Restricted Queries over an Encrypted Index with Applications to Regulatory Compliance

Nikita Borisov[1] and Soumyadeb Mitra[2]

[1] Department of Electric and Computer Engineering
University of Illinois at Urbana–Champaign
nikita@uiuc.edu
[2] Data Domain
soumyadeb@gmail.com

Abstract. Compliance storage is an increasingly important area for businesses faced with a myriad of new document retention regulations. Today, businesses have turned to Write-One Read Many (WORM) storage technology to achieve compliance. But WORM answers only a part of the compliance puzzle; in addition to guaranteed document retention, businesses also need *secure indexing*, to ensure auditors can find required documents in a large database, *secure deletion* to expire documents (and their index entries) from storage once they are past their expiry period, and support for *litigation holds*, which require that certain documents are retained pending the resolution of active litigation.
We build upon previous work in compliance storage and attribute-based encryption to design a system that satisfies all three of these requirements. In particular, we design a new encrypted index, which allows the owner of a database of documents to grant access to only those documents that match a particular query. This enables litigation holds for expired documents, and at the same time restricts auditor access for unexpired documents, greatly limiting the potential for auditor abuse as compared to previous work. We show by way of formal security proofs that our construction is secure and that it prevents reconstruction attacks wherein the index is used to recover the contents of the document. Our experiments show that our scheme can be practical for large databases and moderate sizes of queries.

1 Introduction

Recent regulations require many corporations to ensure trustworthy long-term retention of their routine business documents. The US alone has over 10,000 regulations that mandate how business data should be managed over their entire lifecycle [1]. The focus of most of these regulations (e.g., SEC Rule 17a 4 [2], Heath Insurance Portability and Accountability Act, 1996 (HIPAA), and the Sarbanes–Oxley Act, 2002 [3]). is to ensure that the records are kept immutable during their retention periods and are securely erased at the end of their lifecycle. For example, the HIPAA Section 1173(d) sets security standards for health information, including safeguards to ensure the integrity and confidentiality of

the information and to protect against any reasonably anticipated threats or hazards to the integrity of the information once stored.

This has led to a rush to introduce write-once read-many (WORM) compliance storage devices (e.g., [4–6]). These devices ensure *term immutability* of files. Every file committed to the device has an expiry date either assigned explicitly by the committing application or as a system default. Until the end of its expiration period, the file is read-only and cannot be deleted or altered even by a superuser. A WORM device hence secures critical documents from certain threats originating from company insiders or hackers with administrative privileges. Once a file expires, the device lets the file be deleted, so that external cleanup applications can erase it. Cleanup is required as expired records can often be a liability for the company—e.g., it can be subpoenaed in future lawsuits or regulatory enquiries.

Records are often subject to litigation holds. A record subject to a litigation hold must not be erased from the device until the hold is removed. Litigation holds can be enforced by disabling the automatic deletion (on expiry) of the files, an operation supported by most WORM devices.

Unfortunately, this is not enough to ensure the trustworthiness of litigation holds. Litigation hold requests are often expressed as keyword queries. For example, the judge or auditor might require all email having the words "Martha and Ralph and ImClone" to be retained. Furthermore, the person invoking the hold (judge/auditor) often does not have direct access to the WORM device when the hold request is being enforced—the hold is executed by a company employee (e.g. sysadmin) who may not be trusted in our threat model. In such a scenario, the judge or auditor must be able to verify (at a future point in time) that all the files that are subject to the litigation hold query have been retained.

A simple way to address this problem is to retain the inverted index created over the documents [7] on the WORM device even after the documents have expired. At any future point, the index can be used by the auditor/judge to rerun the litigation hold query to verify that all the resulting documents have been retained. Unfortunately, a keyword index contains too much sensitive information, as the index entries can be used to reconstruct the complete set of keywords in every document, even those which are not subject to the litigation hold.

To address this problem, we design a new type of encrypted index that allows the owner to create a key that gives an auditor access to only those documents that match a particular query. Further, the index prevents reconstruction attacks for any documents that do not match the query. The integrity of the key can be verified, so an auditor checking a litigation hold can be sure that no documents have been removed. Additionally, these per-query keys can be used to restrict auditor access in situations not involving litigation holds, so that auditor access to a WORM device can be restricted to only the documents related to the matter at hand, reducing the potential for auditor abuse.

We implement our scheme and evaluate it using the Enron data set [8], representative of a database at a large enterprise. We show that, although the

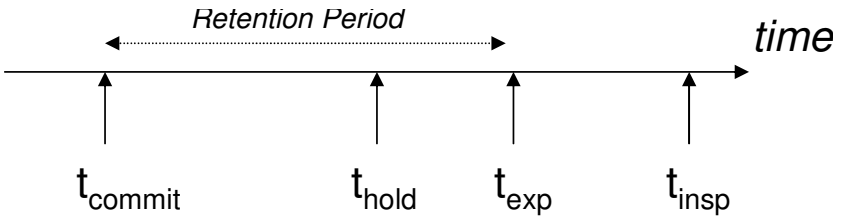


Fig. 1. Record Lifecycle.

encrypted index is both storage- and CPU-intensive, it can be practical for large database sizes and moderate query lengths.

2 Background

2.1 Threat Model for Compliance Records

We use the term *record* to refer to business documents with a fixed retention period, such as financial notes, email, memos, reports, and instant messages. Figure 1 shows the life cycle of record in a typical business environment.

At time t_{commit} a user "Alice" creates a record and commits it to the WORM storage server. We assume that the commit is trustworthy, that is, Alice properly stores the document and correctly sets the expiry time to t_{exp}.

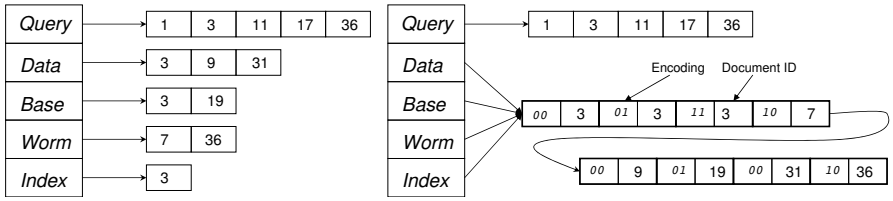
A user Bob (e.g. a judge) subjects the record to litigation hold at time t_{hold} where ($t_{commit} \leq t_{hold} \leq t_{exp}$). In this paper, we consider litigation hold requests that are expressed as keyword queries. An example hold request would be:

> *All documents containing "Martha" and "Ralph" that were created between 06/2002 and 08/2002 must be retained.*

We assume that the user Bob does not have direct access to the WORM device for enforcing the hold request. The hold is implemented by a company employee Mala (e.g. the system administrator)—she must run the litigation hold query to obtain a list of matching documents and ensure that the documents are not deleted on expiry (by disabling their auto-deletion) until they have been inspected by Bob at time t_{insp}.

We however do not trust the user Mala to enforce the hold request properly. For example, Mala might be the CEO who retroactively wants to hide an illegal email conversation she had with her broker about whether to sell stock in her company. Mala cannot alter the email itself, because it is on WORM storage. Mala can however tamper with the litigation hold process. For example, out of all the files that satisfy the litigation hold query, she might retain only the "safe" files while expire the sensitive ones.

At the time of inspection (t_{insp}) Bob must be able to verify the completeness of the litigation hold query result. That is, he should be able to provably verify that all the files satisfying his hold query have been retained. The trivial way



(a) Inverted Index (b) Inverted Index After Merging

Fig. 2. Posting Lists. With each keyword, a *posting list* of documents containing that keyword is stored. After merging, the keyword (or its hash) must also be stored in the posting list.

of ensuring this is to retain the inverted index even after the documents have expired. Bob can run the litigation hold query and obtain the complete list of documents satisfying the query—all those documents must have been retained by Mala.

Unfortunately, the user Bob, although law-abiding, is inquisitive—he may try to reconstruct or extract information about other expired records which do not satisfy his hold request. If the entire inverted index is retained as described above, he can extract information about such documents from the index. Hence, such a scheme is undesirable. Our goal is to develop an index structure which would let Bob obtain the list of documents which satisfy his query without allowing him learn any information about other documents.

2.2 Storage Model

In this paper, we consider a WORM device with a file system interface [6], though our techniques are equally applicable to object-based devices [4]. The interface allows users to create new files and to append to existing files. Appends are required for indexing and can be efficiently supported since the underlying media are magnetic [7]. The append feature should be restricted to the specific storage volume holding the index, to prevent appends to committed files that contain ordinary records.

2.3 Querying and Indexing

The standard query interface for semi-structured and unstructured business records is keyword queries, where a user types an arbitrary set of terms and obtains a list of the documents containing some or all of the terms. Queries can be further constrained by a document creation time interval.

The standard implementation of keyword queries uses an inverted index [9]. As shown in Figure 2.3(a), an inverted index consists of a dictionary of terms and a *posting list* of the identifiers (IDs) of the documents that contain that term. In addition to an ID, each posting list element gives the number of occurrences of that term in the document (its *term frequency*), which is not shown in the figure.

The document IDs are usually assigned in order of document arrival, through an increasing counter. Queries are answered by scanning the posting lists of the terms in the query, thereby obtaining a list of documents having some/all of the keywords. The resulting documents are ranked based on the number of occurrences of the keywords and their relative importance [9].

As explained earlier, we assume that litigation hold queries are also expressed as keyword queries.

2.4 Compliant Inverted Indexes

Because the volume of compliance data is so large and no one wants to wait hours or days for a query answer, index lookup is the only practical record search method. But this means that an adversary can make a record inaccessible by omitting its entries from the index, or altering the index to point to a different version of the record on WORM. Hence, a record can be *logically deleted* or *logically modified* by suitably altering the index structure. To prevent such tampering, the index must be kept on WORM media [7, 10].

An inverted index can be stored on WORM by keeping each posting list in an append-only WORM file. The index can be updated when a new document is added, by appending its document ID to the posting lists of all the keywords it contains. Unfortunately, this operation can prohibitively slow as each append would incur a random I/O. Mitra *et al* have studied the problem of efficiently updating an inverted index on WORM [7] by merging posting lists (into a maximum of as many lists as the number of cache blocks in the storage server [7]). Each merged list contains the union of the document IDs from the individual posting lists that are merged together. A keyword encoding is also stored in each posting element, to identify which keyword in the merged set appears in that document.

The litigation hold techniques that we have developed here treat each keyword posting list as a separate logical unit independent of whether it is stored merged with other keyword posting lists. Specifically, the document IDs are stored encrypted with a keyword specific key. The querier hence would be able to decrypt only those document IDs (in the merged list) which satisfy the query though he would have to run decryption on all the posting elements. Hence, in our case there is no need to store the keyword encoding separately with each posting element.

A compliant index structure must also support deletion. An inverted index contains sufficient information to *reconstruct* the set of indexed terms in a document. For example, from the index in Figure 2.3(a) Bob can learn that document 3 contains the words Query, Data, Base and Index. From the set of words in a document, an adversary can often infer its meaning (e.g., "fire next Thomas week"). Thus it is critical to clean up the index when documents are deleted.

The easiest way to support deletion is to divide the documents into groups called *disposition group* based on their commit times (e.g. all documents committed within a week form one group) and to create a separate index for every disposition group, as shown in Figure 3. The expiry time of a posting list file is

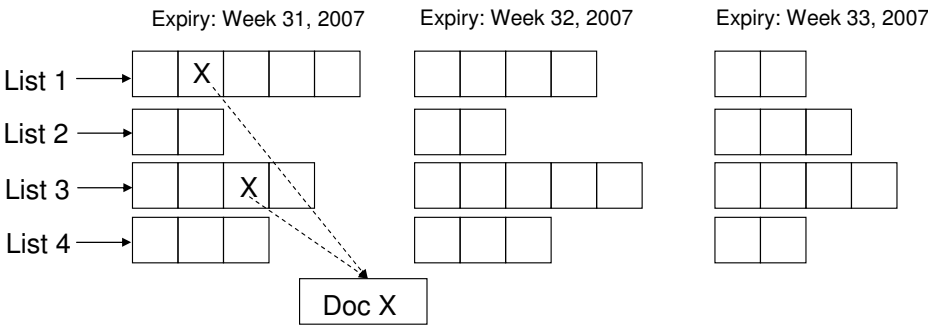


Fig. 3. The split index provides deletion by creating a separate index for each disposition group.

set to that of the corresponding disposition group. Once the disposition group expires, the posting list files are deleted. This deletion scheme is strongly secure: after the posting lists are deleted, the adversary cannot get any information about the deleted documents.

Unfortunately, this approach has very poor query performance. A single-keyword query requires scanning as many posting list files as the number of disposition groups spanned by the query interval. Mitra *et al* have proposed more efficient techniques for supporting secure deletion with better query performance by encrypting the posting list elements with a disposition group specific key and adding noise terms [11]. Our scheme builds on top of this design by replacing the encryption of posting list elements with a new scheme, described in Section 4.

3 Litigation Hold Approaches

In the rest of the paper, we consider a litigation hold request of the following form:

Retain all documents which satisfy the query Q and were committed in disposition interval $\mathcal{D}$

where Q is an arbitrary boolean keyword query involving conjunction of keywords. (A disjunctive query can be implemented as a collection of several separate litigation holds.) All the documents committed in the disposition group $\mathcal{D}$ must be retained in response to this hold request. (although we consider a single disposition interval, the techniques are trivially extendible to multiple disposition intervals).

As shown in Figure 1, Bob invokes the hold request at time t_{hold}. However, the verification of whether all the documents subject to the hold have been retained is done at a later point in time at t_{insp}, where t_{insp} could be after the expiry time of disposition group $\mathcal{D}$. Bob must be able to verify that all the documents in $\mathcal{D}$ satisfying his hold query Q have been retained.

3.1 Trivial Approach

The trivial approach to this problem is to require Mala to retain the entire inverted index for the documents in $\mathcal{D}$ till t_{insp}, along with the documents which satisfy query Q. At t_{insp}. Bob can rerun Q on the index and get a list of documents satisfying Q. These are the documents which should have been retained by Mala.

Since the index is on WORM, Mala cannot tamper with the index. When the index expires, she can delete it and create a new tampered copy of the index. However, Bob will be able to detect this attack by checking the *create-time* of the index.

Unfortunately, this approach also lets Bob get information about documents which do not satisfy his query. An inverted index contains sufficient information to *reconstruct* the set of indexed terms in a document. For example, from the index in Figure 1(a) Bob can learn that document 3 contains the words Query, Data, Base and Index. From the set of words in a document, an adversary can often infer its meaning (e.g., "fire Harriet tomorrow").

This problem can be partially addressed by retaining only the posting lists of keywords involved in the hold query Q. This prevents Bob from reconstructing the contents of arbitrary documents. However, Bob can still learn about documents which do not satisfy his query. For example, even if no documents in $\mathcal{D}$ match the query "Martha and Ralph", Bob may learn that a large number of documents contained the keyword "Martha" during that disposition and infer sensitive information. (Normally, noise techniques proposed by Mitra *et al* [11] prevent such leaks).

Furthermore, a large company may be involved in many separate pieces of litigation and thus be subject to many holds. An auditor involved with one litigation will have access to all of the documents corresponding to all other litigation, which is undesirable.

3.2 Key-Policy Attribute-Based Encryption

A possible way to resolve this problem is to use Key-Policy Attribute-Based Encryption (KP-ABE) [12]. KP-ABE allows one to encrypt a document together with a set of attributes, and then to create a decryption key that allows the decryption of only those documents that satisfy a particular access structure over a document. An access structure can be thought of as a subset of $2^{\mathcal{A}}$, where $\mathcal{A}$ is a set of attributes; documents whose attribute set is in the structure are said to satisfy the structure. As a simple example, a query such as $A \wedge B$ can define an access structure of all attributes sets S such that $\{A, B\} \subset S$.

Consider how an KP-ABE system would be used given attributes A, B, C, D and documents M_1, M_2. Suppose the two documents were encrypted as follows: $E_1 = \text{Encrypt}(M_1, \{A, B, C\})$ and $E_2 = \text{Encrypt}(M_2, \{A, C, D\})$. Then a key K_1 corresponding to the access structure $A \wedge B$ could be used to decrypt E_1 but not E_2. But K_2, corresponding to the access structure $A \wedge (B \vee C)$ could be used to decrypt both documents.

Mapping KP-ABE to our setting, each potential keyword could be assigned to be an attribute. Each index entry for a keyword K occurring in document D would then be encrypted under the set of keywords contained in D. Bob would then be given a key corresponding to the litigation hold query, and he would be able to decrypt only those index entries corresponding to documents matching his query. Other index entries would be indistinguishable from noise terms. In addition, all document contents can be encrypted with KP-ABE using the set of keywords that occur in the document. This would further ensure that Bob could not read any document that does not match his litigation hold, both among the expired documents and those that are currently still within their retention period. As such, Bob would be prevented from abusing his access to the WORM drive to learn company trade secrets or other sensitive information.

However, this scheme has a fatal flaw. A document encrypted under KP-ABE includes, in cleartext, the list of all the attributes that it is encrypted under. This would allow Bob to easily reconstruct document contents from encrypted index entries (or encrypted documents). Thus the situation is even worse than with the trivial approach, since the KP-ABE encryption allows Bob to recover all the keywords in the non-matching documents, rather than only those that are a part of the litigation hold query.

We therefore design a scheme that is based on KP-ABE, but addresses this flaw. It has the advantage that Bob cannot read any document that is not covered by his query, but it also prevents the reconstruction attacks possible with KP-ABE.

4 Encrypted Index with Per-query Keys

4.1 Preliminaries

Our scheme, as with KP-ABE, is based on using bilinear maps:

Definition 1 (Bilinear Map). *Let G_1, G_2, G_T be cyclic groups of same prime order p, and let g_1 and g_2 be generators of G_1 and G_2 respectively. Then $e : G_1 \times G_2 \to G_T$ is a bilinear map if:*

1. *For any a, b, x, y, $e(a^x, b^y) = e(a, b)^{xy}$ (bilinearity)*
2. *$e(g_1, g_2) \neq 1$ (non-degeneracy)*

Efficiently-computable bilinear maps are created based on Weil and Tate pairings over elliptic curves [13, 14]. Usually, $G_1 = G_2$ (or equivalently, there is an efficiently-computable isomorphism between G_1 and G_2 and vice versa.) However, we will use maps where $G_1 \neq G_2$, because we need the following additional assumption:

Assumption 1 (External Diffie–Hellman (XDH)) *Given groups G_1, G_2, G_T and a bilinear map e between them, the External Diffie–Hellman assumption holds if the Decisional Diffie–Hellman problem is hard in G_1. In other words, there is no polynomial time algorithm that can decide whether, given g_1^a, g_1^b, g_1^c, $c = ab$ with a non-negligible advantage.*

It is easy to see that if $G_1 = G_2$, DDH is easy to solve by computing $e(g^a, g^b)$ and $e(g^c, g)$. However, in bilinear maps based on MNT curves [15], where $G_1 \neq G_2$, the XDH assumption is believed to hold [16].

We also require a variant of the standard Bilinear Diffie–Hellman (BDH) assumption [17] for pairings where $G_1 \neq G_2$:

Definition 2 (Asymmetric Bilinear Diffie–Hellman (ABDH)). *Given groups G_1, G_2, G_T, with generators g_1 of G_1 and g_2 of G_2 and a bilinear map e between them, the Asymmetric Bilinear Diffie–Hellman Assumption holds if there is no efficient algorithm that, given as input g_2^a, g_2^b, g_2^c, computes $e(g_1, g_2)^{abc}$.*

Note that due to the existence of an efficiently-computable isomorphism from G_2 to G_1, this assumption is weaker than a variant that uses g_1 rather than g_2.

4.2 Set Up

We consider a collection of documents $\mathcal{D} = \{D_1, \ldots, D_n\}$ and a collection of keywords $\mathcal{K}$. For ease of exposition, we will assume that $\mathcal{K} \subset \mathbb{N}$, since the actual names of the keywords do not matter for our purposes. Let $k : \mathcal{D} \to 2^{\mathcal{K}}$ be the function relating documents to keywords.

In our scheme, each document D_i will be stored under a secret document identifier, id_i. In addition, each keyword will have a corresponding index of encrypted document identifiers. In practice, the WORM drive can contain a hash table mapping identifiers id_i to file names. The secret identifier can also be used to derive an encryption key for the document, making it impossible to read a document without knowing its identifier.

For the purposes of encryption, Alice must generate a secret value s, as well as secrets $s_1, \ldots, s_m$ corresponding to each keyword. These values can be computed from a master secret as, e.g., an HMAC [18] of the keyword name. New secrets will be computed for every disposition period.

Alice will also commit to WORM storage $g_1^{s_1}, \ldots g_1^{s_m}$ and $e(g_1, g_2)^s$. These will be used to verify the integrity of the per-query keys.

4.3 Storage

To store a document D_i, the Alice picks a random identifier d_i. Then, for each keyword $j \in k(D_i)$, Alice adds $g_1^{d_i s_j}$ to the index I_j. Finally, the document identifier is calculated as $id_i = H(e(g_1, g_2)^{s d_i})$, where H is a hash function.

4.4 Lookup by Alice

Alice can easily use the indices to look up a document. For example, to look up a document that has keywords k_1 and k_2, Alice first reads all the identifiers in index I_{k_1} and raises them to the power $s_{k_1}^{-1}$:

$$\left(g_1^{d_i s_{k_1}}\right)^{s_{k_1}^{-1}} = g_1^{d_i}$$

She then performs the same transform on the index I_{k_2} and looks for matches. Given $g_1^{d_i}$ that occurs in both sets, the document ID can easily be computed as $H(e(g_1^{d_i}, g_2)^s)$.

4.5 Per-Query Key

Alice can also generate a key that can be used by Bob without knowing the per-keyword secrets. This key will allow the lookup of documents that match *all* the keywords from a selected set. Given keywords $k_1, k_2, \ldots, k_l$, Alice picks $a_1, \ldots, a_l$, such that:

$$s_{k_1} a_1 + s_{k_2} a_2 + \cdots + s_{k_l} a_l \equiv s \pmod{p}$$

It is easy to see that $a_1, \ldots, a_{l-1}$ can be picked randomly, with a_l being used to solve the remaining equation. Alice then gives $g_2^{a_1}, g_2^{a_2}, \ldots, g_2^{a_l}$ to Bob. Bob can verify that they key is constructed correctly based on the committed values:

$$\pi_{i=1}^{l} e(g_1^{s_{k_i}}, g_2^{a_i}) = e(g_1, g_2)^{\sum_{i=1}^{l} s_{k_i} a_i} = e(g_1, g_2)^s$$

4.6 Lookup Using a Per-Query Key

If Bob is in possession of a key corresponding to keywords $k_1, \ldots, k_l$, he can find documents matching the conjunction of these keywords as follows. First, f each entry $g_1^{d_i s_{k_1}}$ in index I_{k_1}, Bob computes the pairing:

$$e(g_1^{d_i s_{k_1}}, g_2^{a_1}) = e(g_1, g_2)^{d_i s_{k_1} a_1}$$

Similar pairings are computed for each entry in indices $I_{k_2}, \ldots, I_{k_l}$. Then for each tuple of documents $\left(e(g_1, g_2)^{d_{i_1} s_{k_1} a_1}, \ldots, e(g_1, g_2)^{d_{i_l} s_{k_l} a_l}\right)$, Bob computes the product:

$$\prod_{j=1}^{l} e(g_1, g_2)^{d_{i_j} s_{k_j} a_j}$$

If the entries in the tuple all correspond to the same document d_i, then the result will be:

$$\prod_{j=1}^{l} e(g_1, g_2)^{d_i s_{k_j} a_j} =$$

$$e(g_1, g_2)^{\left(\sum_{j=1}^{l} s_{k_j} a_j\right) d_i} = e(g_1, g_2)^{d_i s}$$

Which will make it possible to look up the document. If the d_i's are not equal, the result will be random, and the next tuple should be considered, until all of $|I_1| \cdot |I_2| \cdots |I_l|$ tuples have been checked.

5 Proof of Security

In our scheme, we are concerned with two security properties. First, we want to ensure that the auditor Bob is not able to read any expired document that does not match the particular litigation hold query. Second, we want to prevent the reconstruction attack, which entails hiding the existence of documents that are not matched by a litigation hold.

5.1 Document Secrecy

To ensure document secrecy, the adversary must be unable to determine $e(g_1, g_2)^{ds}$ for a document d for which he does not have a key. We set up the document secrecy game as follow:

Definition 3 (Document Secrecy Game).

Parameters: *The game is parameterized by groups G_1, G_2, G_T, generators g_1, g_2, bilinear map e and keyword set $\mathcal{K}$.*

Keyword selection: *The adversary picks a set of keywords $\gamma \subset \mathcal{K}$ that he wishes to be challenged upon.*

Setup: *The challenger picks secret parameters s, s_i and commits the public parameters $g_2^{s_i}$, $e(g_1, g_2)^s$ to the adversary.*

Challenge: *The challenger picks a document id d and provides $g_1^{ds_i}$ for $i \in \gamma$.*

Learning phase: *The adversary submits to the challenger a polynomial number of queries $\gamma' \subset \mathcal{K}$ such that $\gamma' \not\subset \gamma$. For each γ', the challenger returns the corresponding decryption key $\{g_2^{a_i}\}_{i \in \gamma'}$ such that $\sum_{i \in \gamma'} a_i s_i = s$.*

Response: *The adversary outputs ω, which is his best guess for $e(g_1, g_2)^{sd}$.*

In essence, the game requests that an adversary, given index entries for a document that has keywords γ, must find out the corresponding document ID, while being able to obtain decryption keys for any set of keywords, as long as one of them is not in γ, and hence the keys should not allow the adversary to decrypt the document ID. The advantage of an adversary is defined as:

$$Pr(\omega = e(g_1, g_2)^{sd}) - \frac{1}{|G_T|}$$

Theorem 1 (Document Secrecy).
For any polynomial-time adversary A that has advantage ϵ at the Document Secrecy Game, there is a polynomial-time adversary A' that has advantage ϵ at the Asymmetric Bilinear Diffie–Hellman Game with the same groups.

The proof of this theorem is modeled upon the proof of security of KP-ABE [12], due to the similarities between the two schemes.

Proof. We build a simulator A' that will solve the ABDH problem given A as input. Recall that in the ABDH problem, A' will be given (g_2^a, g_2^b, g_2^c) and A' must compute $e(g_1, g_2)^{abc}$.

A' starts by defining a set of keywords $\mathcal{K}$ and running A to obtain the challenge keyword set γ. A' now defines the secret parameters as follows: It picks $r_i \in_R \mathbb{Z}_p$ for all $i \in \mathcal{K}$. It then defines $s_i = r_i$ for $i \in \gamma$ and $s_i = br_i$ for all $i \notin \gamma$. It also sets $s = ab$.

A' outputs the public parameters: $g_2^{s_i} = g_2^{r_i}$ for $i \in \gamma$, $g_2^{s_i} = \left(g_2^b\right)^{r_i}$ for $i \notin \gamma$, and $e(g_1, g_2)^s = e(g_1^a, g_2^b)$. ($g_1^a$ can be obtained using the isomorphism ϕ from G_2 to G_1.)

A' then outputs the challenge document, with $d = c$:

$$\phi(g_2^c)^{r_i} = g_1^{cr_i} = g_1^{ds_i}$$

A' then runs A, which will proceed to issue queries of the form γ'. Let $\gamma' = \{k_1, \ldots, k_n\}$, and assume without loss of generality that $k_1 \notin \gamma$. A' needs to produce $g_2^{a_1}, \ldots, g_2^{a_n}$ such that $\sum_{i=1}^n a_i s_i = s$.

We first compute $g_2^{a_1'}, \ldots, g_2^{a_n'}$ such that $\sum_{i=1}^n a_i' r_i = a$. We do this by choosing $a_2', \ldots, a_n'$ randomly and computing:

$$g_2^{a_1'} = \left(\frac{g_2^a}{\prod_{i=2}^n g_2^{a_i' r_i}} \right)^{r_1^{-1}}$$

For $i \in \gamma$, we set $a_i = a_i' b$, and compute $g_2^{a_i} = \left(g_2^b\right)^{a_i'}$. (Since $i \neq 1$, a_i' is chosen by A'.) For $i \notin \gamma$, we set $a_i = a_i'$. In both cases, we have that $a_i s_i = (a_i' r_i) b$. Therefore, $\sum_{i=1}^n a_i s_i = ab$.

Finally, A will output ω as a guess for $e(g_1, g_2)^{ds}$. If the guess is correct, $\omega = e(g_1, g_2)^{abc}$ by construction. Therefore, when A' outputs ω, it will enjoy the same advantage as A.

5.2 Reconstruction Attack

To prevent the reconstruction attack, we must ensure that an adversary does not learn anything from the index, other than the identifiers of those documents he should have access to. The noise terms added using the scheme from [11] can be used to ensure that the size of each posting list reveals no information to the attacker. In addition, we want to ensure that given two entries in two posting lists, an attacker should be unable to discover whether those entries correspond to the same or different documents. We therefore set up the reconstruction game as follows.

Definition 4 (Reconstruction Game).

Parameters: *The game is parameterized by G_1, G_2, G_T, g_1, g_2, e, and $\mathcal{K}$.*

Keyword selection: *The adversary picks two keywords $k_1, k_2 \in \mathcal{K}$ that he wishes to be challenged upon.*

Setup: *The challenger defines the private parameters s_i, s and the corresponding public parameters $g_2^{s_i}$ and $e(g_1, g_2)^s$.*

Challenge: *The challenger picks two document IDs, d_0 and d_1 and flips a coin to obtain a bit β. He then sends the adversary $g_1^{d_0 s_1}$ and $g_1^{d_\beta s_2}$, where s_1, s_2 are secrets corresponding to keywords k_1, k_2, as defined in our scheme. If $\beta = 0, d_\beta = d_0$ and the two values belong to the same document, whereas if $\beta = 1$, they belong to different ones.*

Learning phase: *The adversary can perform two types of queries. First, he can pick a set of keywords K' and request to get the encrypted index entries for a document that matches these keywords. I.e. the challenger has to produce $g_1^{ds_i}$ for each $i \in K'$. Second, he can pick a set of keywords K'', such that $\exists k_i \in K$ with $k_i \neq k_1, k_2$ and obtain the decryption key $\{g_2^{a_{k_j}}\}_{k_j \in K''}$. The adversary can perform a polynomial number of either type of query.*

Response: *The adversary then outputs β' as his best guess for β.*

The two types of queries correspond to what Bob might be able to learn given access to the WORM drive. The first query corresponds to finding index entries corresponding to other documents with an arbitrary set of keywords. Such entries may be obtained by using per-query keys for K', or through some external knowledge of what documents may have been stored during disposition period $\mathcal{D}$. The second query corresponds to obtaining keys to litigation holds that include some keyword other than k_1 or k_2. Such keys cannot be used to decrypt the two challenge document IDs and to see if they match a real document.

We can define the advantage of an adversary playing the reconstruction game as:

$$P[\beta' = 0 | \beta = 0] - P[\beta' = 0 | \beta = 1]$$

Theorem 2 (Reconstruction Security). *For any polynomial-time adversary A that has advantage ϵ in the Reconstruction Game, there exists a polynomial-time adversary A' that has an advantage ϵ in solving the Decisional Diffie–Hellman problem in G_1.*

Proof. We prove the theorem by simulation. Given an adversary A, we construct A' as follows.

A' takes as input $g_1, g_1^a, g_1^b, g_1^c \in G_1$, its goal is to determine whether $c = ab$.

A' starts by executing A to obtain the challenge keywords k_1, k_2.

A' sets up the secret parameters by first picking random values r_i for $i \in \mathcal{K}$. It assigns $s_{k_1} = r_{k_1}$, $s_{k_2} = r_{k_2} b$, and $s_i = r_i(1 + b)$ for all others. It also picks a random r and sets $s = r(1 + b)$. Note that s_i are uniformly distributed for all i. It then supplies the public parameters to A.

$$
\begin{aligned}
g_1^{s_{k_1}} &= g_1^{r_{k_1}} \\
g_1^{s_{k_2}} &= (g_1^b)^{r_{k_2}} \\
g_1^{s_i} &= (g_1 \cdot g_1^b)^{r_i} \qquad \text{for all other } i \\
e(g_1, g_2)^s &= e(g_1 \cdot g_1^b, g_2^r)
\end{aligned}
$$

A' then challenges the adversary with $(g_1^a)^{k_1}$ and g_1^c to the adversary. Note that if $c = ab$, then $g_1^c = g_1^{as_2}$ and so $d_0 = d_\beta = a$. Otherwise, $d_\beta = c/b \neq a = d_0$.

For a query of type K' from A, A' picks a random document number d and then computes $g_1^{ds_j}$ for each $j \in K'$ as follows:

- If $j = k_1$, A returns $g_1^{ds_{k_1}}$, since both d and s_{k_1} are known to A
- If $j = k_2$, A returns $\left(g_1^b\right)^{dr_{k_2}} = g_1^{ds_{k_2}}$
- If $j \neq k_1, k_2$, A returns $\left(g_1 \cdot g_1^b\right)^{r_j d} = g_1^{dr_j(1+b)} = g_1^{ds_j}$

For a query of type K'', A' needs to construct a set of a_j such that $\sum_{j \in K''} a_j s_j = s$. If $k_1, k_2 \notin K''$, then we need to simply find $\{a_j\}$ such that $\sum_{j \in K''} a_j r_j = r$. If $k_1 \in K''$ or $k_2 \in K''$, we can assign a_j's by solving a system of two equations. As an example, consider $K'' = \{k_1, k_2, k_3\}$. In this case, we need:

$$a_1 r_{k_1} + a_2 r_{k_2} b + a_3 r_{k_3}(1 + b) = r(1 + b)$$

To find suitable a_j's, we solve the following equations:

$$
\begin{aligned}
a_1 r_{k_1} + a_3 r_{k_3} &= r \\
a_2 r_{k_2} + a_3 r_{k_3} &= r
\end{aligned}
$$

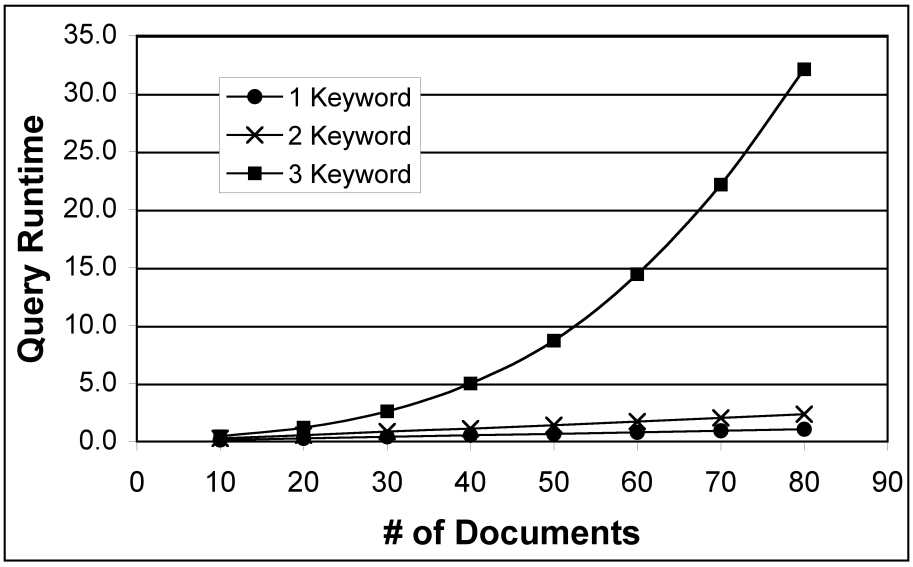
Since all the coefficients in the equations are known to A', and there are three unknowns, A' can solve the equation and produce the correct key. In general, there will be at least two unknowns since in addition to k_1 or k_2, K'' must contain at least one more $k_3 \neq k_1, k_2$, so A' can always solve the equation.

After A has made a polynomial number of queries and has output β, A' outputs that $c = ab$ if $\beta = 0$ and that $c \neq ab$ otherwise. Since A' will be correct whenever A is correct, they both enjoy the same advantage of ϵ.

6 Performance Evaluation

We carried out experiments both on micro-benchmark and on real-world data set to evaluate our scheme. The hardware platform used in our experiments was a 2.4 GHz 64-bit dual core Pentium IV machine running SUSE Linux. (Our tests were single-threaded, so only a single core was in use.)

In our micro-benchmark application, all the posting elements were of the same size and had the same set of document IDs. In other words, all the documents satisfied the query. Figure 4 plots the total runtime as a function of the



(a) Query Runtime

Fig. 4. Runtime overhead of our scheme. x axis plots the number of documents. y axis shows the runtime in msecs.

length of the posting list (number of documents). The different curves correspond to the different number of keywords in the query.

Based on these results, we have developed a model of the executing a query. As described earlier, a query involving q keywords requires the following:

- A bilinear map computation on each posting element of the q query posting lists. The total run time for this (b) is linear in the total length of the posting lists
- A product computation in the group G_T for each possible posting element combination of the q posting lists. The run time (p) for this is proportional to the number of such q posting element combinations, times the time for each such modulo computation (proportional to q). For example, for a 2-keyword query on two posting list of lengths l_1 and l_2, the number of modulo computations is $l_1 * l_2$.

The total runtime (r) is a linear combination of the above two:

$$r = c_b * b + c_p * p$$

By applying least square fitting, we obtained the values of 12.34ms and 0.056ms for c_b and c_p respectively. The runtime predicted by the model was accurate within 3% in the average case and 6% in the worst case.

As a micro-benchmark, we used a collection of 422,000 emails from the Enron email corpus [8]. These emails were exchanged in the 2 year period from January

2000 to December 2001; we omitted the other 50,000 emails in the corpus because they were sprinkled very thinly across the time periods of 1994–2000 and 2002–3. Each email has a metadata tag identifying the sender, receiver, and the time the email was sent. We use this time information to divide the email documents into disposition groups of size 1 week. Noise terms are added to make the posting list same across the disposition groups to some threshold length. Assuming a uniform query model (each keyword equally likely to be queried) the average time for answering a 1, 2,3 and 4 keyword query on each disposition group specific index was 0.8, 1.9, 18.2 and 1034s respectively. The worst case query execution times were 1.2, 3.0, 58.7s and 5461s respectively. This shows that queries with up to three keywords can be practically supported. The cryptographic operations are trivially parallelizable, so as multi-core systems become more common, the CPU time needed to execute queries should be significantly reduced.

Another important consideration is the size of the index. In our experiments, we had set the size of the group G_1 to 320 bits. Each posting element hence is $\frac{320}{8} = 40$ bytes long. Although this inverted index is substantially bigger than the unencrypted inverted index (it is about twice the size of the data corpus), this is not a serious consideration given the low cost of storage. The entire Enron email corpus is only a few GB in size, so the additional cost of the index storage is pennies under current prices.

Index creation requires one bilinear pairing computation for obtaining the document ID and as many modulo exponent computations as the number of keywords in the documents. On our platform, indexing a 100 keyword document takes about 200ms. Once again, multi-core architectures can be used to index documents in parallel.

7 Related Work

The first scheme for searches on encrypted data was proposed by Song et al. [19], with several improvements in following work [20–22]. Golle et al. extended the notion to that of a conjunctive query [23], with improvements in following work [24–27]. The conjunctive query work, while sharing similar goals to ours, uses a model where there are a fixed number of fields, each of which has a single keyword associated. The conjunctive query tests for the presence of a keyword in a particular position. Therefore, they work well for queries on structured fields (e.g., "From: Martha", "To: Ralph"), but not full text indexing of records.

KP-ABE is one of several schemes proposed for attribute-based encryption [28–30, 12, 31, 32]. Much of the work concerns creating efficiently expressing complex policies over attributes. As our construction is similar to KP-ABE [12], it is possible to extend it to support the more general access structures found therein, rather than simple conjunctive queries. However, since our approach requires the enumeration of index entry tuples that may satisfy the query, it would negate the efficiency gains for complex policies. As discussed in Section 3.2, a simple application of attribute-based encryption is insufficient for our needs due

to explicitly stated policies. Kapadia et al. have investigated hidden policies with ABE [33], but they resort to a semi-trusted third party to enforce this property.

8 Conclusion

We have developed a new type of encrypted index. Our index allows the owner to create a keys granting access to only a subset of the documents in the index that match a particular query. We showed how this scheme can be used to address the litigation hold problems of compliance storage, and also to limit potential abuse from auditors. We formally proved the security of our scheme and demonstrated by experiments that it is practical for large databases and moderately-sized queries. Our scheme may have applications beyond compliance storage to other shared databases where people are to be given restricted access based on attributes; it improves upon the Key-Policy Attribute-Based Encryption construction by being able to hide which attributes documents are encrypted under.

Acknowledgments

We would like to thank Ian Goldberg for his advice on the security proof and the anonymous reviewers for their helpful comments. This work was supported by an IBM Fellowship.

References

1. The Enterprise Storage Group, Inc.: Compliance: The effect on information management and the storage industry. `www.enterprisestoragegroup.com` (2003)
2. Securities and Exchange Commission: Guidance to broker-dealers on the use of electronic storage media under the national commerce act of 2000 with respect to rule 17a-4(f). `http://www.sec.gov/rules/interp/34-44238.htm` (2001)
3. Congress of the United States of America: Sarbanes–Oxley act. `http://thomas.loc.gov` (2002)
4. EMC Corpopration: EMC Centera content addressed storage system. `http://www.emc.com/products/systems/centera_ce.jsp` (2003)
5. IBM Corporation: IBM TotalStorage DR550. `http://www-03.ibm.com/systems/storage/index.html` (2006)
6. Network Appliance, Inc.: Snaplock^{TM} compliance and SnapLock enterprise software. `http://www.netapp.com/products/filer/snaplock.html` (2003)
7. Mitra, S., Hsu, W.W., Winslett, M.: Trustworthy keyword search for regulatory-compliant records retention. In Dayal, U., Whang, K.Y., Lomet, D., Alonso, G., Lohman, G., Kersten, M., Cha, S.K., Kim, Y.K., eds.: Conference on Very Large Data Bases, VLDB Endowment (September 2006) 1001–1015
8. Cohen, W.W.: Enron email dataset. `http://www.cs.cmu.edu/~enron` (2005)
9. Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kaufman (1999)

10. Zhu, Q., Hsu, W.W.: Fossilized index: the linchpin of trustworthy non-alterable electronic records. In: ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM (2005) 395–406
11. Mitra, S., Winslett, M., Borisov, N.: Deleting index entries from compliance storage. In Kemper, A., ed.: Conference on Extending Database Technology. (March 2008)
12. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In Wright, R., di Vimercati, S.D.C., eds.: ACM Conference on Computer and Communications Security, New York, NY, USA, ACM (October 2006) 89–98
13. Frey, G., Rück, H.: A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. Mathematics of Computation **62**(206) (1994) 865–874
14. Menezes, A., Okamoto, T., Vanstone, S.: Reducing elliptic curve logarithms to logarithms in a finite field. IEEE Transactions on Information Theory **39**(5) (1993) 1639–1646
15. Miyaji, A., Nakabayashi, M., Takano, S.: New Explicit Conditions of Elliptic Curve Traces for FR-Reduction. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences **84**(5) (2001) 1234–1243
16. Ballard, L., Green, M., de Medeiros, B., Monrose, F.: Correlation-resistant storage via keyword-searchable encryption. Cryptology ePrint Archive, Report 2005/417 (2005) `http://eprint.iacr.org/`.
17. Joux, A.: A one round protocol for tripartite Diffie-Hellman. Journal of Cryptology **17**(4) (2004)
18. Bellare, M., Canetti, R., Krawczyk, H.: Message authentication using hash functions: the HMAC construction. CryptoBytes **2**(1) (Spring 1996) 12–15
19. Song, D., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. IEEE Symposium on Security and Privacy (2000) 44–55
20. Goh, E.J.: Secure indexes. Cryptology ePrint Archive, Report 2003/216 (2003) `http://eprint.iacr.org/`.
21. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. EUROCRYPT (2004)
22. Waters, B., Balfanz, D., Durfee, G., Smetters, D.: Building an encrypted and searchable audit log. Network and Distributed System Security Symposium (2004)
23. Golle, P., Staddon, J., Waters, B.: Secure Conjunctive Keyword Search over Encrypted Data. International Conference on Applied Cryptography and Network Security (June 2004)
24. Ballarci, L., Kamara, S., Monrose, F.: Achieving Efficient Conjunctive Keyword Searches over Encrypted Data. International Conference on Information and Communications Security (December 2005)
25. Park, D., Kim, K., Lee, P.: Public key encryption with conjunctive field keyword search. WISA (2004) 73–86
26. Byun, J., Lee, D., Lim, J.: Efficient Conjunctive Keyword Search on Encrypted Data Storage System. Proceedings of EuroPKI (2006) 184–196
27. Hwang, Y., Lee, P.: Public key encryption with conjunctive keyword search and its extension to a multi-user system. In: Pairing Based Cryptography. Volume 4575 of Lecture Notes in Computer Science., Springer (2007)
28. Ostrovsky, R., Waters, B.: Attribute-based encryption with non-monotonic access structures. [34] 195–203
29. Chase, M.: Multi-authority attribute-based encryption. The Fourth Theory of Cryptography Conference (TCC 2007) (2007)

30. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. IEEE Symposium on Security and Privacy (2007)
31. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Advances in Cryptology–EUROCRYPT. Volume 3494 of Lecture Notes in Computer Science., Springer (2005) 457–473
32. Cheung, L., Newport, C.: Provably secure ciphertext policy ABE. [34] 456–465
33. Kapadia, A., Tsang, P., Smith, S.: Attribute-based publishing with hidden credentials and hidden policies. In Arbaugh, W., Cowan, C., eds.: ndss. (March 2007)
34. Syverson, P., Wright, R., eds.: ACM Conference on Computer and Communications Security. In Syverson, P., Wright, R., eds.: The 14th ACM Conference on Computer and Communications Security, New York, NY, USA, ACM (2007)