Authenticating the Query Results of Text Search Engines

HweeHwa Pang
School of Information Systems
Singapore Management University
hhpang@smu.edu.sg

Kyriakos Mouratidis
School of Information Systems
Singapore Management University
kyriakos@smu.edu.sg

ABSTRACT

The number of successful attacks on the Internet shows that it is very difficult to guarantee the security of online search engines. A breached server that is not detected in time may return incorrect results to the users. To prevent that, we introduce a methodology for generating an integrity proof for each search result. Our solution is targeted at search engines that perform similarity-based document retrieval, and utilize an inverted list implementation (as most search engines do). We formulate the properties that define a correct result, map the task of processing a text search query to adaptations of existing threshold-based algorithms, and devise an authentication scheme for checking the validity of a result. Finally, we confirm the efficiency and practicality of our solution through an empirical evaluation with real documents and benchmark queries.

1. INTRODUCTION

Professional users commonly require certain security provisions from their paid content services. This is particularly so in the financial and legal industries. One security provision is *integrity assurance* [23] – that the content and search results received are correct, and have not been tampered with. For example, a patent examiner using MicroPatent's Web portal would expect from it the same search results as the up-to-date CD-ROM version.

Naturally, the administrator of a search engine would employ a combination of security safeguards, such as firewalls and intrusion detection. Notwithstanding that, servers that are situated in a seemingly well-guarded network can often still be infiltrated through a multi-step intrusion, in which each step paves the way for the next attack [30]. Indeed, the increasing number of successful attacks on online servers over the past decade demonstrates that it is very difficult to guarantee the security of all the servers over extended periods of time. In the event that a search engine is compromised, it could return tampered results:

- *Incomplete results* that omit some legitimate documents. In the MicroPatent case, an attacker could cause his patents to drop out of the search results (to prevent competitors from discovering them), by tampering with the query, the index, or the similarity ranking function.
- *Altered ranking* that deviates from the correct similarity ranking. In the MicroPatent system, for instance, the attacker may divert the searcher's attention from certain patents, or tamper with the document ordering to bias the search results.
- *Spurious results* that include some fake documents. For example, a MicroPatent attacker may seek to discourage potential competitors by adding fake patents to their search results.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand

Existing studies on query result authentication have addressed the problem for relational databases and data streams (e.g. [21, 22]). The general approach is to build authentication information into index tree structures like the B^+-tree [13] and the R-tree [5]. In text retrieval, each term in the dictionary is considered a dimension. Since a realistic dictionary typically contains more than 100,000 terms, the dimensionality far exceeds the capabilities of single- and multi-dimensional index trees, thus ruling out the application of tree-based authentication schemes in text search engines.

The standard document organization/retrieval method in search engines is the inverted index [32]. In this paper, we propose a framework for inverted index authentication that can generate an integrity proof for any search result. Besides enabling the user to confirm the correctness of the result, the integrity proof can also be archived to construct an audit trail for any ensuing decision taken by the user. Our contributions in this paper are as follows:

- We formalize the problem of document search by similarity, and identify the properties that define a correct search result;
- Based on these properties, we introduce a novel authentication mechanism that maps the task of text query processing (on an inverted index) to adaptations of existing threshold-based algorithms, and returns integrity proofs to the users;
- To the best of our knowledge, this is the first authentication mechanism for the inverted index, and also the first for similarity-based text search engines;
- Our techniques are general and readily deployable as they do not interfere with the similarity ranking mechanism of existing search engines;
- We present extensive experiment results, using synthetic workloads as well as standard TREC data and queries [29], that confirm the robustness and practicality of our proposed authentication mechanism.

The rest of this paper is organized as follows. The next section covers background on text retrieval, cryptographic primitives that serve as building blocks for our authentication mechanism, and also related work. Section 3 presents the problem definition and introduces the query processing algorithms and our authentication mechanism. Experiment results are reported in Section 4. Finally, Section 5 concludes the paper and discusses future work.

term id	term t	f_t		Inverted List for t
1	and	1	$\mapsto$	$\langle 6, 0.159\rangle$
2	big	2	$\mapsto$	$\langle 2, 0.148\rangle$ $\langle 3, 0.088\rangle$
3	dark	1	$\mapsto$	$\langle 6, 0.079\rangle$
4	did	1	$\mapsto$	$\langle 4, 0.125\rangle$
5	gown	1	$\mapsto$	$\langle 2, 0.074\rangle$
6	had	1	$\mapsto$	$\langle 3, 0.088\rangle$
7	house	2	$\mapsto$	$\langle 3, 0.088\rangle$ $\langle 2, 0.074\rangle$
8	in	5	$\mapsto$	$\langle 6, 0.159\rangle$ $\langle 2, 0.148\rangle$ $\langle 5, 0.142\rangle$ $\langle 1, 0.058\rangle$ $\langle 7, 0.058\rangle$ $\langle 8, 0.053\rangle$ ...
9	keep	3	$\mapsto$	$\langle 5, 0.088\rangle$ $\langle 1, 0.088\rangle$ $\langle 3, 0.088\rangle$
10	keeper	3	$\mapsto$	$\langle 4, 0.125\rangle$ $\langle 5, 0.088\rangle$ $\langle 1, 0.088\rangle$
11	keeps	3	$\mapsto$	$\langle 5, 0.088\rangle$ $\langle 1, 0.088\rangle$ $\langle 6, 0.079\rangle$
12	light	1	$\mapsto$	$\langle 6, 0.079\rangle$
13	night	3	$\mapsto$	$\langle 5, 0.177\rangle$ $\langle 4, 0.125\rangle$ $\langle 1, 0.088\rangle$
14	old	4	$\mapsto$	$\langle 2, 0.148\rangle$ $\langle 4, 0.125\rangle$ $\langle 1, 0.088\rangle$ $\langle 3, 0.088\rangle$
15	sleeps	1	$\mapsto$	$\langle 6, 0.079\rangle$
16	the	6	$\mapsto$	$\langle 5, 0.265\rangle$ $\langle 3, 0.263\rangle$ $\langle 6, 0.200\rangle$ $\langle 1, 0.159\rangle$ $\langle 2, 0.148\rangle$ $\langle 4, 0.125\rangle$ ...

Figure 1: Example of Frequency-Ordered Inverted List

2. BACKGROUND

2.1 Similarity-Based Text Retrieval

Most text search engines rate the similarity of each document to a query (i.e., a set of keywords) based on these heuristics [32]: (a) terms that appear in many documents are given less weight; (b) terms that appear many times in a document are given more weight; and (c) documents that contain many terms are given less weight. The heuristics are encapsulated in a similarity function, which uses some composition of the following statistical values:

- $f_{d,t}$, the number of times that term t appears in document d;
- $f_{\mathcal{Q},t}$, the number of times that term t appears in query $\mathcal{Q}$;
- f_t, the number of documents that contain term t;
- n, the number of documents in the data set $\mathcal{D}$.

A similarity function that is effective in practice is the Okapi formulation, which defines the score of a document d with respect to a query $\mathcal{Q}$, $S(d|\mathcal{Q})$, to be:

$$S(d|\mathcal{Q}) = \sum_{t \in \mathcal{Q}} w_{\mathcal{Q},t} \times w_{d,t} \quad (1)$$

where

$$K_d = k_1 \left((1-b) + b \frac{W_d}{W_A} \right)$$

$$w_{d,t} = \frac{(k_1+1) f_{d,t}}{K_d + f_{d,t}}$$

$$w_{\mathcal{Q},t} = ln \left(\frac{n - f_t + 0.5}{f_t + 0.5} \right) \times f_{\mathcal{Q},t}$$

In the above formulation, k_1 and b are parameters with recommended settings of 1.2 and 0.75 respectively; while W_d and W_A are the document length and average document length. Intuitively, $w_{d,t}$ ($w_{\mathcal{Q},t}$) is the normalized frequency of term t in document d (in query Q, respectively) and represents its significance therein.

Given a query, a straightforward evaluation algorithm is to compute $S(d|\mathcal{Q})$ for each document d in turn, and return those documents with the highest similarity scores at the end. The execution time of this algorithm is proportional to n, which is not scalable to large collections. Instead, search engines make use of an index that maps terms to the documents that contain them. The most efficient index structure for this purpose is the inverted index. In this paper we assume its most recommended variant, the *frequency-ordered inverted index* [32], and describe it below. For brevity, we refer to it simply as inverted index.

Inverted Index. The index consists of two components – a *dictionary* of terms and a set of *inverted lists*. The dictionary stores, for each distinct term t,

- a count f_t of the documents that contain t, and
- a pointer to the head of the corresponding inverted list.

The inverted list for a term t is a sequence of impact entries $\langle d, w_{d,t}\rangle$ where

- d is the identifier of a document that contains t,
- $w_{d,t}$ is the associated frequency of term t in document d, as defined in the context of Formula (1).

Each inverted list is sorted in decreasing $w_{d,t}$ order. Figure 1 gives an example of a frequency-ordered inverted index, while Figure 2 gives the algorithm for evaluating queries with the inverted index, both adapted from [32]. Given a query, the algorithm begins by calculating $w_{\mathcal{Q},t}$ for each query term t, from $f_{\mathcal{Q},t}$ (derived from the term composition of the query) and f_t (stored in the dictionary). The algorithm then repeatedly reads off the impact entry with the largest term score $c = w_{\mathcal{Q},t} \times w_{d,t}$ among the inverted lists of the query terms, until all the lists are exhausted. We refer to this algorithm as PSCAN, for Prioritized Scanning.

2.2 Cryptographic Primitives

Our proposed authentication mechanism as well as existing verification methods (covered in Section 2.3) build on the following cryptographic primitives.

One-Way Hash: A one-way hash function, denoted as $h(.)$, works in one direction; it is easy to compute the value $h(m)$ for a message m, but computationally infeasible to find a message m that hashes to a given $h(.)$ value. Commonly used hash functions include MD5 [24] and SHA [26]. We refer to $h(m)$ as the hash or digest of m.

Cryptographic Signature: A cryptographic signature algorithm is a tool for verifying the origin, authenticity and integrity of signed messages. Specifically, a signer keeps a private key secret and publishes the corresponding public key. The private key is used by the signer to generate cryptographic signatures on messages. The public key may then be used by anyone to verify a message against its signature. In other words, a message can be authentically signed only by the private key holder, while the signature can be verified with the openly distributed public key. RSA [25] and DSA [9] are two common signature algorithms. We refer to a cryptographic signature simply as signature.

To find the top r matching documents for a query $\mathcal{Q}$, using a frequency-ordered inverted index.

(1) Fetch the first $\langle d, w_{d,t} \rangle$ entry in each query term t's inverted list.
(2) While inverted list entries remain,
  (a) Identify the inverted list entry $\langle d, w_{d,t} \rangle$ with the highest term score $c = w_{\mathcal{Q},t} \times w_{d,t}$, breaking ties arbitrarily.
  (b) If d has not been encountered before, create an accumulator A_d and initialize it to zero.
  (c) $A_d \leftarrow A_d + c$.
  (d) Fetch the next entry in term t's inverted list.
(3) Identify the r largest A_d values and return the corresponding documents.

Figure 2: Prioritized Scanning (PSCAN) Algorithm

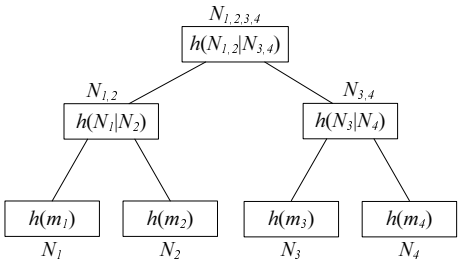


Figure 3: Example of a Merkle Hash Tree

Merkle Hash Tree: The Merkle hash tree (MHT) is a method for collectively authenticating a set of messages [17]. Consider the example in Figure 3, where the owner of messages m_1, m_2, m_3, m_4 wishes to authenticate them. The MHT is built bottom-up, by first computing the leaf nodes N_i as the digests $h(m_i)$ of the messages (where $h(.)$ is a one-way hash function). The value of each internal node is derived from its two child nodes, e.g. $N_{1,2} = h(N_1|N_2)$, where $|$ denotes concatenation. Finally, the digest $N_{1,2,3,4}$ of the root node is signed. The tree can be used to authenticate any subset of the data values, in conjunction with a verification object (VO). For example, to authenticate m_1, the VO contains $N_2, N_{3,4}$ and the signed root $N_{1,2,3,4}$. Upon receipt of m_1, any addressee may verify its authenticity by first computing $h(m_1)$ and then checking whether $h(h(h(m_1)|N_2)|N_{3,4})$ matches the signed root $N_{1,2,3,4}$. If so, m_1 is accepted; otherwise, m_1, N_2, $N_{3,4}$ and/or the signed root have been tampered with. The MHT is a binary tree, though it can be extended to arbitrary directed acyclic graphs [16].

2.3 Related Work

Existing methods for query result verification fall under two categories – MHT approaches and signature-chaining ones. The first methods on authentication of outsourced databases belong to the first category and build upon the MHT (e.g. [8, 20, 2]). [13] proposed the most complete and efficient MHT-based method to date, called Embedded Merkle B-tree (EMB-tree). The general idea is to index the (one-dimensional) data with a B$^+$-tree [7], and to embed into it an MHT with the same fanout. Similar to the original MHT, the root digest is signed by the owner. Posed a range query, the server returns, in addition to the qualifying objects, two *boundary* ones, p^- and p^+, falling immediately to the left and to the right of the range. The VO contains all the left (right) sibling hashes to the path of p^- (p^+). Upon receipt of the result, the user calculates the hashes of the returned objects, and combines them with the VO to reproduce the MHT root digest. If the latter matches the owner's signature, the result is deemed legitimate. In [14] and [22], the MHT approach is extended to data stream authentication.

The signature-chaining schemes [19, 18] rely on multiple signatures. Assuming that the database is ordered according to attribute A, the owner hashes and signs every triple of consecutive data tuples. Posed a range selection query on A, the server returns the qualifying data, along with hashes of the first tuple to the left and the first tuple to the right of the range. It also includes the corresponding signatures in the VO. The user verifies the signatures that "chain" consecutive result tuples. This scheme, initially designed for one-dimensional data, was extended to multi-dimensional index structures in [5, 6]. Signature-chaining approaches are shown to incur very high index construction cost, storage overhead, and user-side verification time [13].

All the above MHT and signature-chaining schemes are built into single- or multi-dimensional index tree structures. In text retrieval, each term in the dictionary is considered a dimension. Since a realistic dictionary easily contains more than 100,000 terms, the dimensionality far exceeds the capability of any index tree [12]; therefore, the existing schemes are not applicable to text search engines. Instead, we need a new authentication mechanism that is designed for inverted indexes, the most widely used structure for document organization/retrieval.

3. AUTHENTICATED TEXT RETRIEVAL

In this section, we introduce our authenticated text retrieval scheme. We begin by defining the system and threat models, as well as the properties that a correct query result must possess. Following that, we map the task of similarity text retrieval to adaptations of two existing threshold-based list merging algorithms, and we present our authentication mechanism for those algorithms. Table 1 summarizes the notation used in the paper.

Symbol	Description	Default
$\mathcal{D}$	Document collection	-
d	A document in $\mathcal{D}$	-
n	# of documents in $\mathcal{D}$	172,961
$\mathcal{T}$	Dictionary of search terms	-
t	A term in $\mathcal{T}$	-
m	# of terms in $\mathcal{T}$	181,978
$\mathcal{L}_i$	Inverted list of term $\mathcal{T}.t_i$	-
l_i	# of entries in $\mathcal{L}_i$	-
$\mathcal{Q}$	A query	-
q	Number of terms in $\mathcal{Q}$	3
$\mathcal{R}$	Query result	-
r	Target # of result documents in $\mathcal{R}$	10
$S(d\|\mathcal{Q})$	Similarity score of document d w.r.t. $\mathcal{Q}$	-
h	A one-way hash function	-
$\|h\|$	Digest size	128 bits
$sign$	A function to generate digital signatures	-
$\|sign\|$	Signature size	1024 bits

Table 1: Notation

3.1 Problem Formulation

System Model. Our system model involves three parties – the data owner, the search engine, and the users.

The data owner manages a data collection $\mathcal{D}$ comprising n documents, $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}, n \geq 1$. To provide similarity text searches, the data owner generates an inverted index on $\mathcal{D}$. The index has two parts – a dictionary and a set of inverted lists. Let $\mathcal{T}$ denote the dictionary of search terms for $\mathcal{D}$, $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}, m \geq 1$. The inverted list $\mathcal{L}_i$ for term $\mathcal{T}.t_i$ is a list of l_i impact pairs, ordered in non-increasing frequency values. Formally, $\mathcal{L}_i = [\langle d_1, f_1\rangle, \langle d_2, f_2\rangle, \ldots, \langle d_{l_i}, f_{l_i}\rangle]$ such that (a) $\forall 1 \leq j \leq l_i$, $\mathcal{L}_i.d_j$ is a document in the collection ($\mathcal{L}_i.d_j \in \mathcal{D}$) and $\mathcal{L}_i.f_j = w_{\mathcal{L}_i.d_j, \mathcal{T}.t_i}$ is $\mathcal{T}.t_i$'s frequency in the document ($w_{d,t}$ is defined in the context of Formula (1)); and (b) $\forall 1 \leq j < k \leq l_i, \mathcal{L}_i.f_j \geq \mathcal{L}_i.f_k$. The data owner transfers the document collection, the inverted index, and the query processing software to a third party, which is contracted to operate the search engine.

The search engine accepts natural language text queries from the users. A user query $\mathcal{Q}$ containing q unique search terms is translated to $\mathcal{Q} = \{\langle t_1, f_1\rangle, \langle t_2, f_2\rangle, \ldots, \langle t_q, f_q\rangle\}$ such that (a) the search terms are in the dictionary ($\forall 1 \leq j \leq q, \mathcal{Q}.t_j \in \mathcal{T}$); and (b) the frequencies of the terms in $\mathcal{Q}$ are $\mathcal{Q}.f_j = w_{\mathcal{Q},\mathcal{Q}.t_j}$ $\forall 1 \leq j \leq q$ ($w_{\mathcal{Q},\mathcal{Q}.t_j}$ is defined in the context of Formula (1)). Any query terms that are not in the dictionary are ignored.

The query result for $\mathcal{Q}$ that is returned to the user, $\mathcal{R}$, is an ordered list of r entries, $\mathcal{R} = [\langle d_1, s_1\rangle, \langle d_2, s_2\rangle, \ldots, \langle d_r, s_r\rangle]$, in which $\forall 1 \leq j \leq r$, $\mathcal{R}.d_j \in \mathcal{D}$ are the result documents and $\mathcal{R}.s_j \in \mathbb{R}$ are their corresponding similarity scores.

Correctness of Query Result. A correct query result $\mathcal{R}$ should relate to the query $\mathcal{Q}$ and underlying document collection $\mathcal{D}$ in the following way: We define the frequency of a document d with respect to a query $\mathcal{Q}$, $freq(d|\mathcal{Q})$, to be the vector of frequency values associated with d in the inverted list for each query term. Formally, $freq(d|\mathcal{Q}) = [f_1, f_2, \ldots, f_q]$ where $\forall 1 \leq j \leq q$, $\mathcal{L}_i$ is the inverted list of search term $\mathcal{Q}.t_j$ (i.e., $\mathcal{T}.t_i = \mathcal{Q}.t_j$), and $d.f_j$ satisfies one of the following conditions:

- if d is in $\mathcal{L}_i$, i.e., $\exists 1 \leq k \leq l_i$, $\mathcal{L}_i.d_k = d$, then $d.f_j$ is the frequency associated with d in $\mathcal{L}_i$, i.e., $d.f_j = \mathcal{L}_i.f_k$;
- if d is not in $\mathcal{L}_i$, i.e., $\forall 1 \leq k \leq l_i, \mathcal{L}_i.d_k \neq d$, then $d.f_j = 0$.

The similarity score of a document d with respect to a query $\mathcal{Q}$ is $S(d|\mathcal{Q}) = \sum_{j=1}^{q} w_{\mathcal{Q},\mathcal{Q}.t_j} \times d.f_j = \sum_{j=1}^{q} w_{\mathcal{Q},\mathcal{Q}.t_j} \times w_{d,t_j}$.

Correctness Criteria: The query result $\mathcal{R}$ is correct if and only if it satisfies the following conditions:

- the result entries are ordered in non-increasing similarity scores, i.e., $\forall 1 \leq j < k \leq r$, $\mathcal{R}.s_j \geq \mathcal{R}.s_k$ where $\mathcal{R}.s_j = S(\mathcal{R}.d_j|\mathcal{Q})$ and $\mathcal{R}.s_k = S(\mathcal{R}.d_k|\mathcal{Q})$;
- all the documents that are excluded from $\mathcal{R}$ have lower similarity scores than the last result entry, i.e., for any non-result document $d \in \mathcal{D}$, it holds that $S(d|\mathcal{Q}) \leq \mathcal{R}.s_r$.

Threat Model. Among the entities in our system model, the third party search engine is the potential adversary. The search engine could be subverted by attackers, or the data owner may not be in a position to qualify the administration procedures employed by the contracted third party. In either case, we assume that the search engine may alter the document collection or the inverted index, it may execute the query processing algorithm incorrectly, or it may tamper with the search results. The users therefore need to verify the correctness of the query results.

We do not consider privacy protection for the user queries. That issue has been studied elsewhere (e.g. [27]) and is beyond the scope of this paper. Instead, we focus on verifying that the query result generated by a text search engine is correct, with respect to the user query and the inverted index as created by the data owner.

3.2 Choice of Authentication Approach

The correctness criteria above hinge on the document scores with respect to the query, $S(d|\mathcal{Q})$. To convince the user that the query result is correct, we therefore need to prove the correctness of the relative document scores. The general approaches include:

1. Pre-certify the document scores. This approach was taken in [21] in authenticating multi-dimensional range aggregates. As $S(d|\mathcal{Q})$ depends on the query terms as well as on $w_{\mathcal{Q},t}$, it is not feasible to materialize all possible document scores beforehand to support ad-hoc search queries.
2. Certify the frequency $w_{d,t}$ for every combination of document $d \in \mathcal{D}$ and term $t \in \mathcal{T}$. Given a query $\mathcal{Q}$, the search engine returns the $w_{d,t}$ values, along with their signatures, for every $d \in \mathcal{D}$ and $t \in \mathcal{Q}$. After verifying the frequencies, the user then computes $S(d|\mathcal{Q})$ himself to find the top r matching documents. The problem with this approach is that the communication cost (to transmit the $q \times n$ certified frequency values) and the user computation overhead (to verify the frequency values and to generate the query result) are prohibitive, thus rendering the approach impractical.
3. Pre-certify every inverted list, and return to the user those that correspond to the query terms. After checking the integrity of the lists, the user may compute the document scores to produce the query result. This approach fits naturally with the PSCAN algorithm in Figure 2. However, the retrieval of entire lists imposes very large I/O costs on the search engine. Also, returning the entire inverted lists as proof incurs excessive communication cost, as well as high verification and memory requirements at the user-side.
4. Dynamically generate certified fragments of the inverted lists. For general ad-hoc queries that are likely to include one or more common terms, it is desirable to return only small fractions of the inverted lists, and still allow the query result to be verified.

Our proposed authenticated threshold algorithms adopt the last approach. To motivate our choice, we observe that the inverted lists for real corpora follow a highly skewed distribution. Most of the terms have very short inverted lists, whereas a small minority of the inverted lists are several orders of magnitude longer. To illustrate, Figure 4 plots the list length distribution for the WSJ corpus (we describe it in detail in Section 4). More than 50% of the terms have only between 2 and 5 entries in their inverted lists, whereas the longest list contains 127,848 entries. With such a distribution, most queries are likely to hit one or more short inverted lists, due to their sheer numbers. At the same time, the long lists correspond to the common terms, so they are likely to be queried too. In practice, most queries would involve a mix of long and short inverted lists. For such queries, the short lists are expected to contribute most of the result documents, because they have smaller f_t's and hence larger $w_{\mathcal{Q},t}$ weights compared to the long lists, while only a few leading entries are needed from the long lists. Therefore, it is advantageous to avoid transmitting to the users the long inverted lists in their entirety. In Sections 3.3 and 3.4 we describe two schemes that achieve this objective.

3.3 Threshold with Random Access

Our first query processing algorithm is Threshold with Random Access (TRA). It modifies the PSCAN algorithm (in Figure 2) to

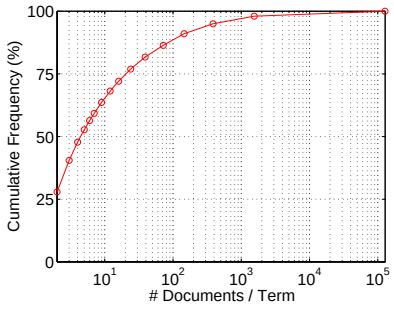


Figure 4: Inverted List Length Distribution for WSJ Corpus

terminate once it determines that the inverted lists are not able to produce new documents with a large enough similarity score to qualify for the query result. The TRA algorithm is summarized in Figure 5.

The algorithm repeatedly reads off the next entry $\langle d_j, f_j \rangle$ from the inverted list $\mathcal{L}_i$ with the largest term score $c_i = w_{Q,Q.t_i} \times \mathcal{L}_i.f_j$ among the inverted lists of the query terms. Document $\mathcal{L}_i.d_j$'s similarity score s is immediately computed (hence Random Access) by fetching directly the query term frequencies from the document-MHT (a structure that is additionally used for proof construction and is described shortly). An entry $\langle d, s \rangle$ for this document is then inserted into the result list $\mathcal{R}$, which is ordered by descending s. TRA also maintains a threshold *thres* that is computed over the current term score c_i of each inverted list, $thres = \sum_{i=1}^{q} c_i$. The threshold forms an upper bound on the similarity score of any non-encountered documents further down the inverted lists. Therefore, as soon as $thres \leq \mathcal{R}.s_r$, the top r matching documents have been found.

To illustrate the algorithm, consider the inverted index in Figure 1. Suppose that the user searches for the two (i.e., $r = 2$) closest documents to the query "sleeps in the dark". The respective $w_{\mathcal{Q},t}$ values and inverted lists are shown in Figure 6. The initial *thres* is 0.8135, based on the term scores c_1 to c_4 from the leading entries in the four inverted lists. In the first iteration, c_3 is the largest, so we pop $\langle 5, 0.265 \rangle$ from the third list. We then retrieve d_5's term frequencies to compute its score $S(d_5|\mathcal{Q}) = 0.416$ and place $\langle 5, 0.416 \rangle$ into $\mathcal{R}$. Following that, we retrieve the next entry $\langle 3, 0.263 \rangle$ from the third list, and update *thres* to 0.8115. After subsequent iterations pop $\langle 6, 0.200 \rangle$, $\langle 6, 0.079 \rangle$, and $\langle 6, 0.079 \rangle$, *thres* falls below $\mathcal{R}.s_2 = 0.416$, and the algorithm terminates.

The above query processing algorithm is an adaptation of the "Threshold with Random Access" algorithm in [10]. The algorithm there examines each list to an equal depth, i.e., the same number of entries are polled from each list. This behavior is not desirable for our search engine, where some inverted lists that correspond to common words are orders of magnitude longer than those for the rarer terms. To minimize processing and authentication costs, we modify the algorithm to favor entries that contribute higher term scores c_i, so as to examine and prove as few $\langle d, f \rangle$ entries as possible from the inverted lists.

To prove that $\mathcal{R}$ satisfies the correctness criteria in Section 3.1, the search engine returns to the user a verification object (VO) that contains the following information:

- For each result document d, the VO includes the query term frequencies in d so that the user can compute the document score $S(d|\mathcal{Q})$;
- For each non-result document d that occurs up to the cut-off threshold in any of the inverted lists involved in the query, the VO includes the query term frequencies in d so the user can verify that its score $S(d|\mathcal{Q})$ is lower than those of the result documents;
- The inverted list entries that correspond to the cut-off threshold, to satisfy the user that any non-encountered documents further down in the lists cannot have similarity scores that exceed those of the result documents.

The entries that constitute the cut-off threshold are shaded in Figure 6, whereas those documents for which the query term frequencies are returned are highlighted in bold. We first present a simple authentication mechanism that is based on plain Merkle hash trees (MHT). After examining the pros and cons of this simple approach, we then introduce our improved Chain-MHT technique.

3.3.1 Authentication with Merkle Hash Trees

Our first authentication mechanism requires the following structures: (a) A MHT is constructed over the entries in each inverted list; this structure is called *term-MHT* of that term/list. Only the document identifiers are used here; their corresponding frequencies are omitted. (b) A *document-MHT* is built for each document d, over the terms that appear in d and their corresponding frequencies. Guided by the findings in [13], we store only the root and the leaves of the MHTs; the intermediate digests are regenerated when they are needed at runtime. We name this the TRA-MHT mechanism (TRA query processing with MHT authentication).

Figure 7 illustrates the MHT for term t_{16} ('the'). The leaves of the MHT are the document identifiers, ordered exactly as they appear in the inverted list for t_{16}. For the query in Figure 6, TRA has read the first four entries, which correspond to documents d_5, d_3, d_6, and d_1. To prove that these indeed are the first four entries in $\mathcal{L}_{16}$, the search engine inserts into the VO the document identifiers (i.e., numbers) $5, 3, 6, 1$, the digest h_{5-8}, together with the signed root of the term-MHT. The user may verify that the entries for d_5, d_3, d_6, d_1 are the leading entries of $\mathcal{L}_{16}$ by computing $h_1 = h(5), h_2 = h(3), h_3 = h(6), h_4 = h(1)$, and combining them to derive h_{12} and h_{34}, and then h_{1-4}. Subsequently, he combines h_{1-4} with h_{5-8} (from the VO), and verifies that the derived digest matches the signed root. After verifying in this manner all the $q = 4$ inverted lists involved in $\mathcal{Q}$, the user proceeds to check the query term frequencies and compute the score of each encountered document, using its document-MHT as follows.

Consider d_6, whose document-MHT is depicted in Figure 8. The leaves of this MHT are the identifier-frequency pairs of all the terms in the document, in ascending identifier order. Since the query terms are t_{15} (sleeps), t_8 (in), t_{16} (the) and t_3 (dark), the search engine adds their corresponding frequencies to the VO. The complementary digests[1] and the signed root are also included in the VO. These items are shaded in the figure. The VO items enable the user to verify the query term frequencies in d_6. Similar VO construction and verification procedures apply to all the other documents that the user needs to check (i.e., the documents whose entries are in bold in Figure 6). In the case where a query term does not appear in a document, the proof entails checking the pair of consecutive terms in the document-MHT that bound the query term in question. For example, if a query involves t_7 and needs to check d_6, then t_3 and t_8 are returned (together with their complementary digests). Since t_3 and t_8 are consecutive leaves in the MHT, the user is assured that d_6 does not contain t_7. The certified $w_{d,\mathcal{Q}.t_i}$ frequencies, combined

[1]These are all the sibling digests along the path from the MHT root down to the corresponding leaf, similar to the standard MHT functionality described in Section 2.2. Note that digests common across multiple leaves need to be included only once in the VO.

To find the top r matching documents for a query $\mathcal{Q}$, using a frequency-ordered inverted index.

(1) Initialize the sorted list $\mathcal{R}$.
(2) Fetch the first $\langle d, f\rangle$ entry in each query term t_i's inverted list $\mathcal{L}_i$.
(3) Compute *thres* $= \sum_{i=1}^{q} w_{\mathcal{Q},\mathcal{Q}.t_i} \times \mathcal{L}_i.f$.
(4) While inverted list entries remain,
  (a) If $\mathcal{R}.s_r \geq$ *thres*, go to step (5).
  (b) Pop the inverted list entry $\langle d, f\rangle$ with the highest term score $c_i = w_{\mathcal{Q},\mathcal{Q}.t_i} \times \mathcal{L}_i.f$, breaking ties arbitrarily.
  (c) If d has not been encountered before,
    (i) Retrieve w_{d,t_j} for every query term t_j.
    (ii) Compute d's similarity score $s = S(d|\mathcal{Q})$.
    (iii) Insert $\langle d, s\rangle$ into $\mathcal{R}$.
  (d) Update *thres*.
(5) Return the first r entries in $\mathcal{R}$ as the query result.

Figure 5: Threshold with Random Access (TRA) Algorithm

Query Term t	$w_{\mathcal{Q},t}$		**Inverted List for** t						
sleeps	2.3979	$\mapsto$	⟨**6, 0.079**⟩	⟨END, 0⟩					
in	1.0986	$\mapsto$	⟨6, 0.159⟩	⟨2, 0.148⟩	⟨5, 0.142⟩	⟨1, 0.058⟩	⟨7, 0.058⟩	⟨8, 0.053⟩	...
the	0.9808	$\mapsto$	⟨**5, 0.265**⟩	⟨**3, 0.263**⟩	⟨**6, 0.200**⟩	⟨1, 0.159⟩	⟨2, 0.148⟩	⟨4, 0.125⟩	...
dark	2.3979	$\mapsto$	⟨**6, 0.079**⟩	⟨END, 0⟩					
Result:		=	⟨**6, 0.750**⟩	⟨**5, 0.416**⟩					

Iteration	*thres*	**Pop Entry**	$\mathcal{R}$
1	0.8135	⟨5, 0.265⟩ for 'the'	[⟨5, 0.416⟩]
2	0.8115	⟨3, 0.263⟩ for 'the'	[⟨5, 0.416⟩, ⟨3, 0.263⟩]
3	0.7497	⟨6, 0.200⟩ for 'the'	[⟨6, 0.750⟩, ⟨5, 0.416⟩]
4	0.7095	⟨6, 0.079⟩ for 'sleeps'	[⟨6, 0.750⟩, ⟨5, 0.416⟩]
5	0.5201	⟨6, 0.079⟩ for 'dark'	[⟨6, 0.750⟩, ⟨5, 0.416⟩]
6	0.3306	Terminate	[⟨6, 0.750⟩, ⟨5, 0.416⟩]

Figure 6: Search for Top Two Matches to the Query "sleeps in the dark" with the TRA Algorithm

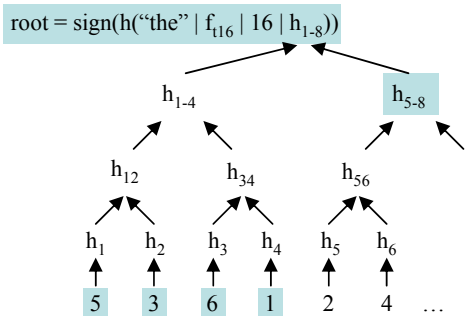


Figure 7: Term-MHT over the Inverted List of t_{16} ('the')

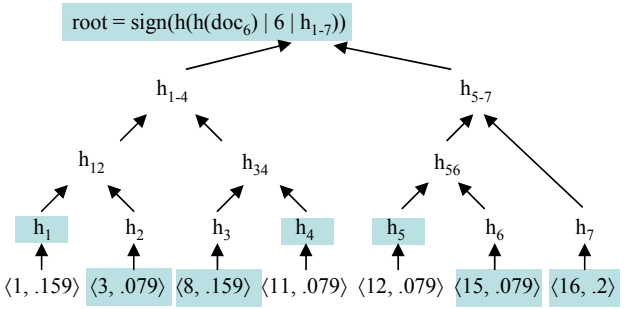


Figure 8: Document-MHT over d_6

with the $f_{\mathcal{Q}.t_i}$ values (which are added to the VO and verified by the root of the corresponding term-MHT, e.g. $f_{t_{16}}$ is included in the signed root of Figure 7), allow the user to compute/verify the score $S(d|\mathcal{Q})$ for each document d encountered by TRA.

Regarding the integrity of the actual document contents, note that the root of each document-MHT includes the digest of the entire document (e.g. $h(doc_6)$ in Figure 8). If TRA encounters a document d and $d \notin \mathcal{R}$, then the corresponding document digest is inserted into the VO. If $d \in \mathcal{R}$, its digest is not included in the VO, but is computed at the user-side during result verification. Any attempt by the server to tamper with the document content would lead to a mismatch with the signed root of the document-MHT.

The above authentication mechanism is a simple application of the Merkle hash tree. However, it has some serious shortcomings:

- Although TRA terminates at the cut-off threshold, the search engine has to retrieve the entire inverted lists in order to re-generate the complementary digests of the term-MHTs (e.g. h_{5-8} in Figure 7). Furthermore, these digests (whose number increases with the length of the list) have to be transmitted to and processed by the user. This is particularly undesirable as some lists are extremely long (see Figure 4).
- The leaves of term-MHTs and document-MHTs are smaller than the upper level digests. Specifically, with 4-byte term identifiers and frequencies, a document-MHT leaf occupies 8 bytes, whereas an internal node (digest) is 16 bytes long. Instead of digests, it may be cheaper to transmit the underlying leaves; e.g. in Figure 8 it is more efficient to return all the leaves and omit the three shaded digests from the VO.

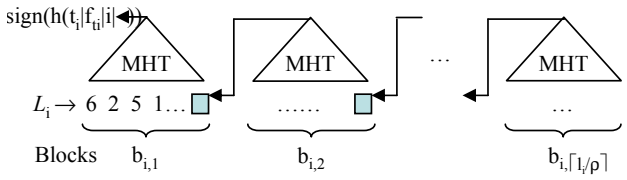


Figure 9: Chain of MHTs over Inverted List $\mathcal{L}_i$

3.3.2 Authentication with Chain Merkle Hash Trees

Our next authentication mechanism, TRA-CMHT (TRA query processing with Chain-MHTs), is designed to overcome the inefficiencies of TRA-MHT through two techniques – chain-MHT, and buddy inclusion.

To avoid retrieving and processing entire inverted lists, we observe that: (a) The entries in an inverted list are stored and retrieved in disk blocks. If even one entry in a block is fetched, the remaining entries in the same block are available without extra I/O cost. (b) The inverted list is always accessed from the front. These observations lead us to propose the following *chain-MHT* scheme.

First, we build an embedded MHT over the inverted list entries within each block. Next, we establish a hash chain over the blocks in an inverted list. Moving from the last block towards the front, we include the digest of each block in the digest computation of the block immediately ahead of it. Finally, the digest of the first block is signed. This signature can be used to verify any j leading blocks of the inverted list, by supplying just the digest of the $j+1$ block. The details are as follows.

Recall that l_i denotes the number of $\langle d, f\rangle$ entries in inverted list $\mathcal{L}_i$. Suppose a block holds up to ρ document identifiers (we will discuss the setting of ρ shortly), $\mathcal{L}_i$ is stored as a sequence of blocks $b_{i,1}, b_{i,2}, \ldots, b_{i,\lceil l_i/\rho\rceil}$, with $b_{i,1}$ holding the first ρ document identifiers in $\mathcal{L}_i$, $b_{i,2}$ holding the next ρ, and so on. Let $b.docid$ denote the document identifiers within block b.

$$\begin{aligned} digest_{i,\lceil l_i/\rho\rceil} &= MHT(b_{i,\lceil l_i/\rho\rceil}.docid) \\ digest_{i,j} &= MHT(b_{i,j}.docid + digest_{i,j+1}), \\ &\qquad \forall 1 \le j < \lceil l_i/\rho\rceil \\ \mathcal{L}_i.signature &= sign(h(\mathcal{T}.t_i \mid f_{\mathcal{T}.t_i} \mid i \mid digest_{i,1})) \end{aligned}$$

where $MHT(obj\text{-}list)$ returns the root digest of the Merkle hash tree over the objects in *obj-list*, and $sign(msg)$ returns a digital signature of *msg*. For any block $b_{i,j}$, only the root digest of its MHT is stored in the preceding block $b_{i,j-1}$. Any internal node in the MHT that is needed at runtime is regenerated dynamically from the MHT leaves in $b_{i,j}$, in order to minimize storage and retrieval overheads. The scheme is illustrated in Figure 9. The leaves in the MHTs here comprise only the document identifiers; their associated frequencies are stored with the document-MHTs as before.

With the chain-MHT, the VO for a query result contains the signature of every query term's inverted list[2]. When the query processing algorithm terminates, all the processed document identifiers in the inverted lists of the query terms are also added to the VO. In addition, for each inverted list, MHT digests that cover the unprocessed entries in the last retrieved block and the digest of the succeeding block are also computed and inserted into the VO. Consider Figure 7 again, but treat the Merkle hash tree as being constructed over the last retrieved block of the term, rather than over the entire inverted list. The shaded digest (h_{5-8}) is the only one that is included in the VO for $\mathcal{L}_{16}$. An important advantage of our chain-MHT scheme is that the number of digests per term in the VO is only proportional to $log_2(\rho+1)$, and is independent of the length of the inverted list. This limits the VO construction and transmission costs, as well as the user verification cost.

We now consider a realistic setting for ρ. Each block in the inverted list has to hold the disk address and digest of the succeeding block. The remaining space in the block is then reserved for the ρ document identifiers. For instance, with 1-Kbyte blocks, 4-byte document identifiers, 4-byte disk addresses and 16-byte digests, $\rho = \lfloor \frac{1024-4-16}{4} \rfloor = 251$.

Next, we turn to the second inefficiency identified in Section 3.3.1. We address it through *buddy inclusion*, which works as follows. We partition the leaves in each (document- or term-) MHT into groups of 2^g, where g is the largest integer that satisfies $(2^g - 1) \times |\text{leaf}| \le g \times |h|$, $|h|$ is the size of a digest, and $|\text{leaf}|$ is the size of a leaf. Whenever a leaf node needs to be added to the VO, its buddies in the same group are also included. To illustrate, consider Figure 8 where $|h| = 16$ bytes and $|\text{leaf}| = 8$ bytes. The above inequality yields $g = 2$, so we organize the leaves into groups of $2^g = 4$. For required entry $\langle 3, 0.079\rangle$, for instance, we additionally include in the VO its buddies $\langle 1, 0.159\rangle$, $\langle 8, 0.159\rangle$ and $\langle 11, 0.079\rangle$ (this happens to cover another required entry, $\langle 8, 0.159\rangle$). Similarly, required entry $\langle 15, 0.079\rangle$ brings the remaining leaves into the VO. Thus, we avoid including h_1, h_4, h_5.

3.4 Threshold with No Random Access

This section introduces another query processing algorithm, called Threshold with No Random Access (TNRA). Like TRA, TNRA terminates once it determines that the top r result documents have emerged. Unlike TRA, TNRA does not retrieve the term frequencies in the polled documents directly. Rather, it waits until enough of the term frequencies are gleamed from the inverted lists to determine the *relative* similarity scores of the polled documents.

Before introducing the algorithm, we first define some notation:

- $S^{UB}(d|\mathcal{Q})$: The upper bound of document d's similarity score with respect to query $\mathcal{Q}$ is $S^{UB}(d|\mathcal{Q}) = \sum_{i=1}^{q} w_{\mathcal{Q},\mathcal{Q}.t_i} \times \gamma_{d,\mathcal{Q}.t_i}$ where $\gamma_{d,\mathcal{Q}.t_i} = w_{d,\mathcal{Q}.t_i}$ if d has been polled from $\mathcal{Q}.t_i$'s inverted list; otherwise $\gamma_{d,\mathcal{Q}.t_i}$ equals to the latest frequency read from that inverted list.
- $S_{LB}(d|\mathcal{Q})$: The lower bound of document d's similarity score with respect to query $\mathcal{Q}$ is $S_{LB}(d|\mathcal{Q}) = \sum_{i=1}^{q} w_{\mathcal{Q},\mathcal{Q}.t_i} \times \gamma_{d,\mathcal{Q}.t_i}$ where $\gamma_{d,\mathcal{Q}.t_i} = w_{d,\mathcal{Q}.t_i}$ if d has been polled from $\mathcal{Q}.t_i$'s inverted list; otherwise $\gamma_{d,\mathcal{Q}.t_i} = 0$.

The TNRA algorithm ensures the correctness of query results, by checking for the following termination conditions:

- there is complete ordering among the documents in the result $\mathcal{R}$, i.e., $\forall 1 \le j < k \le r$, $S_{LB}(\mathcal{R}.d_j|\mathcal{Q}) \ge S^{UB}(\mathcal{R}.d_k|\mathcal{Q})$;
- the upper bound on the score of every document d polled so far, such that $d \notin \mathcal{R}$, does not exceed the lower bound score of the last result document, i.e., $S^{UB}(d|\mathcal{Q}) \le S_{LB}(\mathcal{R}.d_r|\mathcal{Q})$;
- the threshold does not exceed the lower bound score of the last result document, i.e., $thres = \sum_{i=1}^{q} c_i \le S_{LB}(\mathcal{R}.d_r|\mathcal{Q})$.

Figure 10 sketches the TNRA algorithm. To illustrate, consider again the inverted index in Figure 1 and the query for the two closest documents to the query "sleeps in the dark". Figure 11 traces the execution of the algorithm. In the first iteration, c_3 is the largest, so $\langle 5, 0.265\rangle$ is popped from the third list. At this time, the lower bound for d_5's similarity score is $c_3 = 0.260$, while its upper bound is $c_1 + c_2 + c_3 + c_4 = 0.813$. The next two iterations bring d_3 and d_6 into $\mathcal{R}$. Iteration 4 pops $\langle 6, 0.079\rangle$ from the first list, so

[2]The list signatures could be consolidated through an aggregated signature scheme [3], so that only one signature is returned for the entire query result.

To find the top r matching documents for a query $\mathcal{Q}$, using a frequency-ordered inverted index.

(1) Initialize the sorted list $\mathcal{R}$.
(2) Fetch the first $\langle d, f\rangle$ entry in each query term t_i's inverted list $\mathcal{L}_i$.
(3) Compute *thres* $= \sum_{i=1}^{q} w_{\mathcal{Q},\mathcal{Q}.t_i} \times \mathcal{L}_i.f$.
(4) While inverted list entries remain,
  (a) If the following termination conditions hold, go to step (5):
    • $\forall 1 \le j < k \le r$, $S_{LB}(\mathcal{R}.d_j|\mathcal{Q}) \ge S^{UB}(\mathcal{R}.d_k|\mathcal{Q})$;
    • $\forall j > r$, $S^{UB}(\mathcal{R}.d_j|\mathcal{Q}) \le S_{LB}(\mathcal{R}.d_r|\mathcal{Q})$;
    • $S_{LB}(\mathcal{R}.d_r|\mathcal{Q}) \ge$ *thres*.
  (b) Pop the inverted list entry $\langle d, f\rangle$ with the highest term score $c_i = w_{\mathcal{Q},\mathcal{Q}.t_i} \times \mathcal{L}_i.f$, breaking ties arbitrarily.
  (c) If d has not been encountered before, insert $\langle d, S_{LB}(d|\mathcal{Q}), S^{UB}(d|\mathcal{Q})\rangle$ into $\mathcal{R}$;
    else update $S_{LB}(d|\mathcal{Q})$ and $S^{UB}(d|\mathcal{Q})$ in $\mathcal{R}$.
  (d) Update *thres* and $S^{UB}(\mathcal{R}.d_j|\mathcal{Q})$ $\forall \mathcal{R}.d_j$.
(5) Return the first r entries in $\mathcal{R}$ as the query result.

Figure 10: Threshold with No Random Access (TNRA) Algorithm

Query Term t	$w_{\mathcal{Q},t}$		**Inverted List for** t						
sleeps	2.3979	$\mapsto$	⟨**6, 0.079**⟩	⟨END, 0⟩					
in	1.0986	$\mapsto$	⟨**6, 0.159**⟩	⟨**2, 0.148**⟩	⟨**5, 0.142**⟩	⟨1, 0.058⟩	⟨7, 0.058⟩	⟨8, 0.053⟩	...
the	0.9808	$\mapsto$	⟨**5, 0.265**⟩	⟨**3, 0.263**⟩	⟨**6, 0.200**⟩	⟨1, 0.159⟩	⟨2, 0.148⟩	⟨4, 0.125⟩	...
dark	2.3979	$\mapsto$	⟨**6, 0.079**⟩	⟨END, 0⟩					
Result:		=	⟨**6, 0.750**⟩	⟨**5, 0.416**⟩					

Iteration	*thres*	**Pop Entry**	$\mathcal{R}$
1	0.814	⟨5, 0.265⟩ for 'the'	[⟨5, 0.260, 0.813⟩]
2	0.812	⟨3, 0.263⟩ for 'the'	[⟨5, 0.260, 0.813⟩, ⟨3, 0.258, 0.811⟩]
3	0.750	⟨6, 0.200⟩ for 'the'	[⟨5, 0.260, 0.813⟩, ⟨3, 0.258, 0.811⟩, ⟨6, 0.196, 0.750⟩]
4	0.710	⟨6, 0.079⟩ for 'sleeps'	[⟨6, 0.386, 0.750⟩, ⟨5, 0.260, 0.624⟩, ⟨3, 0.258, 0.622⟩]
5	0.520	⟨6, 0.079⟩ for 'dark'	[⟨6, 0.575, 0.750⟩, ⟨5, 0.260, 0.435⟩, ⟨3, 0.258, 0.433⟩]
6	0.331	⟨6, 0.159⟩ for 'in'	[⟨6, 0.750, 0.750⟩, ⟨5, 0.260, 0.423⟩, ⟨3, 0.258, 0.421⟩]
7	0.319	⟨2, 0.148⟩ for 'in'	[⟨6, 0.750, 0.750⟩, ⟨5, 0.260, 0.416⟩, ⟨3, 0.258, 0.414⟩, ⟨2, 0.163, 0.319⟩]
8	0.312	⟨5, 0.142⟩ for 'in'	[⟨6, 0.750, 0.750⟩, ⟨5, 0.416, 0.416⟩, ⟨3, 0.258, 0.322⟩, ⟨2, 0.163, 0.319⟩]
9	0.220	Terminate	[⟨6, 0.750, 0.750⟩, ⟨5, 0.416, 0.416⟩]

Figure 11: Search for Top Two Matches to the Query "sleeps in the dark" with the TNRA Algorithm

d_6's lower bound increases to $c_1 + 0.260 = 0.386$. Since the list contains no further entries, its contribution c_1 is deducted from the upper bound of all the documents in $\mathcal{R}$. The three termination conditions are satisfied only in iteration 9, whereas TRA would finish in 6 iterations as we saw previously. In general, TNRA is expected to poll a higher fraction of the inverted lists than TRA. The advantage of TNRA is that it avoids the I/O cost of fetching immediately the term frequencies in the polled documents. We defer a systematic comparison of TRA versus TNRA to Section 4.

Our TNRA algorithm is an adaptation of the "Threshold with No Random Access" algorithm in [10]. Again, the algorithm there examines each list to an equal depth, i.e., the same number of entries are polled from each list. In contrast, our adaptation favors those inverted lists that contribute higher term scores c_i, and is more appropriate for text search engines in which some inverted lists are orders of magnitude longer than others.

To support query result authentication for TNRA, the search engine has to include in the VO the $\langle d, f\rangle$ entries, from the front down to the cut-off threshold in each inverted list. The cut-off threshold in each list, shaded in Figure 11, must add up to the overall threshold. If the search engine were to return a wrong query answer, it would have to be substantiated by altering either the value in some $\langle d, f\rangle$ entries or the order of the entries within some inverted list. However, any such alteration would cause a mismatch with the signature of the corresponding inverted list(s), so the user would be able to detect that the answer is wrong.

Like TRA, TNRA can be coupled with MHT or CMHT. However, we do not require separate document-MHTs here. Instead, we incorporate the term frequencies into the term-MHT or chain-MHT, so each leaf node is a pair of a document identifier and its term frequency. Figure 12 illustrates the modified CMHT structure. The VO construction and verification procedures remain the same as those of TRA. The number of entries per block, ρ', is computed similarly to ρ in Section 3.3.2, the difference being that now each leaf is (and has the size of) an identifier-frequency pair.

Before proceeding to the empirical study, we discuss a space optimization technique that is applicable to both TRA and TNRA. In our methods as presented so far, the search engine stores one signature for every inverted list. We can reduce this number down to one, at the expense of a larger VO size. Specifically, we can build an implicit (i.e., computed-on-demand) dictionary-MHT on top of the root digests of the individual term-MHTs or chain-MHTs, and sign only the root of the dictionary-MHT. Although this approach reduces the space requirements, it leads to additional digests in the VO (from the dictionary-MHT). This trade-off is not very appealing in general, since the signature size is negligible compared to that of the documents themselves. It may, however, be useful in extreme cases where the search engine has insufficient storage.

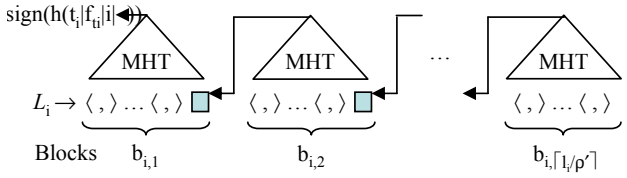


Figure 12: Chain-MHT for the TNRA Algorithm

4. EMPIRICAL EVALUATION

In this section, we experimentally evaluate the performance of our authentication schemes. The key questions that we are seeking answers to include:

- In the context of text search engines, what are the relative strengths and weaknesses of the two query processing algorithms – TRA and TNRA?
- How efficient and robust are our proposed techniques of chain-MHT and buddy inclusion in supporting query result authentication?
- Are the costs associated with our proposed authentication techniques low enough to be practical?

After describing the experiment set-up, we report findings obtained with a synthetic workload (in Sections 4.2 and 4.3) and a set of standard TREC queries (in Section 4.4). Next, we summarize the answers to the above questions obtained from our empirical study. We follow the notation and default parameter settings in Table 1.

4.1 Experiment Set-Up

Dataset: The WSJ corpus for our experiments comprises 172,961 articles published in the Wall Street Journal from December 1986 to March 1992. The combined size of the articles is around 513 Mbytes. After removing stopwords (common words like 'the' and 'a' that are not useful for differentiating between documents) and words that appear in only one document, we are left with 181,978 terms for the dictionary. The removal of these words is a standard step in document indexing [1], and not specific to our methods. The length distribution of the inverted lists is depicted in Figure 4.

Workloads: We run two workloads on the test document set. The first is a synthetic workload of 1000 queries, composed of randomly selected terms from the dictionary. The second workload is made up of the TREC-2 and TREC-3 ad-hoc queries (topics 101 to 200) [29]. The TREC queries contain between 2 and 20 terms each, and provide realistic term compositions for testing our authentication schemes. Another variable in the workload is the result size r, i.e., the number of desired documents in each search result.

Algorithms: The algorithms to be evaluated are TRA-MHT (Threshold with Random Access + MHT authentication), TRA-CMHT (Threshold with Random Access + Chain-MHT authentication), TNRA-MHT (Threshold with No Random Access + MHT authentication), and TNRA-CMHT (Threshold with No Random Access + Chain-MHT authentication).

System implementation: Our system first loads the corpus into the Lucene search engine [15], which parses the documents, performs stopword removal but not stemming [1], and creates an inverted index. Next, we write out Lucene's index into a dictionary of terms, along with an inverted list for each of them. The same inverted index is used in all compared algorithms. The authentication information introduced by TNRA requires less than 1% extra space over a plain, non-authenticated inverted index, while TRA requires around 25% more space (due to its document-MHTs).

System configuration: The test system runs on a Redhat Linux box with a dual Intel Xeon 3GHz CPU, 512MB RAM and a Seagate ST973401KC 73GB hard disk. The disk is formatted with 1-Kbyte blocks, the default in Linux. In view of Figure 4, which shows that over 50% of the inverted lists contain no more than 5 entries, raising the block size would only increase the space wastage and I/O overhead. To model practical search engines that support large document sets, only the dictionary is pinned in memory. We store the inverted lists, documents, and authentication structures on the disk and prevent them from being cached in memory.

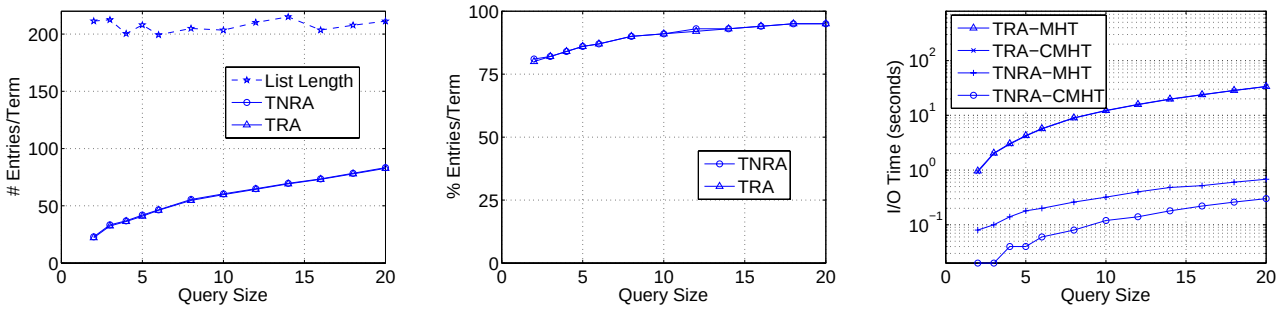
Performance metrics: The metrics capture the dominant costs incurred by the parties in our system model. They are: (i) The disk I/O time at the search engine, which practically accounts for all of the processing time because the CPU computations overlap with I/O activities, and computation times are orders of magnitude shorter than I/Os. (ii) The size of the VOs that are transmitted to the user. (iii) The computation cost that the user expends to verify the query result. We measure only the costs incurred to generate and verify the search results; the cost of retrieving and verifying the actual documents are constant across all algorithms and are omitted.

4.2 Sensitivity to Query Size

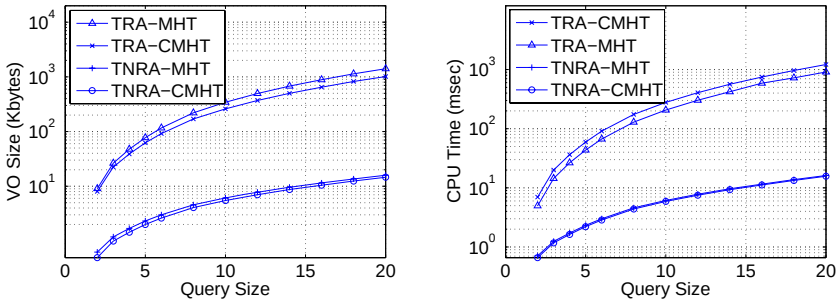
We begin by examining how the various algorithms behave for different query sizes. We run the queries from the synthetic workload, one at a time, to search for the 10 best-matching documents. The other parameters are fixed at their default settings in Table 1. Figure 13 presents average measurements over the 1000 queries.

Figure 13(a) shows the average number of entries that are read from each inverted list for query processing; equivalently, it is the number of entries to be included inside the VO. The line labeled "List Length" shows the average number of entries in the queried inverted lists; it essentially corresponds to the third approach in Section 3.2, and serves as a baseline for our methods. The early termination feature of TRA and TNRA enables them to access far fewer entries. This feature is particularly effective for queries that contain only a few terms. A closer examination of these queries reveals that they rarely involve more than one long inverted list. The short inverted lists are exhausted quickly because of their higher weights in the ranking function (see Formula (1)). When only a long inverted list remains, its front entries are sufficient to complete the query result. As the query size increases, however, the queries are more likely to hit multiple long inverted lists. At the same time, the threshold which is summed from the current term scores of all the lists takes longer to fall below the similarity score of the last result document. This explains the rise in the number of entries read. The advantage of TRA over TNRA is marginal. The reason is that they differ significantly only for long inverted lists, which constitute a small fraction of the queried lists. This is substantiated by Figure 13(b), which shows TRA and TNRA fetching a high fraction of the inverted lists on average, even though the actual number of entries read is low (in Figure 13(a)). Note that only one line is given for each of TRA and TNRA, because their respective variants have the same cut-off thresholds and, thus, equal number of entries read and included in the VO.

Figure 13(c) plots the I/O time of the algorithms in logarithmic scale. Despite reading fewer inverted list entries for query processing, both TRA variants (with MHT and with CMHT) incur higher I/O costs than TNRA. This is because they incur random I/Os in fetching the document-MHTs. Between the two variants, TRA-CMHT has an edge as it does not need to fetch the entire inverted lists to construct the VO, although in the figure the gain is overshadowed by the cost of random accesses to the document-MHTs. In contrast, TNRA benefits from sequential I/Os. Between its vari-



(a) Average # Entries Read (b) % of Inverted List Read (c) I/O Time



(d) VO Size (e) CPU Time

Figure 13: Synthetic Workload, Varying Query Size with Result Size = 10

QSize	2	4	6	8	10	12	14	16	18	20
MHT:										
Data (%)	6	8	9	10	11	12	12	13	13	14
Digest (%)	94	92	91	90	89	88	88	87	87	86
CMHT:										
Data (%)	22	28	31	34	36	38	40	41	42	43
Digest (%)	78	72	69	66	64	62	60	59	58	57

Table 2: Breakdown of VO Size

ants, TNRA-CMHT requires less than 40% the I/O time of TNRA-MHT. The latter reads the entire inverted lists for VO construction (in order to regenerate the internal term-MHT nodes) while the former stops right at the (block that contains the) cut-off threshold.

Figure 13(d) shows the VO size in the same experiment. TRA is worse than TNRA in this aspect too, again due to its document-MHTs. Specifically, in a document-MHT, even non-query terms contribute to the VO in the form of digests associated with the internal nodes of the MHT. Additionally, each query term that is not in a document requires two boundary leaf nodes as a proof. The above factors combined lead to TRA VOs that are several times larger than those of TNRA. Between the TNRA variants, TNRA-CMHT returns a 10% to 20% smaller VO. These savings are achieved through the chain-MHT and buddy inclusion techniques. Figure 13(e) shows the CPU time spent at the user-side for verification; the trends are similar to Figure 13(d), since the verification cost is proportional to the number of digests in the VO.

Next, we take a closer look at the effect of our chain-MHT and buddy inclusion techniques. Table 2 breaks down TRA's VO composition for the MHT and CMHT variants. The rows that are labeled "Data" indicate the size of data objects (i.e., the leaf nodes of the document-MHTs) in the VO, whereas the "Digest" rows show the contribution of the internal node digests. The large 'Data' contribution under CMHT is evidence of the effectiveness of the buddy inclusion and chain-MHT optimizations; the two techniques combined reduce the VO size by 30%.

4.3 Sensitivity to Result Size

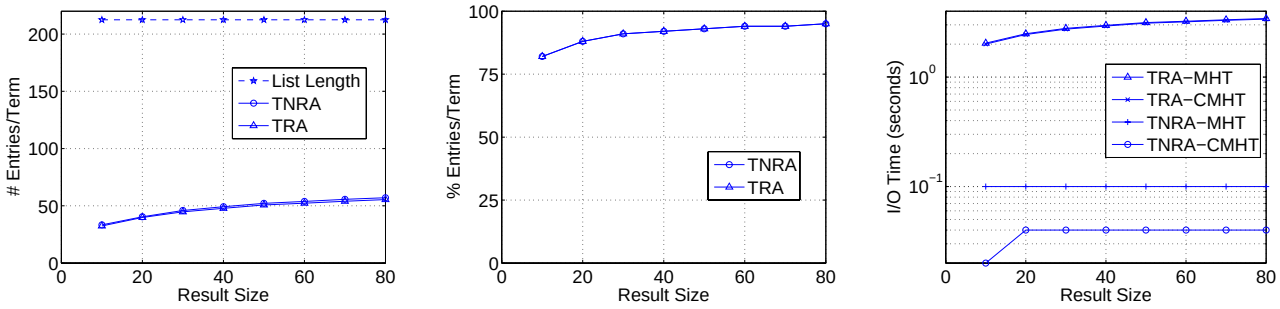
Next, we study the sensitivity of the algorithms to r, the number of desired result documents. We use the synthetic workload and vary r from 10 up to 80, while keeping the remaining parameters to their default values in Table 1. Figure 14 plots the results.

The costs of TRA and TNRA increase with r. One notable observation is that the I/O time of TNRA-CMHT rises only marginally for large r. The reason is that the initial result documents emerge only after the short inverted lists have been exhausted, after which further result documents are usually found simply by scanning down the one remaining list. With 1-Kbyte blocks and 8 bytes per $\langle d, f \rangle$ entry, at most one incremental disk I/O is required. This also explains TNRA's behavior in Figure 14(d). Overall, the algorithms demonstrate similar performance to Figure 13.

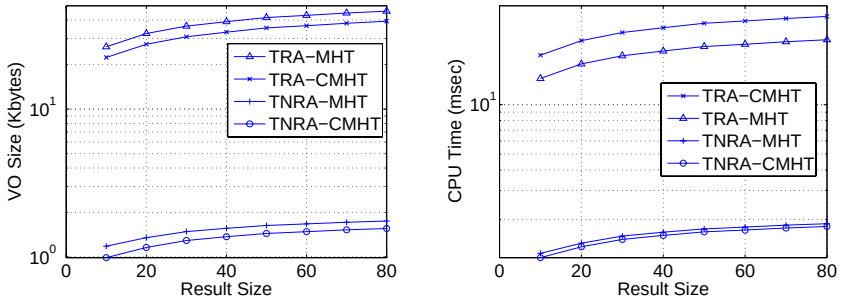
4.4 TREC Queries

Our third experiment uses the TREC workload. The main difference from the synthetic workload is that the TREC queries tend to be longer, and they usually contain some common words. Topic 181 in TREC-3 [29] is an example: "Abuse of the Elderly by *Family Members*, and Medical and Nonmedical Personnel, and Initiatives *Being Taken* to Minimize This Mistreatment." Even after removing stopwords like 'of', 'the' and 'to', the query still contains four terms (highlighted in italic) that appear in more than 10,000 documents each. Figure 15 presents the corresponding results.

In Figure 15(a), the advantage of TRA's earlier termination over TNRA is more pronounced than in previous experiments; the gap here ranges from 10% to 20%. As the TREC queries hit the long inverted lists more frequently, our algorithms are able to read a smaller fraction of the lists as shown in Figure 15(b). The remaining subfigures show similar behaviors to the previous experiments, although the absolute costs are now more than 20 times higher. Notwithstanding that, TNRA-CMHT still achieves subsecond I/O time, below 60 msec user computation time, and less than 50 Kbytes in VO size even for a result size of 80 documents.



(a) Average # Entries Read (b) % of Inverted List Read (c) I/O Time



(d) VO Size (e) CPU Time

Figure 14: Synthetic Workload, Varying Result Size with Query Size = 3

4.5 Discussion on Experiment Results

In all the above experiments, with synthetic and standard TREC workloads, TNRA-CMHT is consistently the clear winner in terms of I/O cost, VO size and CPU cost, due to its ability to effectively prune the longer inverted lists. The results also confirm the effectiveness of our chain-MHT and buddy inclusion techniques in reducing the VO construction and verification costs.

The synthetic workload resembles Web search queries, which tend to contain just a few terms – according to [28], Web search queries include only 2.4 terms on the average, and most users do not look beyond the first page of 10 result documents. Assuming that most queries comprise 3 search terms and require the top 20 matching documents, TNRA-CMHT achieves average query processing and verification times that are below 50 msec and 10 msec respectively, and VOs that are just over 1 Kbyte in size.

In comparison, the TREC workload is representative of natural language queries that are more verbose and contain common words, and of queries that have been expanded through corpus/query analysis or user relevance feedback [31]. Such a workload imposes a higher demand on the underlying query processing and verification mechanisms. Assuming retrieval of the $r = 20$ most similar documents, TNRA-CMHT still achieves around 60 msec I/O time, 32 Kbytes VO size, and 40 msec user verification time. These findings confirm that TNRA-CMHT is a preferred alternative to the conventional PSCAN algorithm, especially for search engines that need to authenticate their query results.

5. CONCLUSION

In this paper, we present the first work for verifying the query results generated by text search engines. Our aim is to enable the users to detect whether their search results indeed contain the most relevant documents, ranked in the right order, and include no spurious entries; in short, whether their search results are the same as what an intact system would produce. We formulate the properties that define a correct search result, map the task of processing a text search query to adaptations of existing threshold-based algorithms, and devise authentication mechanisms for verifying correctness of the results. To the best of our knowledge, this is the first authentication mechanism for the inverted index, a structure that most search engines employ. We experimentally evaluate our techniques and demonstrate their robustness and practicality.

Document retrieval often employs complementary mechanisms to improve effectiveness. For instance, Web search engines may exploit the document metadata or the hyperlink structure among documents to boost the ranking of the authoritative documents (e.g. [4, 11]). Extending our framework to capture such elaborate ranking mechanisms is a promising direction for future work.

6. REFERENCES

[1] R. Baeza-Yates and B. R. Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[2] E. Bertino, B. Carminati, E. Ferrari, B. M. Thuraisingham, and A. Gupta. Selective and Authentic Third-Party Distribution of XML Documents. *IEEE Transactions on Knowledge and Data Engineering*, 16(10), 2004.

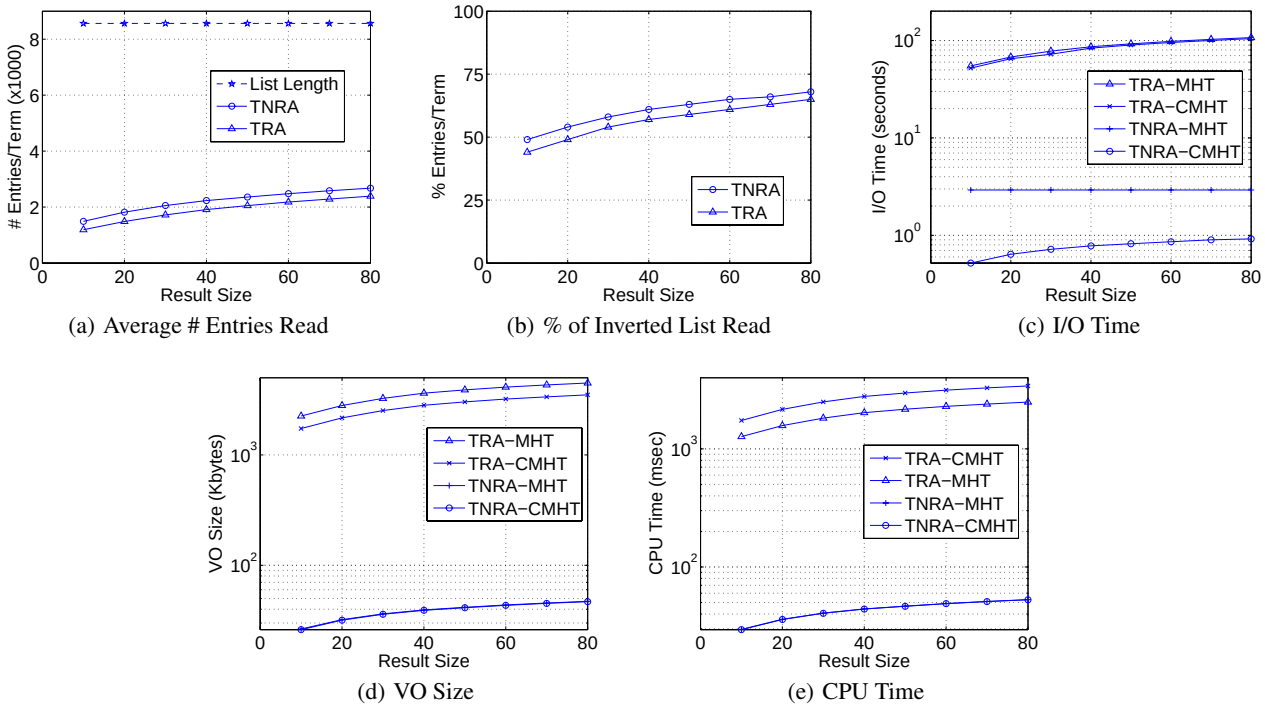
[3] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *EUROCRYPT*, pages 416–432, 2003.

[4] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[5] W. Cheng, H. Pang, and K.-L. Tan. Authenticating Multi-Dimensional Query Results in Data Publishing. In *DBSec*, pages 60–73, July 2006.

[6] W. Cheng and K.-L. Tan. Query Assurance Verification for Outsourced Multi-dimensional Databases. *Journal of Computer Security*, 2008.

[7] D. Comer. Ubiquitous B-Tree. *ACM Computing Surveys*, 11(2):121–137, 1979.



(a) Average # Entries Read (b) % of Inverted List Read (c) I/O Time

(d) VO Size (e) CPU Time

Figure 15: TREC Ad-Hoc Queries

[8] P. T. Devanbu, M. Gertz, C. U. Martel, and S. G. Stubblebine. Authentic Third-Party Data Publication. In *DBSec*, pages 101–112, 2000.

[9] DSS. Proposed Federal Information Processing Standard for Digital Signature Standard. *Federal Register*, 56(169):42980–42982, 1991.

[10] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *Journal of Computer and Systems Sciences*, 66(4):614–656, 2003.

[11] J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999.

[12] F. Korn, B.-U. Pagel, and C. Faloutsos. On the 'Dimensionality Curse' and the 'Self-Similarity Blessing'. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):96–111, January 2001.

[13] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic Authenticated Index Structures for Outsourced Databases. In *ACM SIGMOD*, pages 121–132, 2006.

[14] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios. Proof-Infused Streams: Enabling Authentication of Sliding Window Queries on Streams. In *VLDB*, pages 147–158, 2007.

[15] Lucene. Apache Lucene Search Engine. http://lucene.apache.org/java/docs/.

[16] C. U. Martel, G. Nuckolls, P. T. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A General Model for Authenticated Data Structures. *Algorithmica*, 39(1):21–41, 2004.

[17] R. Merkle. A Certified Digital Signature. In *Crypto*, pages 218–238, 1989.

[18] M. Narasimha and G. Tsudik. DSAC: Integrity for Outsourced Databases with Signature Aggregation and Chaining. In *CIKM*, pages 235–236, 2005.

[19] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying Completeness of Relational Query Results in Data Publishing. In *ACM SIGMOD*, pages 407–418, 2005.

[20] H. Pang and K.-L. Tan. Authenticating Query Results in Edge Computing. In *IEEE ICDE*, pages 560–571, 2004.

[21] H. Pang and K.-L. Tan. Verifying Completeness of Relational Query Answers from Online Servers. *ACM Transactions on Information and System Security*, 11(2):1–50, 2008.

[22] S. Papadopoulos, Y. Yang, and D. Papadias. CADS: Continuous Authentication on Data Streams. In *VLDB*, pages 135–146, 2007.

[23] C. P. Pfleeger and S. L. Pfleeger. *Security in Computing, 3rd Edition*. Prentice Hall, 2003.

[24] R. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. Internet Activities Board, 1992.

[25] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[26] SHA. *Secure Hashing Algorithm*. NIST. FIPS 180-2, 2001.

[27] X. Shen, B. Tan, and C. Zhai. Privacy Protection in Personalized Search. *ACM SIGIR*, 41(1):4–17, June 2007.

[28] A. Spink, D. Wolfram, B. Jansen, and T. Saracevic. Searching the Web: The Public and Their Queries. *Journal of the American Society for Information Science and Technology*, 52(3):226–234, February 2001.

[29] TREC. Text REtrieval Conference. http://trec.nist.gov/.

[30] L. Wang, S. Noel, and S. Jajodia. Minimum-Cost Network Hardening Using Attack Graphs. *Computer Communications*, 29(18):3812–3824, 2006.

[31] J. Xu and W. B. Croft. Query Expansion Using Local and Global Document Analysis. In *ACM SIGIR*, pages 4–11, 1996.

[32] J. Zobel and A. Moffat. Inverted Files for Text Search Engine. *ACM Computing Surveys*, 38(2), July 2006.