Secure Management of Structured Documents

Marcus Hassler
University of Klagenfurt
marcus.hassler@uni-klu.ac.at

Franz Kollmann
University of Klagenfurt
franz.kollmann@uni-klu.ac.at

ABSTRACT

A sophisticated document management system is a fundamental basis to cope with the richness of electronic information. Efficient information retrieval and data security are key concepts which have to be considered early during the system design. Using structured documents in this context has two main advantages: first it can improve retrieval performance and second it allows user tailored document encryption. In this paper we propose a general system architecture for storing, searching, retrieving and securing XML structured documents. The approach relies on a relational database storing the content and structure of documents. Natural language processing techniques provide similarity-based matching of user queries and document elements at different structural levels. In contrast to other approaches, which support either encryption of full documents or no encryption at all, our approach also covers partial encryption of documents.In order to allow element-based encryption within hierarchical organized documents, a method to derive keys from superior to inferior element nodes is proposed. Our model allows the owner of a document to specify which parts of her document have to be encrypted. Besides searching within unencrypted parts of documents, users can also retrieve whole documents. These documents are returned as defined by the owner, thus they may contain encrypted parts. For accessing the encrypted document contents, a user can request the appropriate decryption keys from a license server. To minimize security risks the decryption process itself takes place at the client side only.

Keywords

Document Management, Information Retrieval, Structured Documents, Natural Language Processing, XML Security, Key Derivation

1. INTRODUCTION AND MOTIVATION

The explosion of electronically available data and the need of managing it leads to new approaches for efficient document management systems. The tendency towards structured documents involves further challenges of designing such systems. This raises the question of how large amounts of structured data can be represented, stored, managed, and retrieved automatically. Therefore traditional document management systems have to be adapted to fulfill these needs. An important issue which is often disregarded is how to provide mechanisms to secure these documents at different structural levels for different users. For example an author could be interested in protecting only the source code (e.g. appendix) of his paper from being read by the public. Thus an adequate key management and its integration in document management systems are fundamental aspects.

Especially in the context of electronic documents the term 'structured' has to be defined more precisely. The structure of a document is tightly coupled to the intensions of the author in organizing the text. From the IR point of view this structural heterogeneity is hard to handle efficiently. Hence not only content can be queried, also contextual restrictions in form of structural constraints can be expressed. To cope with this difficulty we propose a mapping of documents onto a common document schema. Within this generic document structure two different kinds of information are distinguished:

- Content, which might be further structured into chapters and sections, refers to what the document is about.
- Metadata, in contrast, refers to additional information describing the content without being part of it.

In order to achieve accurate retrieval results, both kinds of information have to be treated differently. Also clear concepts of searchable and retrievable units are essential for indexing and retrieval. Similarity based matching of elements at all structural levels together with ranking the retrieved elements according to their relevance are key elements of such systems.

Besides IR issues a well designed document management system also has to take care of security concerns. In this context digital rights management is a central topic. To overcome the complexity of rights management systems appropriate languages are developed (e.g. XrML [7]). Nevertheless formalizing and assuring rights in practice are completely different things. The latter is by far the harder part. For example how can a system assure that a specific `mp3` file can be played only three times by each system user? Furthermore the defined rights have to be evaluated by a system component which then (however) allows or not certain rights on documents. Generally, if someone successfully attacks or circumvents (e.g. the system administrator) such a component, the documents behind that rights enforcement logic can be accessed in an unrestricted manner. Because our work aims at protecting parts of documents from reading by unauthorized people, we focus on fine granular reading rights.

IPSI-2005 Conference, November 10-13, 2005, Venice (Italy).

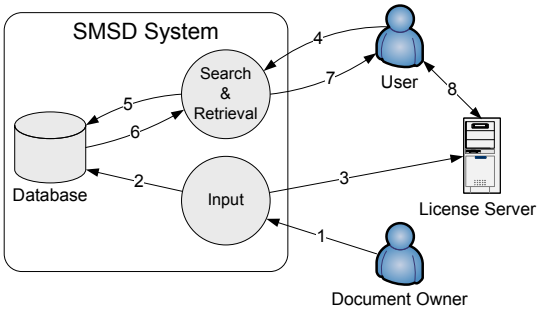

Figure 1: Conceptual model

We propose a Secure Management of Structured Documents (SMSD) system (see Figure 1) where XML documents can be uploaded by document owners and searched/retrieved by users. The input process supports three main tasks: first, it gains as much information as possible about the document (metadata) in order to optimize the performance for search and retrieval later on; second, the input process helps the owner to define and enforce reading rights (by encryption) on different parts of the document; last, it stores the document as specified by the owner with its metadata in the database. The search and retrieval process acts as an interface for users who search within, browse and download documents.

Before the document is submitted to the system, the owner of the document defines reading rights on document parts (step 1). Each node of the plain document is analyzed and a content representation for indexing and retrieval is computed. The system then derives valid keys for all document nodes which the owner has selected to be encrypted. Afterwards the submitted document contents and representations get encrypted according to the specified rights. Next all nodes of the document (contents and representations) are stored either encrypted or plain in a database (step 2). An attack on the system does not bring more information than a standard user gets during a retrieval run. Finally the license server is notified about the new document by using a secure (encrypted and authenticated) data transmission (step 3). This notification consists of the document number, the full structural information and the set of encrypted nodes.

During a search (step 4) only plain representations of document nodes are matched against the query (step 5). Result nodes are ranked according to their relevance and listed to the user (step 6). By selecting a node from the result list its content is displayed. If the user selects a whole document to be retrieved, the system returns it as specified by the owner, encrypted, partially encrypted or not encrypted (step 7). The whole encryption process (if needed) is performed only at the client side. Therefore, the client requests the specific keys for encryption from the license server (step 8).

In our considerations we address the following aspects:

- Minimized key storage
- Key generation as simple as possible
- Easy implementation in practice
- Inheritance of access (reading) rights

In the sequel we briefly review some researches related to our work. Then we clarify the concept of structured documents from our point of view and after that we give an overview about structured document retrieval. Section 5 pinpoints our approach for hierarchical key derivation. Then a general architecture for integrating security issues in the context of structured documents is suggested. Finally the conclusion summarizes the main ideas of this paper.

2. RELATED WORK

In [3] an architecture of a content management server for XML documents stored in their native XML format is suggested. The system is trimmed for large data collections under high load. Indexing and retrieval is restricted on textual data incorporating word and phrase indices. Kunkelmann and Brunelli [16] give requirements for a good content management system emphasizing the importance of metadata. Besides textual content also multimedia information (images, videos) are addressed. Another XML retrieval system developed by Fuhr is HyREX [9, 1]. For query evaluation HyREX relies on the XIRCL language (extended XPath syntax). Different types of metadata support comparison at higher levels (e.g. person names, local closeness). However these works do not cover security issues at all.

To solve the hierarchical access control problem the usage of an encryption function as a one-way function (with trapdoor) was first proposed in [2]. Therein a method to derive keys for hierarchical structured security classes is suggested. In particular a public integer t_i is assigned to each security class U_i with the property that t_i divides all values assigned to its inferior security classes. The secret key K_i for security class U_i is calculated by $K_0^{t_i}$ $(mod\ m)$ where K_0 is the secret key of the central authority and m is the public RSA modulus. Because t_i MOD $t_j = 0$ if $U_i \leq U_j$, t_i grows dramatically with an increasing number of classes.

In [5] a key derivation mechanism for overcoming flexible changes of keys and tree nodes is proposed. This dynamic access control is achieved by a certification authority which updates all public parameters in the system.

In the context of XML documents, some of these flexibilities (multiple parent nodes, dynamical changing documents) are not needed. Because XML is tree-like structured each node (except the root node) has exactly one parent. Since altering the structure of an XML document may lead to key inconsistencies of former document versions, we assume that if a document gets changed (content and/or structure) this leads to a new document with a new document identifier (versioning). Those simplifications allow us to define an easy but effective deriving method (without the need of a certification authority).

3. TAXONOMY FOR STRUCTURED DOCUMENTS

As soon as speaking of structured documents, the question of 'what is structured' and 'how is the structure expressed' is raised. Different authors tend to structure their texts differently, so there is no consistency inherent in a set of documents written by different authors. This structural heterogeneity often leads to inconsistencies and ambiguities, especially in large-scale document management systems. Therefore we introduce a uniform document schema, consisting of a small set of only three structural entities. In

a first step structural ambiguities are eliminated by mapping incoming documents onto our schema. Afterwards optimized data structures, algorithms and storing mechanisms improve indexing and retrieval performance considerably.

3.1 Element typing

Proper retrieval results always depend on a certain level of content interpretation and structural knowledge. This information plays a central role in satisfying the users needs during retrieval. Therefore some nodes, like the gray shaded elements in Figure 2 should be treated more like (meta)data (e.g. the `author`'s name), which might be queried based on Boolean matching model. In contrast, other elements should be handled as full text elements and matched based on their similarity to the query.

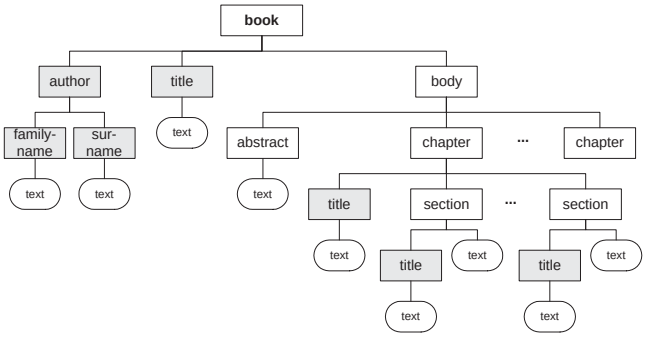

Figure 2: Example document tree

To allow a semantic interpretation of the content of an element, a type hierarchy is proposed by Gövert [10]. An extension of the proposed type hierarchy is depicted in Figure 3. There, types are derivated from a common base element. The first level in the hierarchy corresponds to database supported data types. Thus, they can be used to assign types to the columns of database tables for storing specific element contents. Further types in subsequent levels in the hierarchy are user-defined, having one of the basic database types as ancestor (e.g. `PersonName` is a String).

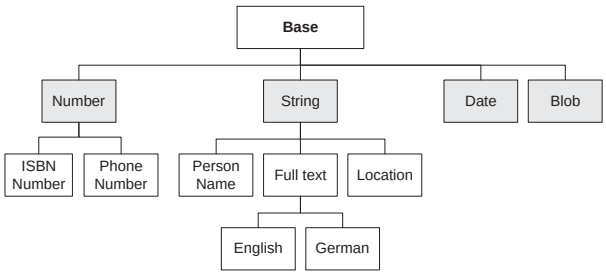

Figure 3: Hierarchical metadata types

In addition to the data types, also predicates for comparison and methods for computation are defined. This allows different treatment of e.g. section titles and full text paragraphs. Whereas titles are probably not full sentences and thus should be treated like keywords, paragraphs might be analyzed in more detail. Predicates also allow more sophisticated similarity based matching of elements of the same type. So documents written by "Albert Einstein" are addressed by a user query stating the author of type `PersonName` as "A. Einstein", whereas "H. Einstein" does not.

3.2 Structured Content

The hierarchical structure for the content of documents is usually covered by terms like chapters, sections and subsections (see Figure 2). To be able to systematically deal with different document sources and XML format specifications efficiently, we introduce a general document format (defined by an XML schema) that consists of only three different main elements (levels): `DOCUMENT`, `SECTION` and `FRAGMENT`.

The `DOCUMENT` element is the root node of all documents. The basic element to structure a documents content is the `SECTION`. Each `SECTION` may contain an arbitrary number of `FRAGMENT`s and/or other `SECTION`s. By this recursive definition, there is no limiting maximum depth for nested structures. To define smallest retrievable units for indexing and retrieval, we use the notion of `FRAGMENT`s. So `FRAGMENT`s define the leaf nodes in our document structure (see Figure 4). With this concept we are able to reflect any tree-like structure within documents.

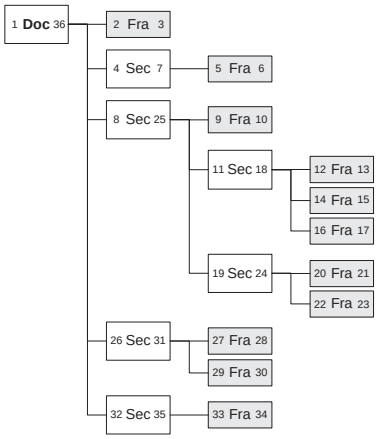

Figure 4: Example document

All three main elements consist of two blocks, a metadata block and a content block. The metadata block contains additional information describing the element and its content. Examples for document metadata are `author`, `year` and `keywords`, section metadata would be the sections `title`. Fragment metadata is used to define its actual content by means of its `content_type`, `language`, and `title` (e.g. figure, table, etc.).

The content block contains the content of the specified element. The contents of `DOCUMENT`s and `SECTION`s are defined as a collection of further sub-`SECTION`s and `FRAGMENT`s. The content of `FRAGMENT`s can be either bytecode (inlined binary information, e.g. figures) or plain text.

Hence a `FRAGMENT` can be understood as basic building block for any kind of content. In this context it acts as a content container for paragraphs, figures, tables, formulas, images, sounds, videos, etc.). The granularity of a `FRAGMENT` depends on how deeply structured a document is. This definition ranges from sentence-level up to the whole content of a logical document structure (e.g chapter, section, subsection, etc.).

Often additional markup within a `FRAGMENT`'s content is needed to support further layout information, mathemat-

ical environments, footnotes and linkage. To cope with such information, the content of a fragment might be substructured to include this markup. But the smallest retrievable unit (index node) remains the whole fragment.

The content block of `DOCUMENT`s, `SECTION`s, and `FRAGMENT`s is not mandatory. This allows us to include contents by using only its metadata information (e.g. if a content is not analyzable by the system). This concept also allows us to incorporate any distributed source of content.

In order to support linkage within documents, two types of links are defined: internal and external links. Internal links are links within the same document (e.g. table of contents, citations, references to figures, tables, etc.). External links refer to other documents (e.g. references, URIs, file paths, etc.).

4. STRUCTURED DOCUMENT RETRIEVAL

Traditionally, content-based retrieval systems rely either on the boolean model or the vector space model (VSM) [4, 21, 20] to represent the (flat) content of documents as a bag of words. Extensions of these models have been proposed, e.g. the fuzzy Boolean model and knowledge-aware models. However, all of these indexing models do ignore the organization of texts and the structure of documents until recently with the advent of "queriable" digital libraries. A precursory work in the direction of structured document retrieval was first proposed in [25, 26], where only fragments of documents are returned to the user in response to his/her query instead of the whole documents. This is actually similar to some extent to passage retrieval.

Structured document retrieval aims at exploiting the document structure to improve retrieval accuracy. One way to structure documents is to use XML markup, where the structure is explicitly defined by a DTD or XML schema. While this structure provides documents with hierarchical levels of granularity, and therefore more precision can be achieved by means of focussed retrieval [14], it does, however, put more requirements on the representation and retrieval mechanisms. With the new generation of retrieval systems, the two aspects, namely the structure and the content, have to be taken into account. To minimally achieve that in presence of a nested structure like chapter-section-subsection-paragraph, traditional information retrieval representation and indexing techniques (e.g. provided by the VSM) have to be adapted to fit the context of structure-aware retrieval. To design such systems, three basic aspects are of high importance:

- **Indexing:** As a first step indexing units, so called index nodes, have to be defined. During retrieval only indexed elements of a document can be retrieved. Index nodes can be defined in two ways: a human marks them explicitly; or all units are considered by the system by a common strategy. In our approach we adopted the second idea to achieve a maximal degree of flexibility.

- **Retrieval:** During retrieval the user can restict the search to certain index nodes. In other words he defines the levels of searched units expressed through their XPaths, e.g. `/Doc`, `/Doc/Sec/Sec`, `//Fra` (dynamic granularity). By default all element levels are considered to be searched. Additionally he specifies the retrieval units, the elements which are returned as a result (by default the same as the searched units). Hence the searched units implicitely define the number of searched elements, and the retrieval units define the desired retrieval granularity, the user himself is able to decide the tradeoff between retrieval quality and retrieval performance.

- **Ranking and result presentation:** Related to indexing, a strategy for ranking the retrieval results has to be defined beforehand. Once ranked, the retrieval results are presented to the user in a way that reflects also the structural level of the retrieved component.

5. HIERARCHICAL KEY DERIVATION

Bruce Schneier classified key management as "the hardest part of cryptography" [23]. Often a careless key management is the main reason why encrypted data get revealed unauthorizedly, although standardized cryptographic mechanisms are used. Why attacking an encryption function if the keys in a key storage can be much easier compromised? In a key management system securing few keys is generally more feasible than protecting many keys. In particular if the keys are expected to increase constantly or even exponentially, the key storage will exceed sooner or later the storage capacity of each security token.

In Section 3 a generic XML document structure was proposed. Structuring documents improves not only retrieval results. It also allows a rights management at different levels in the documents hierarchy. The XML structure also helps to provide an easier rights administration, where rights (i.e. rights for reading) defined at ascendant nodes are inherited by descendants. With this structure it is possible to secure (i.e. encrypt) even parts of documents. For example some authors wants their source code (and main idea respectively) in their document to be readable only by those who pay for it (Figure 5).

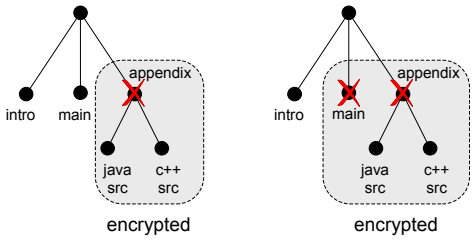

Figure 5: Partial document encryption

In many document management systems rights on documents get managed by the logic which is implemented in the system. For example the logic on the web server shows only that information that a user is allowed to see. But what if the system is hacked and the rights management logic is bypassed? Only cryptographic mechanisms can guarantee a real solution to this problem. Certainly cryptography is not able to solve all problems concerning document rights management. Nevertheless we want at least to break down the 'all-or-nothing encryption' paradigm into partial encryption where the authors decide for themselves which parts of their document have to be encrypted. Surely, if every document gets its own key and individual keys are assigned to nodes

in the structure of a document, the effort to manage such keys becomes significantly high.

Therefore, we propose a method based on derived keys fitting the tree-like structure of XML documents. This reduces the number of secret keys to a minimum: with our approach only one key (master key) has to be stored secretly. All other keys of all documents in the system can be derived from the master key when needed. Because our derivation function is based on a one-way function, it can be public.

In the following we give a conceptual approach for a tree-based key derivation. After that we discuss a practical implementation of the concepts and suggest mechanisms to achieve a good tradeoff between security and feasibility.

5.1 Conceptual design

A key derivation function calculates a key from a master secret and additional parameters. In the literature a key derivation function is often associated with a function $f(ms, s, n)$. f derives a key from a master secret ms and two parameters, a salt value s (pseudo-randomized number) and a number of iterations n. Instead of passwords (which can often be easily attacked by a dictionary attack) pseudo-random numbers for ms are used, thus an extra salt value is not needed. We denote a key derivation function f as

$$f : \{0,1\}^r \times \{0,1\}^s \rightarrow \{0,1\}^t \quad (1)$$

which produces a key k_j of bit length t from a given key k_i of bit length r, and a public constant c of bit length s, given by

$$f(k_i, c) = k_j \quad (2)$$

Because the XML document structure is hierarchical, we need a key derivation functionality which supports hierarchical dependencies. Keys corresponding to a parent node should be more "powerful" than keys belonging to its children. That is why a one-way key derivation mechanism, which allows that keys belonging to children nodes can be derived from keys belonging to their parent but not vice versa, is required. The other way round, deriving a valid key for a parent from any key of its children must be practically impossible (right tree in Figure 6).

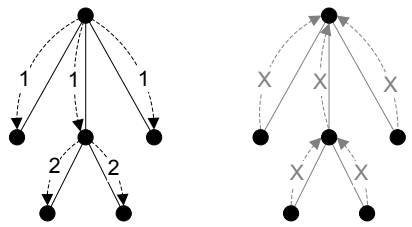

Figure 6: One-way key derivation

Keys belonging to descendant nodes which are more than one level below a given node in the tree are calculated recursively. For example keys assigned to grandchildren nodes are calculated first by deriving keys associated to the children nodes which are parent nodes of the grandchild nodes and afterwards by deriving keys from the children nodes to those of grandchildren nodes (left tree in Figure 6).

In our proposal every XML document gets its unique document identifier *doc_id* and every node in the tree structure of an XML document gets its own node identifier *n_id* (all public). The document key dk of a document can be derived from a master key mk which is secret. An encryption of a document at the root node means that the whole document is encrypted with dk. Only those who obtain the document key (i.e. by buying it from a license issuer) can decrypt the document. From dk all other keys (belonging to any node) in that document can be deduced (left tree in Figure 7). Having a key associated with an inner node instead, only a derivation of keys to descendant nodes of that node is possible. For example in the right tree of Figure 7 only k_3 can be derived from k_1.

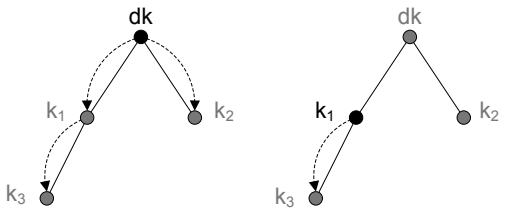

Figure 7: Derived keys in a tree structure

An encryption on tree nodes allows us to prevent whole contents of subtrees from unauthorized access. So structured documents like XML can be encrypted at different nodes in the hierarchy, thus allowing a flexible and fine granular level of encryption. If, for instance, an author of a document only wants the introduction to be read by the public, he selects all other nodes, located at the same level in the tree as the introduction node, to be encrypted (right tree in Figure 5). Then the document management system derives the appropriate keys as described below and encrypts the selected nodes in the document before storing it. In order to have a minimal key storage the produced keys do not have to be stored by the document management system, instead they can be discarded. Later on, if a decryption is required, the keys can be easily recalculated by using the master key which is a randomized bit sequence that only the document management system knows. We assume that there exist appropriate mechanisms to keep the master key secret (i.e. in tamper-resistant hardware).

The required property of a one-way key derivation can be achieved by using a cryptographic hash function. Generally a cryptographic hash function H maps from any arbitrary bit sequence to a fixed size bit sequence l:

$$H : \{0,1\}^* \rightarrow \{0,1\}^l \quad (3)$$

This means that a hash function maps an infinite set to a finite set and thus it is obvious that such a function can not be bijective. So there exist different inputs where the hash function produces identical outputs which is called a collision.

A hash function with two inputs, a key and an arbitrary bit sequence, is called a Message Authentication Code (MAC). To generate a document key dk, a MAC function M is used by taking the master key mk and the unique document identifier as its input:

$$dk = M(mk, doc\_id) \quad (4)$$

Due to the nature of hash functions strict uniqueness of generated keys cannot be guaranteed by such a function. However, we can enlarge the range of the hash function (i.e by using a hash function which produces longer hash values) to reduce the probability of a collision significantly. Moreover, to avoid attacks on the master key it is advantageous to use a further hash function within M (see section 5.2).

The document key which is associated with a document can be calculated on demand and if necessary all other keys in the document can be derived from dk in a similar way. Each key k_j at node level $j \neq 0$ (except the root key) with node identifier n_j_id can be produced by the key k_i, which is associated with its parent node i (see Figure 8), using a MAC function where $k_0 = dk$:

$$k_j = M(k_i, n_j\_id) \tag{5}$$

5.2 Practical design considerations

Common hash functions like SHA1 [13], MD5 [19] or RIPEMD-160 [6] are iterative algorithms. In general, they expand the input m (by padding) such that they can divide m into a sequence of n blocks $m_1, \ldots, m_n$, where each block $m_i \in \{0,1\}^l$ has a fixed length l. In each hash iteration only one input block is being processed. The hash value produced in the i-th iteration only depends on the i-th input block and the hash value from the previous iteration: $h_i = H(h_{i-1}, m_i)$. The starting hash value h_0 used in the first round is defined by some constants.

Designing a document key derivation function as $H(k \,||\, doc\_id)$ is not a very good choice since due to the iteration functionality this allows a length extension attack [8]. Consider having a document key dk to a specific document. In some circumstances (if the input extended by the algorithm fulfills exactly the last block without padding) it is then possible to produce further document keys without knowing k by simply extending the document identifier (the old document identifier is a prefix of the new one). This is possible because of the iterative design of a hash function. By a given document key $dk = h_n$ (the output of the last hash iteration), an attacker can easily calculate $H(h_n \,||\, doc\_id\_extension)$ which returns a new valid document key. If there is no padding, this would be equal to $H(k \,||\, doc\_id \,||\, doc\_id\_extension)$. Even in the other case, calculating other valid document keys is possible because an attacker can perform the padding manually by extending the document identifier properly. To achieve this, the last block has to be fulfilled (by padding) and afterwards any document identifier extension can be appended. This produces a valid document key with $doc\_id' = doc\_id \,||\, padding \,||\, doc\_id\_extension$.

Replacing the arguments and designing the key derivation function as $H(doc\_id \,||\, k)$ is a better approach. Nevertheless a key recovery attack could be forced on that design (although the effort for a realistic key recovering can be illusive). Placing a hash function within a hash function can bring extra security, whereas the order of arguments has no drastic effects on security issues. Composing a key derivation function as $H(H(K \,||\, m))$ leads to the design of HMAC [15] which is a MAC function that uses a nested hash function and two constants a and b:

$$HMAC(m,k) = H(k \oplus a \,||\, H(k \oplus b \,||\, m)) \tag{6}$$

Because the design of HMAC has been approved over several years, the few extra costs in performance could be worth considering a key derivation function according to that design.

For security reasons it can be an advantage to use different hash functions. As an example, the SSL handshake protocol makes use of SHA in the MD5 function to generate a session key. Such a design reduces the risk in case when one hash function will be weakened seriously or even broken in the future. Actually there exist some attacks which can weaken, even though marginal, MD5, RIPEMD-160 and recently also SHA1 [24]. Although RIPEMD-160 is not as fast as SHA or MD5, in our opinion it has an elaborated one-way design which seems to be appropriate for our key derivation function. For the document key derivation function we suggest to use an extended version of RIPEMD-160 namely RIPEMD-256 (denoted as R), which produces 256 bit hash values and within R we propose to use SHA-256 (referred as S):

$$dk = R(mk \oplus a \,||\, S(mk \oplus b \,||\, doc\_id)) \tag{7}$$

The input of this function consists of the master key (e.g. 256 bit), the two HMAC constants and the unique document identifier. From the document key ($dk = k_i$) each key k_j belonging to child node j of the root node can be calculated by following derivation function:

$$k_j = R(k_i \oplus a \,||\, S(k_i \oplus b \,||\, n_j\_id)) \tag{8}$$

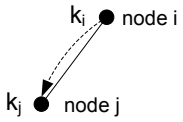

Figure 8: Derived keys on nodes

Applied recursively this function generates all keys in the subtree with its root node i.

Because all structural elements (document identifiers and node identifiers), constants (a and b) and this derivation algorithm are publicly known by the system users, someone having a key to a node can derive all further keys in the subtree rooted at that node. For the encryption and decryption on nodes we propose to use the Advanced Encryption Standard (AES) [18]. The suggested derivation function produces keys with 256 bit length which are supported by the AES.

5.3 Further remarks

To derive keys it is not necessary to take a MAC function. Also an encryption function (symmetric or asymmetric) can be used to achieve similar properties. This can sometimes be an advantage and sometimes it is a drawback. We designed our key derivation function such that it generates pseudo-random keys, not more. We do not need special functionality like a "secret" way back (trapdoor in RSA) from a key belonging to a child node to the key belonging to its parent node. In our design this should be simply not possible. Alternatively a symmetric encryption could be used as a key derivation function (i.e. $E(k_i, n_j\_id) = k_j$). Nevertheless

a hash algorithm guarantees that the output has always a fixed bit length (here 256 bit). In a symmetric encryption scheme the output size depends on the input size.

Instead of using a recursive derivation function, one can design a key derivation function such that a key belonging to any node in the tree is derived form the document key directly. An often required feature in context of a flexible rights management is inheritance of rights. The above mentioned approach does not support inheritance of rights.

6. SYSTEM ARCHITECTURE

This chapter describes the realization of our Secure Management of Structured Documents (SMSD) approach. This covers the processing of new docuemnts (indexing), rights management (encryption and license server notification), storage, search and retrieval.

6.1 Indexing of documents

The indexing process of a document starts with an event-driven parsing (e.g. a SAX parser [22]), where elements and their contents are identified and stored in corresponding database tables. Afterwards the representations of natural language text elements are calculated. This is performed by a natural language analysis, transforming the raw texts to term frequency vectors.

Indexing of element nodes starts at the leaf nodes, representing and storing the content in the database. Every new representation stored in the database updates global term statistics used for term weighting during retrieval accordingly. The same operations are carried out if documents are re-indexed or removed from the system.

Inner node representations are calculated by simply merging the sets of feature terms and summing up their term frequencies. Depending on whether an actual inner node is defined to be calculated in advance, the representation is calculated and stored persistently. This reduces search times during retrieval, but increases the size of the database.

Our natural language processing (NLP) implementation is based on abstract subtask components. Taking advantages of the the modularity aspect, different implementations of the same component are used and selected during runtime. This design enables us to support various implementations of tokenizers, taggers, stemmers, etc. in parallel, which are instantiated on demand. Our prototype also involves ready made-components like the tagger, and the stemmer. Hence our system is capable of multi-language NLP and parameter-based tailored representation computation.

NLP involves several subtasks containing tokenization, tagging, term extraction, stemming, filtering and term frequency calculation (see Figure 9).

1. *Tokenizer:* identifies words and sentences. A text is transformed into a list of sentences, where each sentence consists of a list of tokens. To avoid any misinterpretation of sentence borders our tokenizer supports single- and multi-tokens, token typing, abbreviation detection and special format lookup.

2. *Tagger:* assigns grammatical word categories (tags) to words. This process is based on dictionary lookup, lexical rules and contextual patterns.

3. *Term extractor:* only nouns and verbs are taken into account to represent the content of a text. These are recognized by using their tag information. Experiments showed that including adjectives and adverbs does not improve retrieval results.

4. *Stemmer:* reduces words to their roots (mainly by eliminating ending characters). This step supports similarity matching of different forms of the same word. Also the number of terms is reduced considerable, enabling faster comparison calculation.

5. *Filter:* by means of a stop list containing undesirable words, only meaningful words among those remaining are retained. In the first version of our system the stop list was constructed manually.

6. *Term frequency calculation:* simply counts how often a stemmed term occurs within a document element.

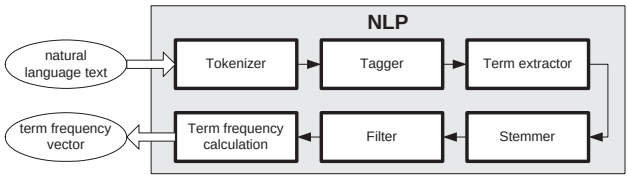

Figure 9: NLP component

One has to note that to this point the representations only contain term frequencies, not term weights. To achieve a maximum degree of dynamic indexing, term weighting itself is accomplished during the retrieval phase only.

6.2 Rights management

During the input process the submitter of a document decides which parts of the document have to be encrypted (see Section 5.1). In particular the submitter selects which nodes in the XML structure have to be encrypted by the system. Therefore our approach realizes user defined reading rights on nodes in XML documents. Document parts, which are marked to be encrypted, are stored encrypted in the database (as described in Section 6.3). Before storing, each document node in the system gets a unique document identifier. The keys used for the encryption are derived as described in Section 5. Inheritance of reading rights within the tree structure is provided by the property of the recursive key derivation function. To achieve maximal flexibility on the choice of using the encryption methods and block modes, we suggest to use XML Encryption [17] on our mapped XML documents conforming to our general XML document format. Besides, XML Encryption supports variable encoding formats (e.g. Base64, UTF16). For the encryption and decryption we propose to use the Advanced Encryption Standard (AES) [18]. The AES is relatively fast and as prevailing encryption standard it has gained special security investigations.

Along with storing the documents in the database, the system generates a notification which contains the document identifier, the tree structure of the document and a list of nodes which are encrypted. After mutual authentication between the system and the license server, the notification is encrypted and sent to the license server. Later on, if users request documents from the search and retrieval component of the system, desired documents are returned and transferred

to them as specified during the input process (encrypted, partially encrypted or not encrypted).

In order to access encrypted document parts, appropriate keys for decryption are necessary. There is no user administration at all: users/clients who have the appropriate keys can decrypt the encrypted XML parts (if there are any). Others can obtain them from the license server by sending a request containing the document and node identifier. After requesting, the license server indicates which requirements has to be fulfilled (i.e. how many to pay) in order to obtain the desired keys. For an independent accomplishment, the license server also keeps the master key safe. With the master key the license server can calculate the proper key to any node in any document stored in the system.

6.3 Storage

The way documents are stored in IR-related systems plays a decisive role on their performance and thus, on their acceptance. Especially in the context of structured documents efficiency during retrieval of elements of any granularity must be provided. Therefore, we adopted a relational database approach for storing the documents.

We depart from the idea of *pre-* and *post-order* introduced in [11, 12]. The goal is to accelerate the access to various structural neighbors of each element in the structure of a document that are: descendants, ancestors, and siblings. The access efficiency comes from the fact that *pre-order* and *post-order* descriptors are unique for a given document and, therefore, can be used conjointly with the ID of that document as primary key in the mapped relational schema. *Pre-* and *post-order* support non-recursive ancestor/descendant detection and access. Figure 4 shows how *pre-order* (number to the left of an element) and *post-order* (number to the right of an element) are inserted.

A structural entry is described by the tuple ($docID$,pre-$order$,$post$-$order$,$parentID$,$tagID$,$pathID$,$encM$,$encC$) (see Table 1). The root element has $pre\text{-}order = 1$ and $parentID = 0$ (no parent node) per definition. The $tagID$ is included for fast name lookup and access. For the sake of performance we added the elements full path (without positional information) $pathID$ to circumvent recursive path generations by using the $parentID$ relation.

$encM$ and $encC$ are both boolean values which indicate whether the set of metadata ($encM$) and/or the content ($encC$) is/are stored encrypted. In case of content encryption all available representations are also encrypted. Hence the encryption of nodes is based on inheritance, the same $encM$ and $encC$ values are assigned to all descendant nodes. Thus their metadata and contents are also encrypted the same way as before, but with another key derived from the parent node (see Section 5).

Inserting documents into the database is linear in time and size of the input source. By using an event-based parsing framework for XML documents like SAX [22], we are guaranteed to need only very limited temporary space during storing [11].

The content of XML nodes is stored in separate tables. Hence not all structural elements consist of content themselves, content is not mandatory. As defined leaf nodes, `FRAGMENT`s are consisting of content, so they have to be inserted in the content table (see Table 2). Other element contents of inner nodes (`SECTIONS`s and `DOCUMENT`s) can be calculated on the basis of the contained leaf nodes. Additionally these dynamically generated contents can also be stored in the contents table. This leads to redundancy but increases performance during retrieval. Two independent content tables are maintained: one for storing the plain content and another one for storing the content representation.

doc	*pre*	*post*	*par*	*tag*	*path*	*encM*	*encC*
d_1	1	36	0	`Doc`	`/D`	0	0
d_1	2	3	1	`Sec`	`/D/F`	0	0
d_1	4	7	1	`Sec`	`/D/S`	0	0
d_1	5	6	4	`Fra`	`/D/S/F`	0	0
d_1	8	25	1	`Sec`	`/D/S`	0	0
d_1	9	10	8	`Fra`	`/D/S/F`	0	0
d_1	11	18	8	`Fra`	`/D/S/S`	1	1
d_1	12	13	11	`Fra`	`/D/S/S/F`	1	1
d_1	14	15	11	`Fra`	`/D/S/S/F`	1	1
...	...	...	...	...	...	...	...
d_2	1	70	0	`Doc`	`/Doc`	1	0

Table 1: Structural entries

doc	*pre*	*cdata*
d_1	2	`To begin with the number ...`
d_1	5	`The content of document ...`
d_1	9	`To improve performance ...`
d_1	12	`xxxxxxxxxxxxxxxxxxxxxx ...`
...	...	...

Table 2: Content entries (plain content)

To improve performance user-defined metadata is treated differently. Therefore, the database supports three levels of metadata sets, each for one of the three main elements (`DOCUMENT`, `SECTION`, and `FRAGMENT`). Instead of having several structural entries with the same number of content entries, a single row in a metadata table is used to store pooled meta data.

Hence all structural elements (even `DOCUMENT`s) are uniquely identified via $docID$ and $pre\text{-}order$, three different tables defined by ($docID$,pre-$order$,$meta_1$,$meta_2$,...,$meta_n$) hold all metadata (see Table 3). The basic reason of having only one `SECTION` metadata set is that all `SECTION` elements (chapters, sections, subsections, etc.) are assumed to have a quite homogenous set of meta elements (e.g. `title`). Although this may lead to some '$NULL$' values (unstated elements) in the database, the a whole set of metadata can be accessed by a single database select statement. This simplifies database like querying of metadata and accelerates access.

doc	*pre*	*id*	*author*	*title*	...
d_1	1	`K728`	`R. Smith`	`In the summer`	
d_2	1	`xxxxx`	`xxxxx`	`xxxxxxxxxxxxxxxx`	
...	1	...	...	...	

Table 3: Metadata entries (document level)

Each element is uniquely identified by its document ID

(*docID*) and element identifier (*pre-order*). Associated with this pair are metadata sets (of all three main elements) and content information (plain content and representation). Both metadata and content entries are optional. Additional kinds of representations (e.g. semantic concepts, figure representations, etc.) can easily be integrated in this architecture.

By using the document identifiers (resp. node identifies) to derive keys on the fly, we do not need to store any keys explicitly. This reduces key management efforts significantly because the system does not have to take care of securing new keys and locating old ones.

6.4 Search and retrieval

As soon as a query is sent to the system, the query text is analyzed the same way the document elements were during indexing, also resulting in a term frequency vector for the query (see Figure 9). Besides the text of the query several other parameters may be defined:

- The **search units** define which element levels are to be matched against the query. This parameter has a deep impact on retrieval time, hence it defines the number of elements that are to be weighted and compared to the query.
- The **retrieval unit** indicates which elements of the result set are to be returned to the user.
- A maximum **number of retrieval results** parameter can be used to truncate ranked retrieval results at a certain level. A similar effect can be achieved by stating a **minimum similarity** parameter of the retrieved elements, thus eliminating results below a given similarity threshold.

During retrieval only specified (search units) and not encrypted document nodes are compared to the user query. In order to calculate a similarity measure between an element and the query term frequency vector the terms are first mapped onto a common term space, consisting of all terms known to the system. Then, both vectors (element and query) are weighted according to the standard vector space model [21]. Two weighted term vectors e and q are matched using the cosine similarity, given by Equation 9.

$$sim(e,q) = \frac{\sum_{i=1}^{k}(w_{i,e} \cdot w_{i,q})}{\sqrt{\sum_{i=1}^{k} {w_{i,e}}^2 \cdot \sum_{i=1}^{k} {w_{i,q}}^2}} \quad (9)$$

Here, $w_{i,q}$ and $w_{i,e}$ reflect the weight of term i in the query vector q, respectively in the element vector e; k denotes the total number of terms. More similar weighted vectors will result in a higher cosine similarity. For further details see [4].

6.5 Ranking and result presentation

Ranking is the task by which similar units are retrieved ordered by their relevance. The ranking process is impacted strongly by the desired granularity (retrieval unit). For example, if the user specifies the document level (focus), the system should return only relevant documents. This can be done by measuring the similarity of the query to all elements of the document. The similarity of that document to the query can then be either the similarity of the document's content (root node) generated recursively from the descendants or the maximum similarity of the documents units.

After all desired elements are matched against the user query, the similarity values are used for ranking. The ranked results are truncated at the maximum number of retrieval results. Furthermore remaining elements not meeting the minimum similarity criteria are removed.

We think of presenting the results to the user as a sorted list of elements in decreasing order of their rank, where a single result node can be selected. Because searched nodes are not encrypted their plain content is directly displayed to the user. If a user downloads a whole document, it is delivered as specified by the document owner during submission. The keys for decryption can be requested from the license server.

7. CONCLUSION

In this paper we presented a new approach for partial encryption of XML documents in the context of structured document retrieval. Document owners have the possibility to define parts of documents (XML nodes) as to be encrypted separately.

For structural disambiguation and performance reason incoming documents are mapped onto a general document schema. This schema consists only of three main elements, namely `DOCUMENT`s, `SECTION`s and `FRAGMENT`s. After submission, documents conforming to this schema are indexed. During this procedure all element nodes are analyzed and their representations are calculated. According to the owner's rights specification marked elements (metadata, contents and representations) are encrypted.

For the encryption and decryption process approved mechanisms like the AES can be used. In order to get the appropriate keys for encryption and decryption of document parts, a two level hashing approach is applied: first a document key is derivated from the master key and a document identifier using some hash function. Second we use the same concept recursively to derive keys from a parent node to keys belonging to its children nodes. Starting with the document key, all keys for all nodes in that document can be computed. This approach reduces key storage size to a minimum (master key), and allows inheritance of rights within subtrees.

Afterwards the document is stored in the database (encrypted, partially encrypted or not encrypted). In the next step the license server, which provides the keys for the client side decryption, is notified about the new document. During retrieval, document nodes are matched against a query and a relevance measure for each node is calculated. This also involves similarity based matching of metadata according to their types. All results are listed to the user in decreasing order of their relevance. By selecting a result, the system displays the content of that document node. Additionally a user can download a whole document. In this case it is re-generated from the database and sent to the user. Thus it may also include encrypted parts as intended by the owner. In order to access these parts a user can request decryption keys from the license server. The decryption process is shifted completely to the client side (user).

Because all documents in the system are stored encrypted as defined by the owner, bypassing the system logic and attacking the storage directly is useless. In addition to this

key management is kept as simple as possible, and no kind of user management is needed. Furthermore the system is freed of evaluating any rights at all.

8. REFERENCES

[1] Mohammad Abolhassani, Norbert Fuhr, Norbert Gövert, and Kai Grossjohann, *HyREX: Hypermedia retrieval engine for XML*, Research report, University of Dortmund, Department of Computer Science, Dortmund, Germany, 2002.

[2] S. Akl and P. Taylor, *Cryptographic Solutions to a Problem of Access Control in a Hierarchy*, ACM Transactions on Computer Systems, ACM, 1983, pp. 239–248.

[3] T. Arnold-Moore, M. Fuller, A. Kent, R. Sacks-Davis, and Neil Sharman, *Architecture of a Content Management Server for XML Document Applications*, 1st International Conference on Web Information Systems Engineering (WISE00), IEEE, 2000.

[4] Ricardo Baeza-Yates and Berthier Ribeiro-Neto, *Modern information retrieval*, Addison Wesley, ACM Press, New York, Essex, England, 1999.

[5] T.S. Chen, Y.F. Chung, and C.S. Tian, *A Novel Key Management Scheme for Dynamic Access Control in a User Hierarchy*, Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC04), IEEE, 2004.

[6] H. Dobbertin, A. Bosselaers, and B. Preneel, *The hash function RIPEMD-160*, `www.esat.kuleuven.ac.be/~bosselae/ripemd160.html`, 1996.

[7] *eXtensible rights Markup Language*, `http://www.xrml.org/`, 2005.

[8] N. Ferguson and B. Schneier, *Practical cryptography*, Wiley Publishing, 2003.

[9] Norbert Fuhr, Norbert Gövert, and Kai Grossjohann, *HyREX: Hyper-media retrieval engine for XML*, Proceedings of the 25th Annual International Conference on Research and Development in Information Retrieval (New York) (Kalervo Järvelin, Micheline Beaulieu, Ricardo Baeza-Yates, and Sung Hyon Myaeng, eds.), ACM, 2002, Demonstration, p. 449.

[10] Norbert Gövert, *Bilingual information retrieval with HyREX and Internet translation services*, Cross-Language Information Retrieval and Evaluation (Heidelberg et al.) (Carol Peters, ed.), Lecture Notes in Computer Science, vol. 2069, Springer, 2001, pp. 237–244.

[11] Torsten Grust, *Accelerating XPath location steps*, Proceedings of the 2002 ACM SIGMOD international conference on Management of data, ACM Press, 2002, pp. 109–120.

[12] Djoerd Hiemstra, *A database approach to content-based XML retrieval*, INitiative for the Evaluation of XML Retrieval (INEX). Proceedings of the First INEX Workshop. Dagstuhl, Germany, December 8–11, 2002 (Sophia Antipolis, France) (Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas, eds.), ERCIM Workshop Proceedings, ERCIM, March 2003, pp. 111–118.

[13] P. Jones, *RFC 3174 - US Secure Hash Algorithm 1 (SHA1)*, `www.faqs.org/rfcs/rfc3174.html`, 2001.

[14] G. Kazai, M. Lalmas, and T. Rölleke, *Focussed structured document retrieval*, Proceedings of the 9 Retrieval (SPIRE 2002), Springer, 2002, pp. 241–247.

[15] H. Krawczyk, M. Bellare, and R. Canetti, *RFC 2104 - HMAC: Keyed-Hashing for Message Authentication*, `www.faqs.org/rfcs/rfc2104.html`, 1997.

[16] T. Kunkelmann and R. Brunelli, *Advanced Indexing and Retrieval in Present-day Content Management Systems*, Proceedings of the 28th Euromicro Conference (EUROMICRO02), IEEE, 2002.

[17] Joseph Reagle, *W3C XML Encryption WG*, `http://www.w3.org/Encryption/2001/`, 2001.

[18] Vincent Rijmen, *The block cipher Rijndael*, `http://csrc.nist.gov/CryptoToolkit/aes/`, 2004.

[19] R. Rivest, *RFC 1321 - The MD5 Message-Digest-Algorithm*, `www.faqs.org/rfcs/rfc1321.html`, 1992.

[20] G. Salton, *The smart retrieval system - experiments in automatic document processing*, Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

[21] G. Salton and M. E. Lesk, *Computer evaluation of indexing and text processing*, Journal of the ACM **15** (1968), no. 1, 8–36.

[22] *SAX (Simple API for XML)*, `http://sax.sourceforge.net`, 2005.

[23] B. Schneier, *Applied cryptography*, John Wiley and sons, 1996.

[24] X. Wang, Y. L. Yin, and H. Yu, *Collision Search Attacks on SHA1*, `http://cryptome.org/sha1-attacks.htm`, 2005.

[25] Ross Wilkinson, *Effective retrieval of structured documents*, Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval, Springer-Verlag New York, Inc., 1994, pp. 311–317.

[26] Ross Wilkinson and Justin Zobel, *Comparison of fragmentation schemes for document retrieval*, Overview of the Third Text REtrieval Conference (TREC-3). NIST Special Publication 500-225, National Institute of Standards and Technology (Donna K. Harman, ed.), 1994, pp. 81–84.