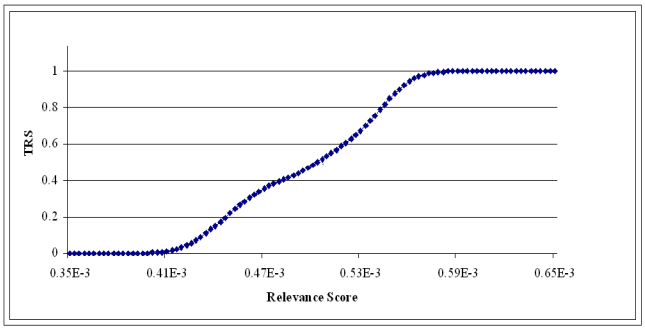

Figure 8: An Example *RSTF* for a Term

Using training set from our data sets (see Section 7.1), Figure 8 illustrates an example ***RSTF*** function for the German term "Vergütung" (reimbursement). The X-axis shows the input relevance score, the Y-axis illustrates its output ***TRS*** value computed using Formula 8.

4.1.3 σ Selection

In order to ensure the uniformness of the ***TRS*** distribution on the whole corpus it is needed to ensure the correct prediction of the relevance score distribution during the ***RSTF*** computation. Correctness of the prediction depends on the representativeness of the training set as well as on the correct σ selection.

The σ parameter represents the scale of the Gaussian function reflecting the generality of the ***RSTF***. σ is responsible for the learning/memorizing effect. Smaller σ means a broader Gaussian bell – and thus a more general prediction. Higher σ value means a narrower bell, meaning a less general function which represents the particular training point (also known as overfitting).

To select an optimal σ value we use the cross-validation technique to measure the uniformness of the ***TRS*** distribution in a control set. As a basic measurement for uniformness we compute the variance in the distribution of the ***TRS*** values of a particular term in the control set with respect to a uniform distribution, that is, how far the ***TRS*** distribution is from a uniform distribution.

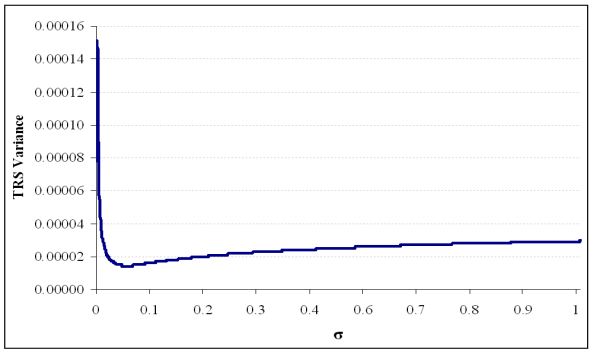

Figure 9: *TRS* Variance Depending on the Selected σ

Figure 9 presents the ***TRS*** variance in the control set dependent on the σ value. The X-axis shows the σ value, the Y-axis shows the variance within ***TRS*** values in the control set. At first, the ***TRS*** values are distributed more uniformly with an increasing σ. However, after reaching the minimum (an optimal σ), the overfitting effect appears and the uniformness is destroyed. An optimal σ for a particular term is the infimum of the variance function. As Figure 9 depicts, a good selection of σ provides a variance of smaller than 0.00002 (standard deviation of 0.0044, that is, 0.44% of the range [0, 1]).

The process of cross-validation is time consuming. Finding a method for directly determining an optimal σ value is an interesting direction for future research.

4.2 Query Answering using ZERBER+R

To execute a keyword query, the user first authenticates herself to an index server and tells the server which posting list she wants to query as well as the k-value. The index server determines user's access rights and returns a number of highly ranked elements from the requested list. The client decrypts posting elements and filters out elements for terms not queried. If the client did not obtain the desired number of elements belonging to the queried term it sends a follow-up request to the server. Dependent on the response size, the number of follow-up requests for rare and frequent terms can differ, leaking information to the adversary. In the following we discuss heuristics reducing the number of follow-up requests.

Suppose that all the posting lists are merged into M lists $L_1,\ldots,L_M$.

The total workload cost ***Q*** for a set of queries for retrieving the top-***k*** elements can be calculated as:

$$Q \cong \sum_{L_i \in M} \left[N(L_i) \times \left(\sum_{j \in L_i} q_j \right) \right], \qquad (9)$$

where q_j is the query frequency of term j, and $N(L_j)$ is the number of elements to be retrieved from the merged posting list L_i.

Posting elements in each merged list are sorted based on their ***TRS***. Following the uniform distribution, the first position pos_1 of the term t in the list can be approximated as:

$$pos_1(t) \le \frac{1}{p_t} = \frac{\sum_{t_i \in L} n_d(t_i)}{n_d(t)} \quad (10)$$

where p_t is the probability of term t and $n_d(t)$ is the document frequency of the term t in a merged list L. In order to obtain the top-k elements of the term t the total number N of elements to be retrieved from the list L is:

$$N(L) = k \cdot pos_1(t) = \frac{k}{p_t} = k\left(\frac{\sum_{t_i \in L} n_d(t_i)}{n_d(t)}\right) \quad (11)$$

where $n_d(t)$ is the document frequency of the term t in a merged list L. In order to retrieve the top-k elements from the merged list without knowing which particular term is queried, the number of retrieved elements should be sufficiently large to include the top-k elements for all merged terms in the posting list L.

However, this is impractical in case a posting list contains very rare terms. For instance, for a list with terms for which $n_d(t)$=1 (term t is contained only in one document), the whole posting list will be returned in response to the query. Thus we need a heuristic to determine the query response size that reduces the total workload cost. This heuristic should minimize the used bandwidth while including the top-k answers in the first response to most of the received queries.

From query load logs described in Section 7.1.3 we know that the most frequent queries constitute nearly the whole query workload (Figure 10). Thus to reduce the total workload cost, the query answering heuristic should provide high efficiency for the most frequent queries. Due to confidentiality concerns we can not use query frequency directly. However, document frequencies and query frequencies and correlated, though some frequent terms are rarely queried (e.g., "although") [16]. To provide efficient query answering for the most frequent queries while reducing the bandwidth, ZERBER^{+R} puts a bound b on the initial response size, such that only sufficiently frequent terms with probability p_t*b>=1 are necessarily returned inside of the first query response. We discuss selection of the initial response size in Section 7.4.

In case a rare term t with probability p_t*b<1 is requested the user sending the query might need to issue several follow-up requests to obtain the top-k results. This can give an adversary the possibility to infer a rare term has been requested, therefore allowing her to distinguish between two merged terms in case one is frequent and the other rare. In order to prevent this situation, ZERBER^{+R} makes use of the BFM index (using the Breadth First Merging of posting lists) described in [22], which ensures that the terms merged in a posting list have similar frequency distributions. Therefore even if retrieving top-k results from a merged list containing rare terms could require a number of follow-up requests, this number will be similar for all terms contained in the list, avoiding that an adversary is able to infer any additional information. An adversary could also try to estimate the position of a queried term in the posting list based on the querying behaviour of the user. ZERBER^{+R} reduces the information leakage in this case by progressively increasing response size for follow-up requests. Assuming that the user can process at least the amount of data she already obtained ZERBER^{+R} doubles response size for each follow-up request until the user is satisfied with the result or obtains the whole list.

4.3 Summary

ZERBER^{+R} allows for r-confidential server-side top-k retrieval as well as insert in presence of adversary. ***RSTF*** proposed in this work is monotonic, such that it does not affect accuracy of the single-term top-k query. However, as score calculation does not include IDF factor, accuracy of multi-term queries can be slightly affected. This effect was considered in the literature [21].

Confidentiality guarantees provided by ZERBER^{+R} depend on the ability of the ***RSTF*** to uniformly distribute the relevance score values among the given range. We proposed an experimental measure to quantify the uniformness of the distribution.

5. EVALUATION

This section evaluates ZERBER^{+R} in terms of security guarantees, resource usage and efficiency in query answering.

5.1 Experimental Setup

This subsection provides details about our documents and workload. We used two data sets, from the Stud IP Learning Management System [20] and the Open Directory Project crawl data [17]. For each of our document collections we created an index containing 32K merged posting lists. We use a web search engine query log as the workload. All experiments ran on a 2-processor 2.0 GHz Intel CPU T2500 with 2 GB RAM.

5.1.1 Stud IP Data

The Stud IP Learning Management System [20] allows sharing of access-controlled materials within groups of students and teachers and is used by several universities in Germany. The Stud IP installation we use for our experiments has over 3,300 courses and 6,000 registered students. A mid-semester snapshot used for our experiments contained 8,500 documents with 570,000 terms.

5.1.2 Web Data

We used a collection from the Open Directory Project [17] (a human edited directory of the Web) crawled in 2005, with 237,000 documents and 987,700 distinct terms. The crawler's strategy was to find pages on a variety of topics [14], such that 100 topics were randomly selected; we used the set of documents on one topic as the set of documents of one group. To obtain a representative sample for the ***RSTF*** initialization we randomly selected 30% of the documents from each data set as a training set. We randomly chosen about one third from the initial sample for the control set and used the rest as training data and minimized variance among the ***TRS*** values using cross-validation technique.

5.1.3 Web Search Engine Query Log

Our query log has 7 million queries containing 2.4 terms on average and 135,000 distinct query terms. ZERBER^{+R} considers a multi-term query as a sequence of single-term queries. Therefore for our experiments we considered each query term separately. Figure 10 shows the correlation of the query frequency and the corresponding cumulative query workload for retrieving top-10 results (computed using Formula 9).

The log-scale X-axis shows the query terms in decreasing order of frequency (from most to least popular).

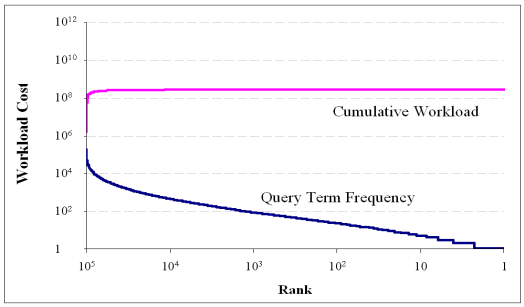

Figure 10: Cumulative Query Workload Cost

The most frequent queries constitute nearly the whole query workload. Thus to reduce the total workload cost, the system should provide high efficiency for the most frequent queries.

5.2 Security Guarantees

If an adversary Alice compromises an index server, she can attempt to amplify her knowledge in many ways. For example Alice can now examine the unencrypted ***TRS*** values attached to the posting elements in an index but ZERBER^{+R} should ensure that she can not learn anything in order to preserve our concept of r-confidentiality. To achieve maximal security effectiveness the ***RSTF*** needs to distribute the relevance scores of each term from the real dataset equally well as it is achieved in the training set values. Otherwise, term specific distribution patterns would be introduced allowing Alice binding posting elements within specific posting list areas to their corresponding terms with higher probability. In case the document training set is a representative sample of the corpus and σ value is selected properly, all terms will have equal probability to obtain a given ***TRS*** value, such that using ***TRS*** does not introduce any additional attack possibilities. Alice can also observe user queries and query responses. As discussed in Section 6.3 it is impractical to include top-k results into the initial query response for all possible terms[1]. Thus for rare terms which top-k elements are not contained in the initial response Alice could observe an increased number of follow-up requests and conclude that a rare term has been requested. However, as a Zerber BFM index contains terms of similar probability inside of a posting list, the number of requests observed by Alice will not differ for the terms contained in one merged list, such that ***r***-confidentiality of the index will not be affected. Alice could also try to estimate a position of such a rare (unknown) term based on the number of returned posting elements. Thereby growing response size for follow-up requests will reduce the total number of such requests and introduce an increasing degree of uncertainty in Alice's claims regarding the position of the (unknown) rare queried term.

5.3 Storage Overhead

To allow for top-k processing an ordinary inverted index typically contains relevance score information attached to each posting element..ZERBER^{+R} attaches a transformed relevance score ***TRS*** to each posting element, which is sufficient for effective posting element ranking on the server side. Thus it does not introduce any storage overhead compared with an ordinary inverted index.

5.4 Selection of the Initial Response Size

As discussed in Section 6.3 ZERBER^{+R} increases its response size progressively depending on the number of follow-up requests to a query. The initial response size should be selected in a way to minimize the number of follow-up requests and the bandwidth overhead for the majority of the queries in the workload.

We denote the number of posting elements in the first response as initial response size ***b*** and the accumulated number of posting elements in a sequence containing ***n*** follow-up requests as a total response size ***TRes***. Given the number of follow-up requests, the total response size can be calculated as:

$$T\,\mathrm{Re}\,s = b \cdot \sum_{i=0}^{n} 2^i \qquad (12)$$

Figure 11 presents an average bandwidth overhead ***AvBO*** over the set of queries ***Q*** in the query workload calculated as the ratio between the total response size ***TRes*** of ZERBER^{+R} required in order to obtain the top-k elements and ***k*** elements returned by an ordinary inverted index in response to a top-***k*** query:

$$AvBO = \frac{\sum_{q \in Q} \left(T\,\mathrm{Re}\,s(q) / k \right)}{|Q|} \qquad (13)$$

The X-axis of Figure 11 shows the number of posting elements in the initial query response. The Y-axis shows an average bandwidth overhead in both test collections calculated using Formula 13 for ***k***=1, 10 and 50.

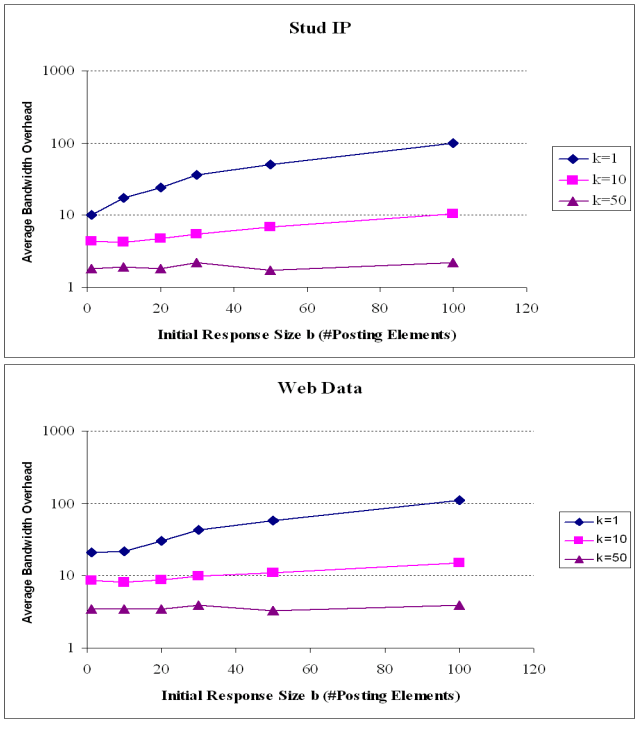

Figure 11: Average Bandwidth Overhead

[1] In this paper we focus on a fix size of results in an initial response to a query. However, we leave for further work optimizations where this size could vary depending on the frequency of the terms of each merged posting list.

Figure 11 shows that the minimal bandwidth overhead for a top-***k*** query in ZERBER^{+R} can be achieved with b=k, i.e. by returning around ***k*** elements. Further enlargement of the initial response size leads to an increased bandwidth overhead.

Figure 12 shows an average number of requests required in order to obtain the top-***k*** elements dependent on the number of elements in the initial response for ***k***=1, 10 and 50 in both test collections. The X-axis shows the number of elements in the initial response. The Y-axis shows an average number of requests required to obtain the top-k elements for the queries in the workload.

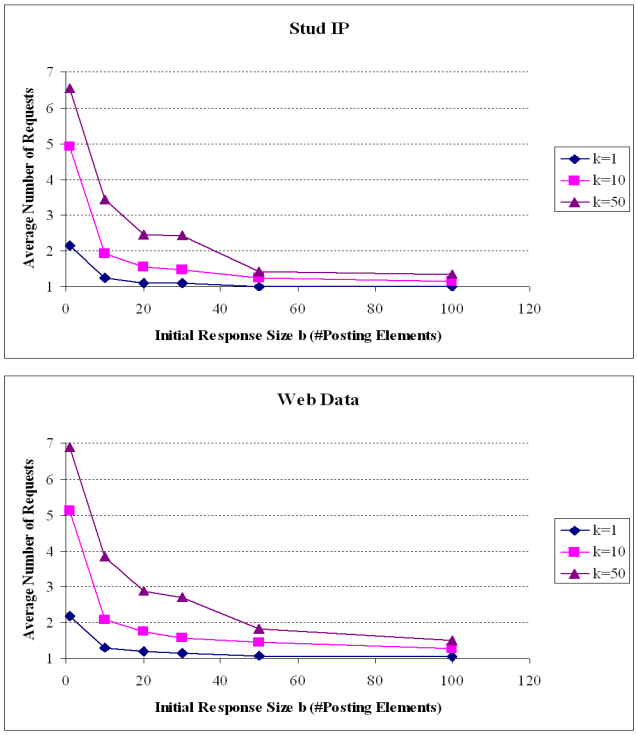

Figure 12: Average Number of Requests

Figure 12 also illustrates that with an initial response size of approximately 10 elements most of the query terms return the top-10 results within 2 requests (returning 30 posting elements in total). In order to further reduce the number of requests, the initial response size needs to be significantly increased. However this is not desirable because of the significant increase in the bandwidth overhead. Thus for our further experiments we selected ***k*** as the preferable initial response size for a top-***k*** query.

5.5 Query Performance

We calculated the efficiency in query answering $QRatio_{eff}$ introduced by different sizes of the initial response as the ratio between ***k*** and the total ZERBER^{+R} response size:

$$QRatio_{eff} = k / T\,Res \qquad (14)$$

Figure 13 plots the efficiency in query answering $QRatio_{eff}$ for the top-***k*** request with ***k***=10 and the initial response size b=10, 20 and 50 elements in the both test collections. In this figure, the Y-axis shows $QRatio_{eff}$ and the X-axis represents the query terms in the workload (in %), ordered by $QRatio_{eff}$.

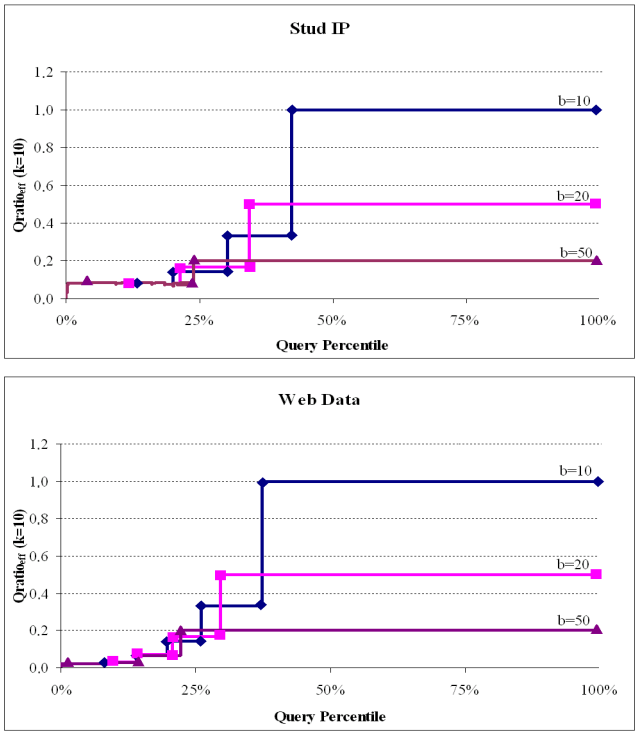

Figure 13: Efficiency in Query Answering

The best query efficiency distribution for the top-10 request in the both test collections is attained using the initial response size ***b***=10. In this case around 60% of the longest running queries in the workload have an efficiency value $QRatio_{eff}$=1and the next 20% longest-running queries have $QRatio_{eff}$=0.2 on average. The shortest running 20% of the queries have average $QRatio_{eff}$=0.1. Increasing the initial response size to 20 elements leads to the significant reduction of $QRatio_{eff}$ for all longest running queries. In this case around 70% of the longest running queries in the workload have an efficiency value $QRatio_{eff}$=0.5 and the shortest running 30% of the queries have average $QRatio_{eff}$=0.1.

The simulation results described above have shown that initial response size of ***b***=10 offers very reasonable query performance for a top-10 query. Our experiments show that using ZERBER^{+R} the query workload cost ratio can be kept comparable to a conventional inverted index for 60% of the queries, while preserving the ***r***-confidentiality of the index.

5.6 Network Bandwidth

An initial response of ZERBER^{+R} contains ***b*** elements and the total response size including follow-up requests required to obtain top-***k*** elements is calculated using Formula 12. For our calculations, we assume the following intranet setup: users connect over a mobile device with a 56 Kb/s modem, while servers use 100 Mb/s LAN connections. We use initial response size ***b, k***=10. The document snippets arrive in XML format.

We use a real-world query workload and the Open Directory Project (ODP) data described in Section 7.1.2. For our experiments we assume that the user has access to all documents

in the ODP data collection. In this workload, about 85 posting elements are returned from the ODP index per query term on average. Assuming that each posting element is encoded using 64 bits, this is approximately 5.3 Kb (0.7 KB) per query term response. The queries in the workload contain 2.4 terms on average, which allows a server for the execution of about 750 queries/second. On average, each snippet contains about 250 B including XML formatting, which yields 2.5 KB for the top-10 snippets. Thus average total response size for the top-10 results is 3.5 KB. In comparison, Google's response for the top-10 results is about 15 KB, including the snippets as well as information used for presentation purposes (HTML, CSS, etc.). Altavista returns 37 KB and Yahoo returns 59 KB of top-10 results. As ZERBER^{+R} posting elements are encrypted, query respose is represented by a random bit string and standard HTML compression is ineffective. The compressed responses of Google, Altavista and Yahoo are comparable to ZERBER^{+R} responses. Further optimization can be achieved by adding search result checksums and caching them on the client (defined in HTTP 1.0).

6. RELATED WORK

In the literature different ways of protecting outsourced shared information were proposed. Encryption is a standard technique for storing data confidentially [10, 13]. Ways to search encrypted text or tables stored on remote untrusted servers we proposed in [5, 8, 11, 19]. [4, 15] provide a framework for policy-based protection of XML data. Other techniques include suppressing and/or generalizing data into less specific forms, so that they no longer uniquely represent individuals [9, 12]; k-anonymity is one popular form of generalization (e.g., [3]).

Unfortunately, encryption of posting elements does not hide critical statistical data which can be used by an adversary to reverse-engineer the terms [6]. Probabilistic index protection techniques suppress statistical data by introducing a controlled amount of uncertainty. For instance, the μ-Serv system developed by IBM inserts false positive posting elements in the index [2]. The lack of precision in search results represents a tradeoff between search efficiency and confidentiality preservation.

Zerber [22] combines the benefits of both probabilistic and encryption techniques. It allows obtaining precise search results from an outsourced encrypted inverted index while providing confidentiality guarantees for the indexed documents. Zerber provides tunable resistance to statistical attacks by supplementing encryption with a novel probabilistic term merging scheme.

Unfortunately, solutions discussed above either do not allow for top-k retrieval from an outsourced inverted index, or do not consider ranking information as sensitive. ZERBER^{+R} fills this gap by offering a ranking model which allows attaching relevance score information to the encrypted posting elements without introducing any information leakage from an outsourced index.

There has been a considerable amount of work on top-k retrieval of plain text documents [18]. Thereby an index server makes use of the relevance score information attached to each posting element to return the top-k documents most relevant to a user query. These relevance scores are calculated based on the term frequency information as well as on the collection statistics. However, term frequency is term specific and thus, if stored in plain text, can allow an adversary to discover the terms it belongs to. To prevent statistical attacks ZERBER^{+R} makes relevance scores related to different terms indistinguishable, while preserving ordering of posting elements for the top-k processing.

The idea of uniformly distributing posting elements using an order preserving cryptographic function was first discussed in [21]. However, uniform distribution of posting elements alone does not hide the document frequency and thus allows an adversary to recover encrypted terms. Moreover, the order preserving mapping function proposed in [21] currently does not support efficient index inserts and updates such that, at least in some cases, the posting list has to be completely rebuilt. On the contrary, ZERBER^{+R} is based on an ***r***-confidential inverted index which protects document frequency information. Moreover, the ***RSTF*** of ZERBER^{+R} does not introduce any overhead for inserts and updates compared with an ordinary inverted index. This function is created only once at the index initialization time and allows for unlimited index update and insert operations.

7. CONCLUSION & FUTURE WORK

Privacy-preserving document sharing among collaboration groups in an enterprise requires techniques allowing for centralized exchange of access-controlled information through largely untrusted servers. Such system needs to provide confidentiality guarantees for shared information while offering IR properties comparable to the ordinary search engines. In this paper we presented ZERBER^{+R}, a ranking model which allows top-k retrieval from a confidential outsourced inverted index. ZERBER^{+R} creates a relevance score transformation function which makes relevance scores of different terms indistinguishable, in a way that even if they are known to an adversary they do not reveal any information about the indexed data. This enables the server to provide the top-k results most relevant to a user query. Our experiments on two real-world data sets show that ZERBER^{+R} makes economical usage of bandwidth and offers information retrieval properties comparable with an ordinary inverted index while preserving the confidentiality of the indexed data.

In this paper multi-term queries are viewed as a sequence of single-term queries. Confidential ranking of the multi-term queries is an interesting direction for future research.

8. REFERENCES

[1] Alspach, D. and Sorenson, H. Nonlinear Bayesian Estimation Using Gaussian Sum Approximations. IEEE Transactions on Automatic Control, Vol. 17, No.4, p. 439 – 448, Aug., 1972.

[2] Bawa, M., Bayardo, Jr. R. J. and Agrawal, R. Privacy-preserving indexing of documents on the network. In Proceedings of the VLDB, 2003.

[3] Bayardo, R. and Agrawal, R. Data privacy through optimal k-anonymization. In Proceedings of ICDE, 2005.

[4] Bertino, E., Castano, S. and Ferrari, E. Securing XML documents with Author-X. In IEEE Internet Computing, May/June 2001.

[5] Boneh, D., Crescenzo, G. D., Ostrovsky, R., and Persiano, G., Public-key encryption with keyword search. In Proceedings of Eurocrypt 2004.

[6] Büttcher, S. and Clarke, C. L.A. A Security Model for Full-Text File System Search in Multi-User Environments. In Proceedings of the FAST, 2005.

[7] Central limit theorem. Wikipedia. Retrieved in March 2008. http://en.wikipedia.org/wiki/Central_limit_theorem

[8] Chang, Y.-C. and Mitzenmacher, M. Privacy preserving keyword searches on remote encrypted data. Cryptology ePrint Archive, Report 2004/051, Feb 2004.

[9] Fung, B. C. M., Wang, K. and Yu, P. S. Top-down specialization for information and privacy preservation. In Proceedings of ICDE 2005.

[10] Goh, E., Shacham, H., Modadugu, N. and Boneh, D. Sirius: Securing remote untrusted storage. In NDSS, 2003.

[11] Hacigumus, H., Iyer, B. R., Li, C. and Mehrotra, S. Executing SQL over encrypted data in the database-service-provider model. In Proceedings of the SIGMOD, 2002.

[12] Iyengar, V. Transforming data to satisfy privacy constraints. In Proceedings of the SIGKDD, 2002.

[13] Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q. and Fu, K. Plutus: scalable secure file sharing on untrusted storage. In Proceedings of the FAST, 2003.

[14] Kohlschütter, C., Chirita, P.-A. and Nejdl W. Using Link Analysis to Identify Aspects in Faceted Web Search. SIGIR'2006 Faceted Search Workshop, 2006, Seattle, WA.

[15] Miklau, G. and Suciu, D. Controlling Access to Published Data Using Cryptography. In Proc. of the VLDB 2003.

[16] Mitra, S., Hsu, W. W. and Winslett, M. Trustworthy keyword search for regulatory-compliant records retention, In Proceedings of VLDB, 2006, Seoul, Korea, 1001-1012.

[17] Open Directory Project: http://www.dmoz.org/

[18] Singhal, A. Modern Information Retrieval: A Brief Overview. In IEEE, Data Eng. Bull. 24(4), 2001

[19] Song, D. X., Wagner, D., Perrig, A. Practical Techniques for Searches on Encrypted Data. In Proceedings of IEEE Security and Privacy Symposium, May 2000, 44-55.

[20] Stud IP LMS. Available at: http://www.studip.de/.

[21] Swaminathan, A., Mao, Y., Su, G.-M., Gou, H., Varna, A. L., He, S., Wu,M., Oard, D. W. Confidentiality-preserving rank-ordered search. In Proc. of StorageSS '07 Workshop.

[22] Zerr, S., Demidova, E., Olmedilla, D., Nejdl, W., Winslett M., Mitra, S. Zerber: r-Confidential Indexing for Distributed Documents. In Proceedings of the EDBT 2008.